.TEA „POLITEHNICA" TIMIŞOARA
)E AUTOMATICĂ ŞI CALCULATOARE

Viorel COIFAN

CONTRIBUȚII LA OPTIMIZAREA PERFORMANȚEI RAPORTATĂ LA DEPENDABILITATE A SISTEMELOR MULTIPROCESOR CU MEMORIE PARTAJATĂ

Dizertație pentru obținerea titlului de doctor inginer

Coordonator:
Prof. dr. ing. Crișan STRUGARU

Timişoara, 2001

PREFAȚĂ

În această dizertație sunt tratate subiecte referitoare la ameliorarea *performanței* sistemelor de calcul multiprocesor în funcție de *dependabilitate*.

Dependabilitatea este un termen introdus de Laprie în 1985 [Lapr85] ce înglobează conceptele de *fiabilitate*, *disponibilitate, siguranță în funcționare*, *mentenabilitate*, *performabilitate* și *testabilitate*. Aceste concepte sunt exemple de măsuri utilizate pentru cuantificarea dependabilității unui sistem de calcul.

Capitolul 1, intitulat Introducere, este rezervat prezentării unor probleme fundamentale în abordarea domeniului proiectării sistemelor de calcul multiprocesor. Sunt trecute în revistă subiecte legate de tendințele arhitecturale în dezvoltarea acestor sisteme, reliefându-se strânsa legătură dintre dezvoltarea tehnologică și cea arhitecturală.
Convergența arhitecturilor paralele exprimă o realitate ce se referă la structura de comunicație, spațiul de adrese partajate, transferul de mesaj, etc. Vechile taxonomii, ca de exemplu cea a lui Flynn, încep să aibă o arie restrânsă de aplicabilitate în comparație cu noile abordări din domeniu.
Coerența și consistența memoriilor cache reprezintă probleme fundamentale în definirea și proiectarea sistemelor multiprocesor și, prin urmare, tratarea lor reprezintă subiecte privilegiate.
Rețelele de interconectare asigură într-un grad ridicat nivelul de performanță al acestor sisteme. *Reducerea latenței* și *mărirea benzii de trecere* reprezintă obiective ce sunt permanent în atenția proiectanților acesor rețele.
Toleranța la defect este prezentată atât ca și problematică majoră în proiectarea sistemelor multiprocesor, cât și ca subiect privilegiat în analiza de performanță.

În Capitolul 2, intitulat Analiza arhitecturilor sistemelor de calcul MIMD, sunt dezvoltate subiecte care contribuie la o corectă analiză a arhitecturilor sistemelor multiprocesor din clasa MIMD. Ele se referă la granulația de paralelism, latența memoriei, rețele din interconectare, mecanisme de sincronizare a proceselor.

Capitolul 3, intitulat Analiza de performanță a sistemelor de calcul din clasa MIMD, este dedicat analizei de performanță a sistemelor de calcul din clasa MIMD.

După prezentarea tehnologiei de bază pentru evaluarea performanțelor rețelelor de interconectare (RIN) se analizează performanțele principalelor tipuri de rețele: cu *magistrală unică*, cu *magistrală multiplă*, *crossbar* şi *multinivel*. Sunt prezentate expresiile analitice ale *lățimii de bandă*, *capacității de trecere*, *probabilității de acceptanță*. În continuare se face o analiză comparativă a facilităților hardware oferite de aceste tipuri de rețele.

Analiza de performanță a arhitecturilor sistemelor multiprocesor (SMP) cu *memorie partajată* şi a celor cu *memorie partajată distribuită* urmează în mod firesc. Se prezintă *in extenso* mecanisme şi protocoale de menținere a coerenței memoriilor cache şi modele de consistență a memoriilor.

Concluziile acestui capitol se referă în special la relația dintre caracteristicile arhitecturale şi mecanismele de menținere a coerenței.

În Capitolul 4 care este intitulat Analiza arhitecturilor sistemelor de calcul din clasa MIMD tolerante la defect se prezintă o analiză a arhitecutrilor sistemelor de calcul MIMD *tolerante la defect*. Terminologia prezentată ca şi enumerarea obiectivelor toleranței la defecte au drept scop reliefarea conceptului de dependabilitate şi a tehnicilor de evaluare a ei.

Detectarea defectelor prin duplicare şi comparare, prin diagnoză cât şi controlul erorii în memoriile de mare viteză sunt enumerate în continuare.

Strategiile de recuperare din defect, crearea şi exploatarea punctelor de verificare, recuperarea prin *rulare înapoi* şi recuperarea prin *avansare* sunt analizate distinct, atât pentru sistemele multiprocesor cu memorie partajată cât şi pentru cele cu transfer de mesaje.

Reconfigurarea sistemelor multiprocesor constituie un paragraf distinct. Sunt prezentate şi analizate tehnici de reconfigurare pentru diverse tipuri de RIN, insistându-se în mod particular pe reconfigurarea RIN *multinivel*. În continuare sunt concluzionate câteva aspecte referitoare la proiectarea RIN-MN tolerante la defect.

Analiza şi *modelarea dependabilității* pentru sistemele multiprocesor cu memorie partajată sunt prezentate în Capitolul 5.

Sunt descrise modele de fiabilitate şi tehnici de evaluare a fiabilităţii pentru diferite sisteme multiprocesor bazate pe *magistrală*, *crossbar*, RIN-MN, *hipercub*, etc. Sunt deduse expresiile ce definesc fiabilitatea de terminal pentru diferite arhitecturi.

Modelele de disponibilitate sunt prezentate, utilizând lanţuri Markov, evidenţiindu-se contribuţia ratelor de defectare respectiv a ratelor de reparare ale diferitelor componente.

În continuare sunt discutate câteva modele de dependabilitate raportate la performanţă, introducându-se parametri ca şi *disponibilitatea lăţimii de bandă*, *performanţa acumulată la un moment dat* şi *performabilitatea*.

Modelarea disponibilităţii pentru un multiprocesor cu memorie partajată orientat în jurul unei RIN-MN reprezintă un subiect care a fost tratat cu atenţie. Sunt prezentate şi analizate subsistemele procesor-memorie, RIN-MN în diferite configuraţii cu 16, 64 de noduri şi în final un model generalizat.

Capitolul 6, intitulat Modelul experimental, este destinat prezentării unui model experimental constituit dintr-un sistem multiprocesor cu memorie partajată distribuită. El este bazat pe o reţea de interconectare, de tip magistrală multinivel. Se prezintă structura sistemului şi se scot în evidenţă avantajele utilizării acestui tip de reţea în comparaţie cu o reţea de interconectare clasică. Se dezvoltă modelul analitic al acestui sistem şi apoi se calculează parametrii de performanţă ca şi: *factorul de utilizare a sistemului*, *timpul de răspuns, fiabilitatea*, etc. Simularea funcţionării a fost efectuată sub PROTEUS, un simulator dezvoltat la MIT.

Capitolul 7, intitulat Concluzii generale, prezintă concluziile lucrării de dizertaţie, contribuţiile autorului şi tendinţe viitoare de cercetare.

Prezenta lucrare de dizertaţie a fost elaborată sub îndrumarea competentă şi exigentă a conducătorului ştiinţific, dl. prof. dr. ing. Crişan STRUGARU. Autorul îi mulţumeşte în mod deosebit pentru sprijinul permanent acordat, pentru generozitatea şi răbdarea pe care a manifestat-o în toată această perioadă.

Conceptul de dependabilitate, ca şi un criteriu de analiză a performanţei sistemelor multiprocesor, i-a fost sugerat autorului de prof. dr. ing. Mircea VLĂDUŢIU cu care acesta a purtat numeroase discuţii interesante referitoare la proiectarea şi analiza sistemelor tolerante la defect. Colaborarea cu domnia sa a constituit un privilegiu pentru care autorul îi mulţumeşte şi pe această cale.

Lucrarea elaborată beneficiază de referinţe bibliografice la zi. Selectarea bibliografiei, obţinerea în timp util a informaţiei necesare, contactele cu colegii din universităţi de prestigiu ca şi: Texas A&M University, University of Minnesota, Princeton University şi Stanford University ar fi fost imposibile fără sprijinul colegial şi cordial acordat de prof. dr. ing. Petru ELES de la Universitatea din Linköpings (Suedia), fost student şi coleg al autorului. Îi mulţumesc încă odată pentru generozitatea şi prietenia sa care mă onorează.

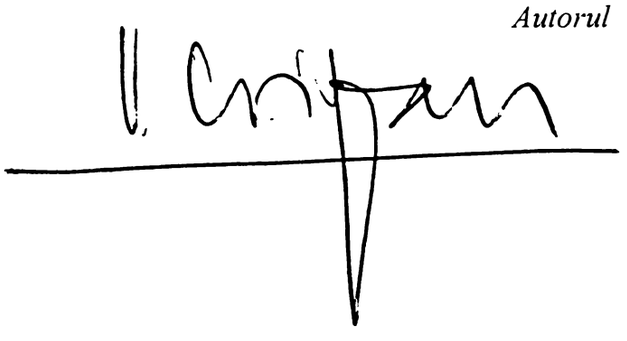
Dactilografierea, prezentarea grafică şi în general toate facilităţile oferite de o tehnoredactare modernă şi profesionistă au fost asigurate de dl. ing. Norbert HANIGOVSZKI căruia autorul îi mulţumeşte pentru răbdarea dar şi pentru ingeniozitatea dovedită în mod constant.

Pe tot parcursul elaborării acestei lucrări, autorul a beneficiat de prietenia şi de sprijinul moral constant al colegilor prof. dr. ing. Octavian PROŞTEAN şi prof. dr. ing. Ştefan HOLBAN de la Facultatea de Automatică şi Calculatoare.

În fine, dar nu în ultimul rând, autorul îşi manifestă întreaga sa gratitudine faţă de familia sa – mamă, soţie şi copii – care l-a înţeles şi l-a sprijinit în toată această perioadă, renunţând de multe ori la micile bucurii cotidiene pe care le-ar fi binemeritat din plin.

Mulţumesc tuturor, care prin cuvinte, fapte şi atitudini au contribuit la finalizarea acestei dizertaţii.

Autorul



Familiei mele

CUPRINS

Pag.

1. INTRODUCERE 1

1.1 Tendințe arhitecturale 1

1.2 Convergența arhitecturilor paralele 1

1.2.1 Arhitectura de comunicație 3

1.2.2 Spațiul de adresă partajat 3

1.2.3 Transferul de mesaj 9

1.2.4 Convergența 11

1.3 Coerența şi consistența memoriilor cache 12

1.3.1 Modele de consistență 14

1.3.2 Coerența memoriilor cache..... 15

1.3.3 Relația între modelele de consistență şi coerența memoriilor cache 17

1.3.4 Scheme de asigurare a coerenței memoriilor cache 17

1.3.5 Factori ce afectează mecanismele de menținere a coerenței memoriilor cache 20

1.4 Rețele de interconectare 20

1.4.1 Caracteristici 20

1.4.2 Performanță de comunicare de bază în RIN 22

1.5 Toleranța la latență 27

1.5.1 Cerințe fundamentale, beneficii şi limitări ale toleranței la latență 29

1.6 Toleranța la defect 32

1.6.1 Implicații ale toleranței la defect în arhitecturile MIMD ... 34

2. ANALIZA ARHITECTURILOR SISTEMELOR DE CALCUL DIN CLASA MIMD 35

2.1 Aspecte generale 35

2.2 Granulația de paralelism 40

2.3 Latența memoriei şi rezolvarea ei 42

2.3.1 Comutarea de proces 43
2.3.2 Utilizarea arhitecturilor LOAD/STORE 43
2.3.3 Coerența memoriilor cache 44
2.4 Rețele de interconectare 58
2.4.1 Terminologie 58
2.4.2 Sistemul multiprocesor cu memorie partajată orientate pe magistrală 64
2.4.3 Rețele de interconectare *statice* 71
2.4.4 Rețele de interconectare *dinamice* 75
2.5 Mecanisme de sincronizare a proceselor la arhitecturi din clasa MIMD 94
2.5.1 Moduri de sincronizare 94
2.5.2 Sincronizarea bazată pe variabile partajate 96
2.5.3 Sincronizarea bazată pe transfer de mesaje 102
2.6 Concluzii 106
3. ANALIZA DE PERFORMANȚĂ A SISTEMELOR DE CALCUL DIN CLASA MIMD 109
3.1 Analiza de performanță la RIN 109
3.1.1 Terminologie de bază utilizată în evaluarea performanțelor RIN 109
3.1.2 Analiza performanțelor magistralei unice 111
3.1.3 Analiza performanțelor magistralei multiple 115
3.1.4 Analiza performanțelor RIN *crossbar* 125
3.1.5 Analiza performanțelor RIN *multinivel* 128
3.1.6 Analiza comparativă a performanțelor RIN de tip *crossbar*, *magistrală unică* şi *multinivel* 130
3.2 Analiza de performanță a sistemelor de memorie în arhitecturile SMP din clasa MIMD 135
3.2.1 Arhitecturi cu memorie partajată centralizată 138
3.2.2 Arhitecturi cu memorie partajată distribuită 152
3.2.3 Modele de consistență a memoriei 164
3.3 Concluzii 173
4. ANALIZA ARHITECTURILOR SISTEMELOR DE CALCUL DIN CLASA MIMD TOLERANTE LA DEFECT 177
4.1 Terminologie 177

4.1.1 Obiectivele toleranței la defect la sistemele multiprocesor 180
4.1.2 Parametri de evaluare a dependabilității 181
4.2 Toleranța la defect prin utilizarea redundanței 185
4.2.1 Toleranța la defect prin utilizarea *redundanței statice* 185
4.2.2 Toleranța la defect prin utilizarea *redundanței dinamice* .. 192
4.3 Detectarea defectelor în sistemele multiprocesor 193
4.3.1 Detectarea defectelor prin *duplicare* şi *comparare* 193
4.3.2 Detectarea defectului utilizând *diagnosticarea* şi *tehnici de codificare* 195
4.3.3 Controlul erorii în memoriile de mare viteză 196
4.4 Strategii de recuperare pentru sistemele multiprocesor 199
4.4.1 Recuperarea din defect prin *rulare înapoi* 208
4.4.2 Recuperarea din defect prin *rulare înainte* 216
4.5 Reconfigurarea în sistemele multiprocesor 226
4.5.1 Reconfigurarea în SMP-MP bazate pe *magistrală* 227
4.5.2 Reconfigurarea rețelelor de tip *crossbar* 228
4.5.3 Reconfigurarea rețelelor *hipercub* 229
4.5.4 Reconfigurarea în rețelele *plasă* 236
4.5.5 Reconfigurarea în rețelele *arbore* 238
4.5.6 Reconfigurarea în rețelele *multinivel* 241
4.6 Concluzii 260
5. MODELAREA DEPENDABILITĂȚII PENTRU SISTEMELE MULTIPROCESOR CU MEMORIE PARTAJATĂ 263
5.1 Etapele preliminare în descrierea modelului 263
5.2 Modele de fiabilitate..... 265
5.2.1 Fiabilitatea de tip *terminal*..... 265
5.2.2 Fiabilitatea de tip *multiterminal*..... 267
5.2.3 Fiabilitatea *bazată pe sarcină*..... 268
5.2.4 Fiabilitatea de tip *rețea*..... 272
5.3 Modele de disponibilitate 272
5.3.1 Sisteme orientate pe *crossbar*..... 273
5.3.2 Sisteme orientate pe magistrală..... 273
5.3.3 Sisteme orientate pe RIN-MN..... 273

5.4 Modele de dependabilitate raportate la performanță 275

5.5 Modelarea disponibilității utilizând descompunerea de sistem pentru SMP-MP, orientate pe RIN-MN 279

5.5.1 Descrierea sistemului 279

5.5.2 Tehnica de modelare 280

5.5.3 Un model generalizat 290

5.6 Concluzii 294

6. MODELUL EXPERIMENTAL 297

6.1 Arhitectura SMP-MP utilizat ca şi model 297

6.2 Structura rețelei de interconectare de tip RMM 301

6.2.1 Algoritm de rutare pentru RMM 315

6.2.2 Analiza de performanță a RMM 321

6.2.3 Rezultate numerice şi discuții 332

6.2.4 Toleranța la defect şi fiabilitatea RMM 340

6.3 Protocoale de menținere a coerenței 344

6.4 Simularea pilotată de execuție 352

6.4.1 Construcția simulării 352

6.4.2 Interfața cu rețeaua 353

6.4.3 Parametrii de simulare 354

6.4.4 Protocolul de menținere a coerenței şi sincronizarea 355

6.4.5 Programe de evaluare a performanțelor 355

6.4.6 Rezultate ale simulării 356

6.5 Efectul politicilor de management al memoriei 358

6.5.1 Algoritm pentru determinarea amplasării paginilor bazată pe acces 360

6.6 Concluzii 361

7. CONCLUZII GENERALE 363

ANEXA 1 373

ANEXA 2 379

ANEXA 3 383

BIBLIOGRAFIE 389

LISTĂ DE TABELE

TABEL		Pagina
I	Rezumatul caracteristicilor hardware a trei RIN	111
II	Rezultatul analizelor RIN *crossbar*	127
III	Sumarul facilităților hardware a trei RIN	131
IV	Scalarea componentelor de calcul, de comunicare şi a raportului	138
V	Mesajele transmise în noduri pentru menținerea coerenței	156
VI	Ordonările impuse de diferite modele de consistență	169
VII	Condiții de defectare	204
VIII	Patru scheme diferite cu *rulare înainte*	219
IX	Tabelul de căutare a conexiunilor	285
X	Tabelul nodurilor de comutare	291
XI	Lungimile de cale pentru diferite direcționări	320
XII	Numărul de destinații pentru diferite dimensiuni ale RIN	328
XIII	Dimensiunea optimă a unui comutator pentru eficacitatea costului	339
XIV	Utilizarea procesului şi timpul de răspuns la RMM şi RMB	340
XV	Parametrii de simulare	354
XVI	Caracteristici ale aplicațiilor utilizate în evaluare	356
XVII	Latențele medii ale mesajelor utilizând RMM şi RMB	357

LISTĂ DE FIGURI

Figura		Pagina
1.1	Numărul de procesoare în SMP-MP comerciale	2
1.2	Lățimea de bandă a SMP-MP disponibile pe piață	2
1.3	Model tipic de memorie pentru programe paralele cu memorie partajată	4
1.4	Extensia unui sistem prin adăugarea de module procesor	5
1.5	Scheme de interconectare tipice în SMP-MP	6
1.6	Organizarea unui SMP-MP scalabil organizat NUMA	8
1.7	Un model de abstractizare a transferului de mesaj la nivel de utilizator	10
1.8	Un exemplu de problemă de coerență a memoriei cache	16
1.9	Comportamentul la saturație pentru o rețea de interconectare tipică	26
1.10	Limitări şi beneficii ale toleranței la latență	30
1.11	Oferta de SMP în funcție de dependabilitate / performanță	33
2.1	Relația dintre carecteristicile algoritmilor şi arhitecturile SMP	36
2.2	SMP-MP din subtaxonul MIMD-T	37
2.3	SMP-MD din subtaxonul MIMD-L	37
2.4	SMP-MP cu arhitectură tip "sală de dans"	37
2.5	O clasificare a SMP paralel/distribuite	39
2.6a	SMP-MP cu memorii cache private	46
2.6b	SMP-MP cu memorii cache partajate	47
2.6c	SMP-MP cu memorii cache private şi partajate	47
2.7	SMP-MP cu magistrală unică utilizând tehnica *snooping*	47
2.8	Diagrama de stări pentru un protocol *invalidare de scriere*	49
2.9	Taxonomia schemelor de asigurare a coerenței la memoriile cache	51
2.10	Condiții pentru un acces la un bloc cache perimat	57
2.11	Sumarul mecanismelor de asigurare a coerenței memoriilor cache mazate pe *autoinvalidare*	58

2.12	Clasificare a rețelelor de interconectare – RIN	60
2.13	Structura unui nod de interconectare la RIN	61
2.14	Exemple de RIN *statice*	62
2.15	Exemple de interconectări	63
2.16	SMP-MP orientat pe magistrală	65
2.17	Structuri de memorii ierarhizate utilizate pentru optimizarea traficului	68
2.18	SMP-MP cu *magistrală multiplă*	71
2.19	RIN *arbore*	72
2.20	Hipercub cu 16 procesoare	74
2.21	Comutatoare	76
2.22	Stările unui comutator 2 X 2	77
2.23	Comanda unei intrări la un comutator 2 X 2	78
2.24	Conexiuni în cubul de ordin 3	79
2.25	Interconectare *amestec perfect*	80
2.26	Inerconectare *butterfly*	81
2.27	Interconectare *bit oglindit*	82
2.28	Rețea Illiac	83
2.29	Interconectare bazată pe *deplasare*	83
2.30	Schema bloc a unei RIN *dinamică*	86
2.31	RIN ce realizează funcția *amestec perfect*	86
2.32	RIN *crossbar*	88
2.33	Memorii multiport	88
2.34	Schema bloc a unei RIN *dinamice*	89
2.35	RIN *omega* ce conectează o intrare cu o ieşire	90
2.36	Situație coflictuală într-o RIN *omega*	91
2.37	RIN *delta* pentru n=2	92
2.38	Clasificarea topologiilor RIN	93
2.39	Procese partajând o structură de date D	95
2.40	SMP-TM cu 8 noduri PM organizate în cub boolean	103
2.41	Structura proceselor Occam şi canalul lor de comunicare	104

3.1	Structura timpului de procesare	112
3.2	Saturația magistralei	114
3.3	Raportul între numărul de procesoare în așteptare și cele active	115
3.4	Structura unui SMP-MP cu magistrală multiplă	116
3.5	Structura unei magistrale multiple	117
3.6	Comportarea asimptotică a parametrului *BW*	122
3.7	Schema bloc a unui arbitru pentru magistrala multiplă	123
3.8	Structura registrului de stare	124
3.9	Un model cu șiruri de așteptare pentru SMP-MP *crossbar*	128
3.10	P_A funcție de talia sistemului	133
3.11	*BW* funcție de *p* pentru un sistem sincron	134
3.12	P_u funcție de *p*	134
3.13	Structura de bază a unui SMP-MP cu ierarhie de memorie	136
3.14	Arhitectura de bază pentru un SMP-TM cu memorie partajată distribuită	136
3.15	Problema coerenței pentru o locație de memorie cache	139
3.16	Protocol de invalidare în cazul unei magistrale cu monitorizare	142
3.17	Protocol de *actualizare la scriere*	142
3.18	Mecanism de asigurare a coerenței	144
3.19	Protocol cu *invalidare de scriere* pentru MCH cu rescriere	145
3.20	Diagrama de stare a coerenței MCH	146
3.21	Variația ratelor de eșec în funcție de numărul de procesoare	148
3.22	Variația ratelor de eșec în funcție de talia memoriei cache	148
3.23	Rata de eșec în funcție de talia blocului	149
3.24	Traficul pe magistrală în funcție de talia blocului	149
3.25	Componentele ratei de eșec în funcție de talia memoriei cache	150
3.26	Componentele ratei de eșec în funcție de talia blocului	150
3.27	Rata de eșec în funcție de talia memoriei cache	151
3.28	Evoluția traficului în porțiunea de sistem de operare	151
3.29	Arhitectura SMP-TM cu directori atașați nodului	154
3.30	Diagrama tranzițiilor de stare pentru un bloc cache	158

3.31	Diagrama tranziţiilor de stare pentru directori la o memorie cache individuală	159
3.32	Rata de eşec în funcţie de numărul de procesoare	162
3.33	Scăderea ratelor de eşec pe măsura creşterii taliei memoriei cache	162
3.34	Rata de eşec în funcţie de talia blocului într-un SMP cu 64 de procesoare şi 128 Kbyte memorie cache	163
3.35	Creşterea numărului de bytes per referinţă de dată în funcţie de creşterea taliei blocului	164
3.36	Un exemplu de consistenţă secvenţială	165
3.37	Reducerea numărului de ordine impuse pe măsura relaxării modelelor	170
3.38	Performanţa modelelor de consistenţă relaxate	172
4.1	Arborele dependabilităţii	179
4.2	Un sistem SMR	186
4.3	Agrementul Bizantin	189
4.4	Agrementul Bizantin după efectuarea algoritmului	190
4.5	Un Agrement Bizantin imposibil	190
4.6	Exemplu de mapare a grafurilor duplicate	195
4.7	Crearea de puncte de verificare bazate pe procesor	201
4.8	Tehnici tolerante la defect pentru copierea în cascadă	203
4.9	Conceptul de bază pentru crearea unui PV virtual	205
4.10	Maparea unei singure pagini	206
4.11	Prima referire după un PV	207
4.12	Pagină referită anterior	207
4.13	O diagramă de timp pentru un calcul distribuit	211
4.14	Tăieturi consistente şi inconsistente	212
4.15	Un exemplu de mesaj *orfan*	213
4.16	Un exemplu de mesaj *pierdut*	214
4.17	Situaţia blocajului activ	214
4.18	Relansarea concurentă în schemele RFCS	218
4.19	Tehnica de relansare concurentă în schemele RFCS	219

4.20	Schema optimistă cu sau fără modul de rezervă	220
4.21	Scheme pesimiste	221
4.22	Diferite scheme ce utlizează *rularea înainte*	222
4.23	Comparația în perfomanță dintre schemele optimiste şi pesimiste	223
4.24	Comparații ale siguranței în funcționare între schemele optimiste şi pesimiste	223
4.25	Întârzierea permanentă la ieşirile schemelor cu *rulare înapoi*	224
4.26	Întârzierea temporară la ieşirile schemelor cu *rulare înainte*	225
4.27	RIN *crossbar* tolerantă la defect	229
4.28	Reconfigurarea unui *hipercub*	232
4.29	Reconfigurarea unui *hipercub* utilizând noduri de extensie	234
4.30	Direcționarea în *hipercuburi*	235
4.31	Reconfigurarea unei RIN *plasă*	237
4.32	Reconfigurarea unei RIN *plasă* prin mapare logică	238
4.33	Reconfigurarea în RIN *arbore*	240
4.34	O diagramă generică pentru RIN *multinivel*	241
4.35	RIN *omega* 16 X 16	247
4.36	RIN-MN cu identificarea subseturilor conjugate de comutatoare	248
4.37	Exemplu de izolare a unui comutator defect	248
4.38	Scheme de realizare a unor conexiuni multiple	251
4.39	Un exemplu de rețea ASEN-Max	252
4.40	O rețea ASEN-2 pentru *N=16*	252
4.41	Grafurile de redundanță pentru rețelele ASEN 2 şi ASEN 4	257
4.42	Probabilitatea de terminal efectivă pentru ASEN	258
4.43	Parametrul *MTTF* pentru diferite RIN-MN	258
4.44	Probabilitatea de acceptanță pentru diferite RIN-MN	259
5.1	O diagramă bloc a fiabilității unei RIN	267
5.2	Un SMP-MP cu RIN *butterfly* de dimensiuni 16 X 16	270
5.3	Un lanț Markov pentru un subsistem procesor	273
5.4	Un lanț Markov pentru o RIN-MN	275
5.5	Fiabilitatea unui SMP-MP de talie 16 X 16	277

5.6	Dependabilitatea raportată la performanță	277
5.7	Distribuția mediată în timp a BW	277
5.8	Decompoziția unui sistem	280
5.9	Lanț Markov pentru subsistemul SPM	281
5.10	Lanț Markov pentru RIN-MN	286
6.1	Arhitectura SMP experimental	299
6.2	Arhitectura SMP cu ierarhie de memorie	300
6.3	Un SMP de talie 16 X 16 bazat pe RMM	302
6.4	O rețea RMM 16 X 16 ce utilizează comutatoare 4 X 4	302
6.5	O rețea RMM 16 X 16 ce utilizează comutatoare 2 X 2	304
6.6	Diagrama bloc a unui comutator 2 X 2 din rețeaua RMM	305
6.7	Diagrama bloc a unui controller de trafic	306
6.8	Înlănțuirea pachetelor într-un comutator RMM	308
6.9	Rutarea RD în RMM	310
6.10	Rutarea RI în RMM	311
6.11	Rutarea BD în RMM	312
6.12	Rutarea BI în RMM	313
6.13	Cozi de aşteptare în comutatoare RMM şi RMB	329
6.14	Comparație a rezultatelor analizei şi simulării pentru P_u la RMM	334
6.15	Comparație a rezultatelor analizei şi simulării pentru T_r la RMM	334
6.16	Comparație între utilizările procesorului, variind m	335
6.17	Comparație între timpii de răspuns, variind m	335
6.18	Utilizarea comutatorului în RMM şi în RMC variind dimensiunea lui	336
6.19	Eficacitatea costului în RMM şi în RMC variind dimensiunea	336
6.20	Utilizarea procesorului: scalabilitatea RMM în raport cu RMB	338
6.21	Timp de răspuns: scalabilitatea RMM în raport cu RMB	338
6.22	Graf de redundanță în RMM	342
6.23	Fiabilitatea diferitelor RIN pentru comutatoare 2 X 2	343
6.24	Fiabilitatea diferitelor RIN pentru comutatoare 4 X 4	343
6.25	Un arbore ce conectează modulul M_4 cu toate procesoarele	345
6.26	Diagrame ale tranzițiilor de stare	346

6.27	Configurarea adresei fizice	350
6.28	Organizarea directorului într-un comutator RMM	351
6.29	Simulator pilotat prin execuție	352
6.30	Configurația unui nod și interfața cu rețeaua	353
6.31	Rezultate ale simulării bazate pe execuție	357
6.32	Organizare de tip IGR a unui SMP-MPD	359

LISTA PRINCIPALELOR ABREVIERI

ASEN	Rețea de interconectare augmentată bazată pe amestec perfect
BD	Buclare directă
BI	Buclare indirectă
CCH	Coerența memoriei cache
LAN	Rețea cu arie locală
M	Memorie
MC	Memorie comună
MCH	Memorie cache
MIMD	SMP din taxonul „instrucții multiple, date multiple”
MIMD-L	SMP din taxonul MIMD, slab cuplat
MIMD-T	SMP din taxonul MIMD, strâns cuplat
MK	Lanț Markov
ML	Memorie locală
MM	Magistrală multiplă
MP	Memorie partajată
MPS	Multiprocesor simetric
NUMA	Acess non-uniform la memorie
P	Procesor
P	Proces
PE	Element de procesare
PM	Cuplu procesor-memorie
PV	Punct de verificare
RD	Rutare directă
RI	Rutare indirectă
RIN	Rețea de interconectare
RIN-CB	Rețea de interconectare *crossbar*
RIN-MM	Rețea de interconectare cu magistrale multiple
RIN-MN	Rețea de interconectare multinivel
RIN-MN(B)	Rețea de interconectare multinivel bidirecțională

RMB	Rețea de interconectare multinivel bidirecțională
RMM	Rețea de interconectare bazată pe magistrală multinivel
SAC-MCH	Schemă de asigurare a coerenței memoriilor cache
SE	Element de comutare
SMP	Sistem multiprocesor
SMP-MP	Sistem multiprocesor cu memorie partajată
SMP-MPD	Sistem multiprocesor cu memorie partajată distribuită
SMP-TM	Sistem multiprocesor cu transfer de mesaje
SW	Comutator
UMA	Acces uniform la memorie

1. INTRODUCERE

Calculul paralel pe scară largă devine un instrument indispensabil în diferite domenii ale ştiinţei şi tehnologiei cu un grad ridicat de diversitate. Proiectarea şi evaluarea sistemelor multiprocesor reprezintă abordări complexe şi necesită explorarea unui spaţiu de proiectare foarte larg.

1.1 TENDINŢE ARHITECTURALE

Noile tendinţe în arhitecturile SMP sunt în strânsă legătură cu dezvoltarea tehnologică în sensul convertirii acesteia în performanţă şi capabilitate. Fundamental, cele două căi prin care un volum mare de resurse (de exemplu mai mulţi tranzistori) ameliorează performanţa, sunt *paralelismul* şi *localitatea*. Cele două abordări concurează în ultimă instanţă pentru aceleaşi resurse. În general, cea mai bună performanţă este obţinută printr-o strategie combinată ce alocă resurse pentru a exploata un anumit grad de paralelism şi un anumit grad de localitate. Paralelismul şi localitatea interacţionează pe diferite niveluri, de la circuit integrat până la structură.

În fig. 1.1 se prezintă evoluţia în timp a sistemelor multiprocesor, evidenţiindu-se creşterea numărului de procesoare.

În fig. 1.2 se evidenţiază lăţimea de bandă în diferite multiprocesoare comerciale. Se remarcă că după o creştere lentă în decurs de câţiva ani, a început o nouă eră în 1991, cea a sistemelor bazate pe magistrală ce suportă creşterea substanţială a numărului de procesoare foarte rapide.

1.2 CONVERGENŢA ARHITECTURILOR PARALELE

Din punct de vedere istoric, maşinile paralele s-au dezvoltat în jurul câtorva concepte arhitectonice şi cele mai multe discuţii se poartă şi astăzi în jurul taxonomiei acestor proiecte. Proiectanţii sunt influenţaţi de aceleaşi forţe tehnologice şi de necesităţi similare generate de aplicaţii. Din această cauză a apărut un pronunţat fenomen de convergenţă în domeniu. El se manifestă mai ales în arhitecturile aparţinând ultimului deceniu.

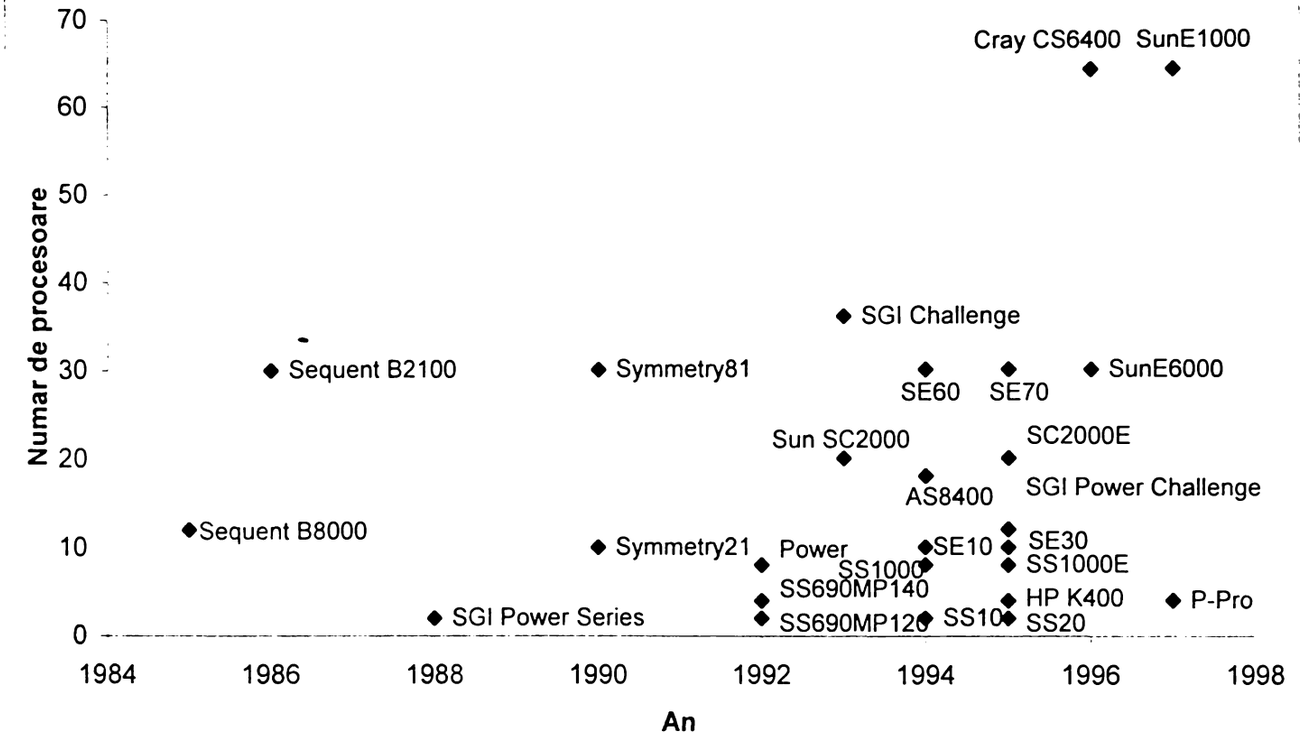


Fig. 1.1 Numărul de procesoare în SMP-MP comerciale

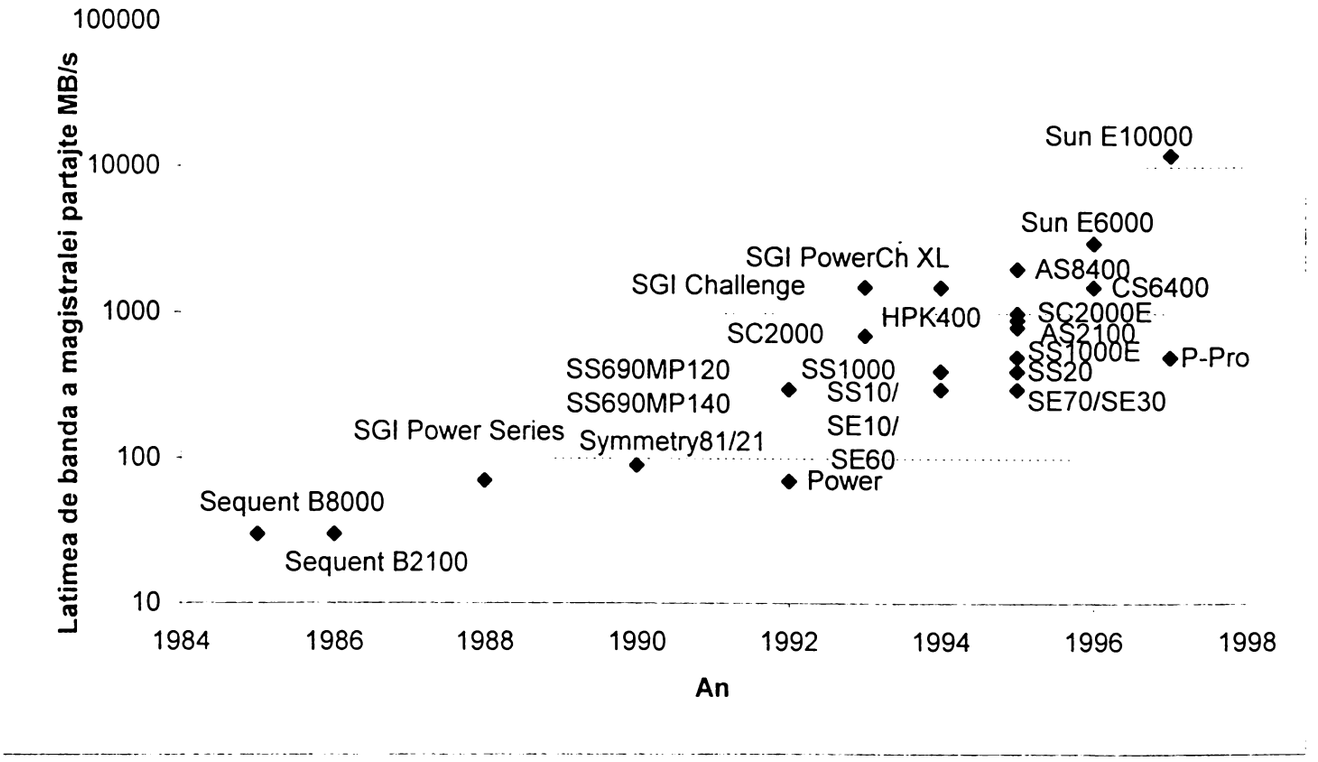


Fig. 1.2 Lățimea de bandă a SMP-MP disponibile pe piață

1.2.1 Arhitectura de comunicație

Plecând de la plastica definiție a lui Almasi şi Gottlieb în care un calculator paralel este văzut ca şi "*o colecție de elemente de procesare ce comunică şi cooperează pentru a rezolva rapid mari probleme*" [AG89], este rațional să se privească o arhitectură paralelă ca fiind extensia unei arhitecturi convenționale. În arhitectura paralelă se utilizează *comunicarea* şi *cooperarea* între procesoare. În esență, arhitectura paralelă extinde conceptele uzuale ale arhitecturii calculatorului cu cele ale unei arhitecturi de sisteme de comunicație. Arhitectura calculatorului are două fațete distincte. Una se referă la frontiera hard/soft şi la frontiera utilizator/sistem. Cea de-a doua se referă la structura organizatorică ce realizează aceste cerințe pentru a obține o performanță ridicată la un cost convenabil.

O *arhitectură de comunicație* defineşte modelul de comunicare de bază şi sincronizarea structurând modulele ce realizează aceste operații. Modelele de programare în arhitecturile paralele sunt: *adresa partajată*, *transferul de mesaje*, *prelucrarea în paralel a datelor*, *multiprogramarea*.

Din perspectiva istorică a unui model de programare se poate gândi arhitectura ca fiind îmbinarea armonioasă a *modelului de programare*, a *modelului de comunicare* şi a *organizării sistemului*. Rezultă, în această manieră, arhitecturi cu memorie partajată, cu transfer de mesaje, etc.

1.2.2 Spațiul de adresă partajat

Una dintre cele mai importante clase de sisteme paralele o reprezintă *multiprocesoarele cu memorie partajată* (SMP-MP). O proprietate fundamentală a acestei clase este apariția comunicării în mod implicit ca şi un rezultat al instrucțiilor de acces la memoria convențională. Un model primar de programare pentru aceste sisteme este în esență unul de partajare a timpului pe un singur procesor, cu excepția faptului că paralelismul real înlocuieşte interpătrunderea în timp. În mod formal, un *proces* este un spațiu de adresă virtuală şi unul sau mai multe fluxuri de comandă. Procesele pot fi configurate de o astfel de manieră încât porțiuni ale spațiului de adresă virtuală sunt partajate, ceea ce înseamnă că sunt mapate la o locație fizică comună, aşa cum se sugerează în fig. 1.3.

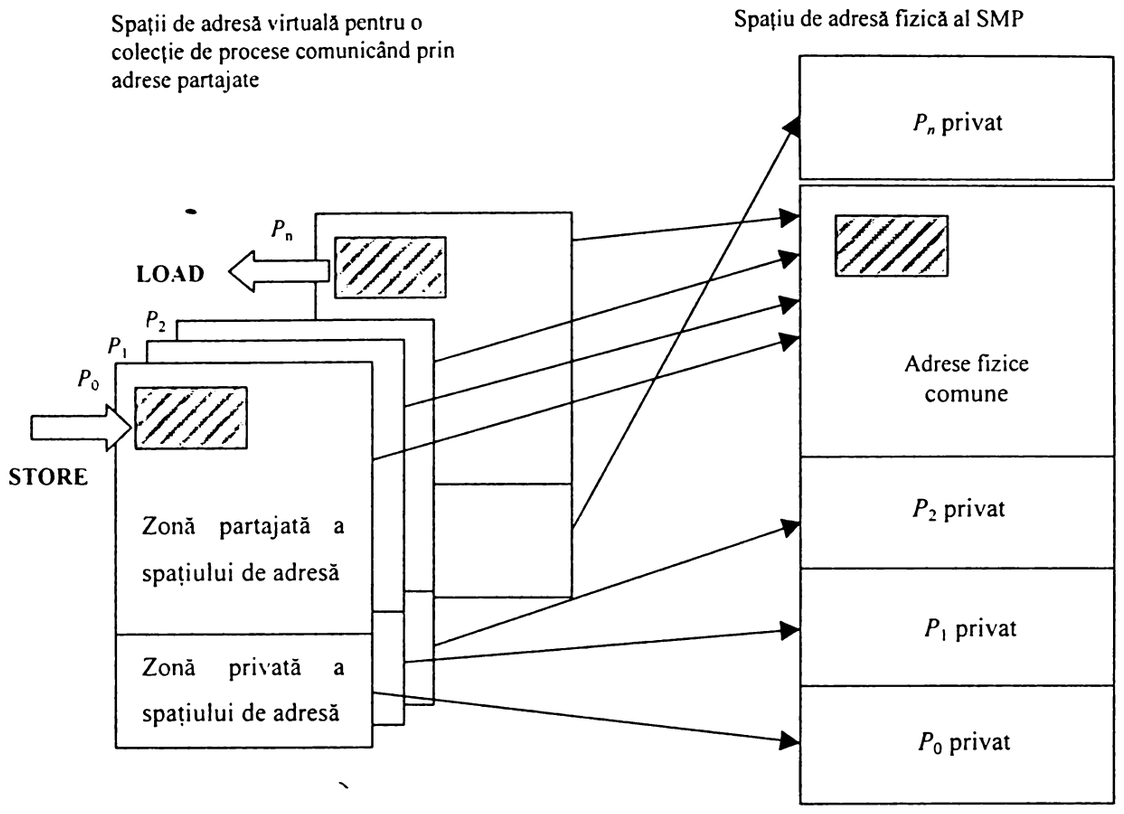


Fig. 1.3 Model tipic de memorie pentru programe paralele cu memorie partajată

Cooperarea şi coordonarea pentru fluxuri sunt realizate prin citire şi scriere de variabile partajate ca şi de pointeri ce se referă la adresele partajate. *Scrierile la o adresă logic partajată de către un flux sunt vizibile pentru citirile altor fluxuri.*

Arhitectura de comunicare utilizează operaţiuni convenţionale referitoare la memorie pentru a asigura comunicarea printre adresele partajate în aceeaşi măsură ca şi operaţiuni speciale la nivel atomic pentru sincronizare.

Hardware-ul de comunicaţie pentru SMP-MP reprezintă o extensie naturală a sistemului de memorie ce se regăseşte în cele mai multe dintre calculatoare. În esenţă, toate sistemele de calcul paralele permit unui procesor şi unui set de controlere I/E să acceseze o colecţie de module de memorie printr-un sistem de interconectare. Capacitatea memoriei este mărită prin adăugarea de module suplimentare. Această capacitate suplimentară poate sau nu să mărească lăţimea de bandă a memoriei disponibile, depinzând de organizarea specifică a sistemului. Capacitatea I/E se măreşte prin

adăugarea de dispozitive I/E la controlerele I/E sau prin inserarea de noi controlere. Există două căi posibile de creştere a capacităţii de procesare: să se aştepte apariţia unor procesoare mai rapide sau să se adauge mai multe procesoare. Configuraţia cadru a unui astfel de SMP-MP este prezentată în fig. 1.4.

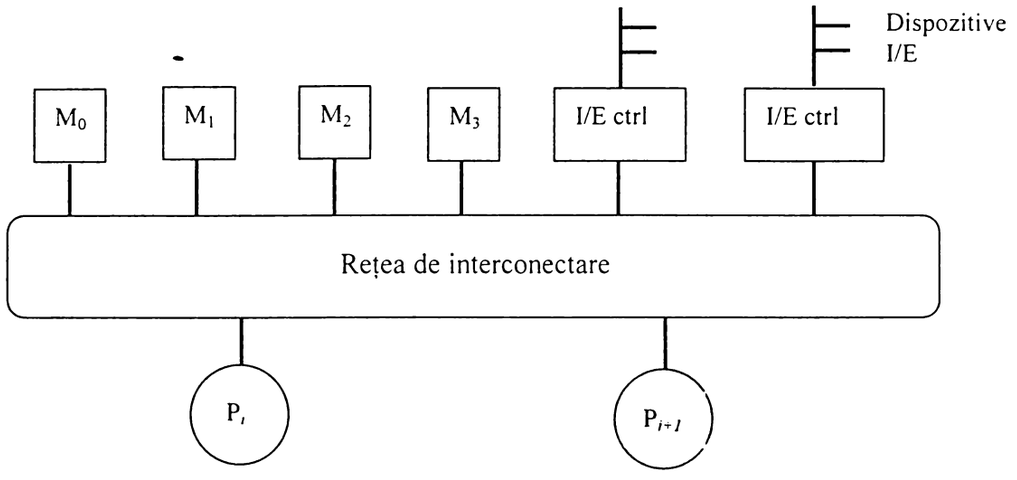
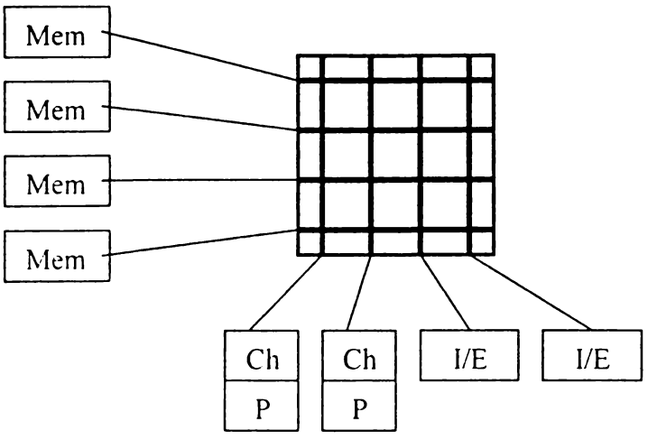


Fig. 1.4 Extensia unui sistem într-un SMP-MP prin adăugarea de module procesor

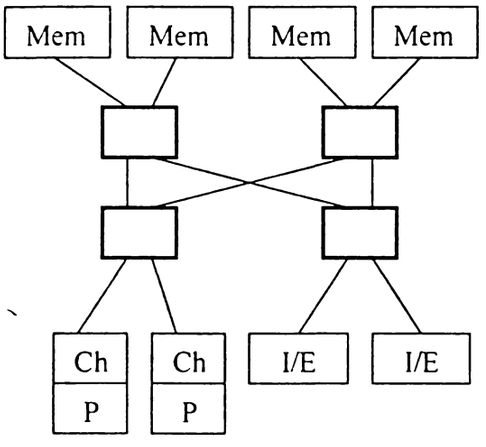
Din această configuraţie au derivat şi altele, în funcţie de avansul tehnologic.

În fig. 1.5 a), b), c) se prezintă cele trei configuraţii de bază ale SMP-MP.

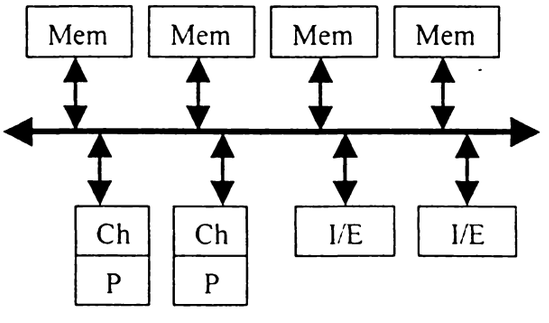
Iniţial sistemele au fost organizate ca şi în fig. 1.5 a) unde o conexiune *crossbar* conectează CPU şi câteva canale I/E la câteva module de memorie. Adăugarea de noi procesoare a reprezentat în primă instanţă o problemă de expandare a dimensiunii comutatorului; structura hardware de acces la memorie a rămas nemodificată. Dimensiunea şi costul procesorului au limitat aceste structuri timpurii la un număr mic de procesoare, dar pe măsură ce densitatea de încapsulare hardware s-a ameliorat şi costurile au scăzut, au putut fi proiectate sisteme mai mari. Costul scalării unei reţele *crossbar* a devenit un factor limitativ şi în cele mai multe dintre cazuri reţeaua a fost înlocuită cu *reţeaua de interconectare multinivel* (RIN-MN) ca şi în fig. 1.5 b). În acest caz costul a evoluat mult mai încet în funcţie de numărul de porturi. Aceste economii s-au făcut însă pe cheltuiala unei latenţe mai mari şi a unei lăţimi de bandă mai mici. Abilitatea de a accesa toată memoria direct de la fiecare procesor are câteva avantaje: fiecare procesor poate rula orice proces sau poate manipula orice eveniment I/E iar structurile de date pot fi partajate în cadrul sistemului de operare (SO).



a) Comutator *crossbar*



b) Rețea de interconectare multinivel



c) Interconectare de tip magistrală

Fig. 1.5 Scheme de interconectare tipice în SMP-MP

Utilizarea masivă a SMP-MP a început odată cu revoluția microprocesorului pe 32 bit la mijlocul anilor '80, atunci când procesorul, cache-ul, unitatea de virgulă flotantă şi unitatea de management a memoriei au putut fi încorporate pe o singură placă. Multe sisteme de rang mediu, incluzând minicalculatoare, servere, stații de lucru şi PC-uri sunt organizate în jurul unor magistrale ca şi în fig. 1.5 c). Mecanismul de acces standard al magistralei permite oricărui procesor să acceseze orice adresă fizică din sistem. Ca şi în sistemele bazate pe comutatoare, toate locațiile de memorie sunt echidistante față de toate procesoarele. Astfel toate procesoarele experimentează acelaşi timp de acces sau aceeaşi latență la o referire de memorie. Această configurație este adeseori denumită *multiprocesor simetric* (MPS).

Factorii ce limitează numărul de procesoare ce pot fi suportate într-o organizare bazată pe magistrală sunt diferiți de aceia dintr-o structură bazată pe comutator. Adăugarea de procesoare la comutator este o operație costisitoare; oricum, lățimea de bandă creşte cu numărul de porturi. Costul adăugării unui procesor la magistrală este redus, dar lățimea de bandă este fixată. Divizând această lățime de bandă fixată pentru mai multe procesoare se limitează scalabilitatea abordării. Problema este parțial rezolvată prin introducerea memoriilor cache ce reduc solicitările pentru lățime de bandă dar introduc alte limitări. Datorită datelor replicate în cache-urile locale, apare însă problema consistenței şi coerenței, ce vor fi tratate în continuare.

Scalabilitatea alături de latență şi de lățime de bandă, reprezintă un alt parametru important în evaluarea performanței unui SMP. SMP-MP orientate pe magistrală nu sunt scalabile deoarece au lățime de bandă fixă. *Crossbar*-ul nu scalează convenabil deoarece costul evoluează cu patratul numărului de porturi. Există multe alternative de RIN scalabile în care lățimea de bandă creşte pe măsură ce se adaugă procesoare iar costul nu creşte excesiv.

O abordare naturală în construcția unor SMP-MP scalabile este menținerea *accesului uniform la memorie* (UMA) ca şi în arhitectura tip "sală de dans" prezentată în fig. 1.4 şi furnizarea unei interconectări scalabile între procesoare şi memorii. Fiecare acces la memorie este convertit într-o tranzacție-mesaj prin RIN, mult mai profund decât s-ar fi putut face în cazul MPS. Dezavantajul major al acestor abordări este acela că latența RIN este experimentată la fiecare acces la memorie şi trebuie să fie furnizată o mare lățime de bandă fiecărui procesor.

O abordare alternativă o reprezintă interconectarea completă a procesoarelor, fiecare cu memoria sa locală, ca şi în fig. 1.6. În această abordare cu *acces neuniform la memorie* (NUMA), controlerul memoriei locale determină dacă este cazul să se facă un acces la memoria locală sau o tranzacţie-mesaj cu un controler al unei memorii situate la distanţă. Accesarea memoriei locale este mai rapidă decât accesarea memoriei la distanţă. Sistemul I/E poate fi fie parte din fiecare nod, sau poate fi consolidat în noduri I/E speciale. Accesele la datele private, ca şi în cazul codurilor sau al stivei, pot fi frecvent operate local. La fel se prezintă situaţia în cazul datelor partajate ce sunt înmagazinate în nodul local.

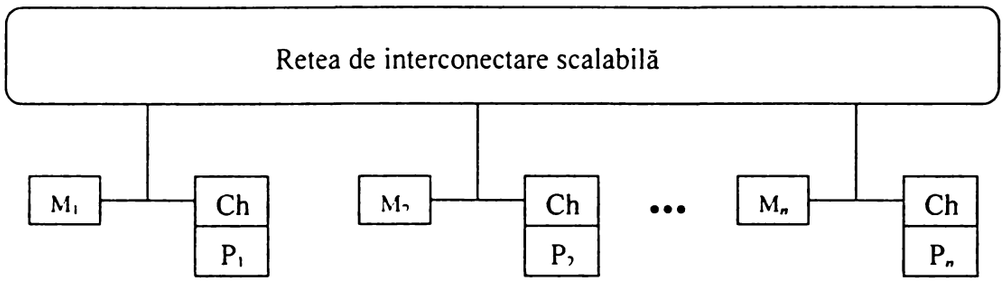


Fig. 1.6 Organizarea unui SMP-MP scalabil cu acces neuniform la memorie (NUMA)

Abilitatea de a accesa rapid memoria locală nu creşte semnificativ timpul de acces la datele situate la distanţă. Se reduce însă timpul mediu de acces, mai ales în situaţia în care o mare parte din accese sunt pentru datele locale. De asemenea se reduce cererea de lăţime de bandă, plasată pe RIN. Deşi a avea toate datele partajate situate echidistant faţă de orice procesor reprezintă o anumită simplificare în proiectare, abordarea NUMA a devenit prevalentă în majoritatea sistemelor moderne. Citām sistemele CRAY T3E şi SGI Origin.

În concluzie, comunicarea şi cooperarea în modelul de programare cu spaţiu de adresă partajată constau în citiri şi scrieri de variabile partajate; aceste operaţii sunt mapate direct la un model de comunicare constând în instrucţii LOAD/STORE ce accesează un spaţiu global de adrese partajate. Ficare procesor poate *nominaliza* fiecare adresă fizică din maşină. Un proces poate nominaliza toate datele pe care le partajează cu altele în cadrul spaţiului de adrese virtuale.

Datele sunt transferate fie ca şi primitive în setul de instrucţii (byte, cuvinte, etc.) fie în blocuri cache. Fiecare proces efectuează operaţii referitoare la memorie pe adrese

aparținând spațiului de adrese virtuale. Procesul de translatare a adresei identifică o adresă fizică, ce poate fi locală sau la distanță față de procesor şi poate fi partajată cu alte procese. Translatarea de adresă realizează protecția în cadrul spațiului de adresă partajată, ca şi în cazul uniprocesoarelor, deoarece un proces poate doar accesa datele în spațiul de adresă virtuală.

Eficacitatea SMP-MP depinde de latența înglobată în accesele la memorie ca şi de lățimea de bandă a transferului de date ce poate fi suportat de sistem.

1.2.3 Transferul de mesaj

O a doua clasă importantă de maşini paralele, denumită *arhitecturi cu transfer de mesaje*, utilizează calculatoare complet echipate ca şi blocuri constitutive şi furnizează comunicare între procesoare în termeni de operații explicite de I/E. Diagrama bloc de nivel înalt pentru un *sistem multiprocesor cu transfer de mesaje* (SMP-TM) este în esență similară cu abordarea NUMA în cazul SMP-MP, prezentată în fig. 1.6. Prima diferență constă în modul de integrare al comunicării la nivel de I/E şi nu de memorie. Acest stil de proiectare are multe în comun cu rețelele de stații de lucru, sau *grupuri* (*clusters*), exceptând faptul că gradul de compactare a nodurilor este mult mai masiv, neexistând monitor sau tastatură la nivelul nodului, iar RIN are o capabilitate mult mai mare decât LAN-urile standard. Integrarea dintre procesor şi rețea este mult mai puternică decât în structurile tradiționale I/E, deoarece transferul de mesaje este în mod fundamental o comunicare procesor la procesor.

În transferul de mesaje, există o distanță substanțială între modelul de programare şi primitivele hardware, existând o comunicare la nivel de utilizator efectuată prin SO sau apeluri din bibliotecă, ceea ce provoacă multe acțiuni la nivel inferior.

Cele mai comune acțiuni de comunicare la nivelul utilizatorului sunt variante de SEND/RECEIVE. În forma sa cea mai simplă, SEND specifică un tampon de date local al cărui conținut trebuie transmis şi un proces receptor, de obicei rezident într-un procesor situat la distanță. RECEIVE specifică un proces transmițător şi un tampon local în care data transmisă va fi depozitată. Interacțiunea dintre un SEND şi un RECEIVE cauzează un transfer de date de la un proces la altul, situație prezentată în fig. 1.7.

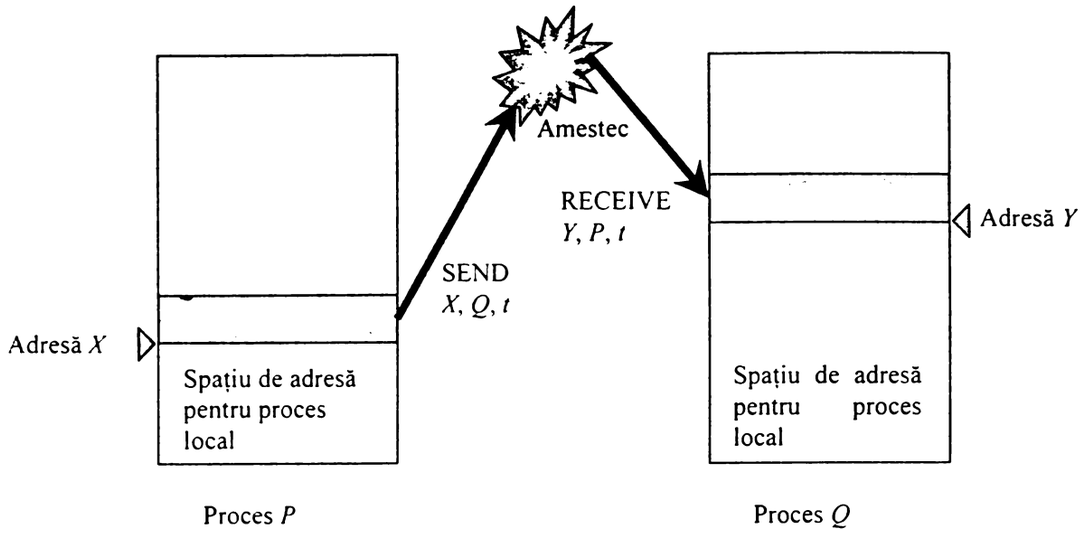


Fig. 1.7 Un model de abstractizare a transferului de mesaj de tip SEND/RECEIVE la nivel de utilizator

În cele mai multe dintre SMP-TM, operația SEND permite de asemenea ataşarea la mesaj a unui indicator sau *etichetă de rutare* iar la recepție se specifică regula de dezmembrare a mesajului. Astfel, programul utilizatorului nominalizează adresele locale şi intrările într-un spațiu abstract proces-etichetă. Există câteva posibile variante de sincronizare, depinzând de faptul că SEND se mai execută în timp ce RECEIVE a fost executat, când tamponul de transmisie este disponibil pentru reutilizare sau când cererea a fost acceptată.

SMP-TM din primele generații au furnizat primitive hardware ce erau foarte apropiate de nivelul de comunicare SEND/RECEIVE la nivel de utilizator, având însă câteva restricții adiționale. Multe SMP-TM în fazele incipiente au fost *hipercuburi* unde fiecare nod este conectat la n alte noduri dintr-un total de 2^n noduri. Altele au fost *plase,* unde nodurile sunt conectate la vecini pe două sau trei dimensiuni. Topologia RIN a fost importantă în SMP-TM incipiente deoarece doar procesoarele învecinate ar fi putut fi nominalizate în cadrul operațiilor SEND/RECEIVE.

Topologia fizică a RIN a impus la început denumirea algoritmilor paraleli. Ei au fost denumiți: *inel*, *hipercub*, etc. Accentul pe topologia RIN a fost semnificativ redus odată cu introducerea RIN cu un grad ridicat de generalitate, care asigurau transferul în loturi a mesajului prin fiecare dintre mecanismele de rutare ce formau RIN. Câteva exemple sunt: IBM-SP2, RS6000, Intel Paragon, etc.

1.2.4 Convergența

Evoluția SMP în termeni de hardware şi software a estompat frontiera, odată clar delimitată, dintre SMP-MP şi SMP-TM. Se constată o anumită convergență între cele două arhitecturi ce apar nu numai în concepte ci şi în realizările structurale [CSG99].

În primul rând, la nivelul *operațiunilor de comunicare* disponibile la procesele utilizatorilor există:

- Transferuri de mesaje tradiționale, SEND/RECEIVE, ce sunt suportate de cele mai multe din SMP-MP prin tampoane de stocare partajate. SEND presupune scrierea de date sau o indicare a lor în tampon; RECEIVE implică citirea datelor din tampoanele partajate. Fanioanele şi blocajele sunt utilizate pentru a comanda accesul spre tampon şi pentru a indica evenimente ca şi sosirea mesajelor.
- La un SMP-TM, un proces utilizator poate construi un spațiu global de adresă prin transportul pointerilor specificând procesul şi o adresă virtuală locală în acest proces. Accesul la o astfel de adresă globală poate fi obținut software printr-o tranzacție de mesaj explicită. O citire logică se realizează prin transmiterea unei cereri spre procesul conținând obiectul şi recepționând un răspuns.
- Un spațiu de adresă virtuală poate fi stabilit la un SMP-TM la nivel de pagină. O colecție de procese are o regiune de adrese partajate dar, pentru fiecare proces, doar paginile ce sunt locale sunt accesibile. După un acces la o pagină ratată, de obicei situată la distanță, apare un defect de pagină şi SO angajează nodul de la distanță într-o tranzacție de mesaj cu scopul de a transfera pagina şi de a o proiecta în spațiul de adresă al utilizatorului.

La nivelul *organizării sistemului*, o convergență substanțială a apărut în ultimii ani. Arhitecturile SMP-TM apar, la nivel de schemă bloc, identice cu arhitecturile scalabile NUMA. În cazul memoriilor partajate, RIN a fost integrată cu controlerul cache sau cu cel al memoriei în scopul de a observa eşecurile de acces la cache şi pentru a conduce la generarea unui mesaj pentru accesul la memoria dispusă într-un nod situat la distanță. În SMP-TM, interfața cu RIN este de obicei asigurată de un dispozitiv I/E. Tendința a fost de a integra acest dispozitiv mult mai profund în sistemul de memorie şi de a transfera datele direct de la şi spre spațiul de adrese al utilizatorului. Unele proiecte prevăd

transferul DMA prin RIN de la memoria unui sistem la memoria altui sistem, astfel ca interfața RIN să fie puternic integrată cu sistemul de memorie.

În plus față de convergența SMP-TM scalabile şi a SMP-MP, au apărut LAN-uri bazate pe comutator ce includ Ethernet, ATM, şi alte medii de comunicare, oferind interconectări scalabile ce se apropie de ceea ce maşinile paralele tradiționale pot oferi. Aceste noi rețele sunt utilizate pentru a conecta colecții de sisteme în *grupuri* care pot opera ca şi o maşină paralelă asupra unor probleme mari sau ca şi multe maşini individuale în multiprogramare.

În concluzie, *transferul de mesaj* şi *spațiul de adresă partajată* reprezintă două modele distincte de programare, fiecare furnizând o bine definită paradigmă pentru *partajare*, *comunicare* şi *sincronizare*. Prin urmare, este normal să se considere ambele aspecte într-un cadru comun. Integrarea mult mai puternică a comunicării în sistemul de memorie tinde să reducă latența tranzacțiilor prin RIN şi să amelioreze lățimea de bandă ce poate fi furnizată sau acceptată de RIN.

1.3 COERENȚA ŞI CONSISTENȚA MEMORIILOR CACHE

Reducerea *latenței* comunicației reprezintă una dintre finalitățile pe care un proiectant de arhitecturi de sisteme de calcul paralele trebuie să o aibă în vedere. Această reducere se efectuează atât la nivel de ierarhie de memorie cât şi la cel al rețelei de interconectare. În ceea ce priveşte ierarhia de memorie, un rol major îl au memoriile cache private situate pe nivelul superior, adică cel mai apropiat de procesor, şi care reduc efectiv întârzierea medie a accesului la memorie. În sistemele multiprocesor, datorită dispersiei datelor printre procesoare şi datorită problemei coerenței memoriilor cache, efectul benefic al introducerii memoriilor cache nu este atât de evident ca şi în cazul sistemelor uniprocesor. În literatură [Lil93], [CF78], [CFKA90], [Choi96] sunt descrise o varietate largă de mecanisme de menținere a coerenței cache în SMP-MP de capacitate mare. Pentru a se putea face o comparare şi în acelaşi timp o evaluare a performanței acestor mecanisme, arhitectul de sistem de calcul trebuie să ia în considerare următoarele:

- *strategia de detectare a coerenței* prin care posibilele accese incoerente la memorie sunt detectate fie statistic în timpul compilării, fie dinamic în timpul rulării programului

- *strategia de întărire a coerenței*, manifestată prin protocoale de actualizare sau de invalidare şi care este utilizată pentru a se asigura că intrările în memoriile cache neactualizate nu vor fi niciodată referite de către un procesor
- *precizia informației referitoare la partajarea blocurilor* care dă o măsură a costului de implementare şi a performanței unui mecanism de menținere a coerenței
- *dimensiunea blocului cache* ce afectează performanța sistemului de memorie

Secvența adreselor de memorie generate de către un program afişează proprietățile localității *temporale* şi *spațiale* [Smit82]. *Localitatea temporală*, sau localitatea în timp, înseamnă că memoriile ce au fost recent adresate de către un program, vor fi din nou referite în viitorul foarte apropiat. *Localitatea spațială* înseamnă că adresele referite de către un program într-o perioadă scurtă de timp vor cuprinde o porțiune relativ mică a întregului spațiu de adresă. Memoriile cache private care sunt mici, rapide şi situate lângă un procesor, exploatează aceste proprietăți de referire la memorie pentru a reduce timpul mediu necesar accesării memoriei principale care uzual are o capacitate mult mai mare. Prin înmagazinarea temporară în cache a unei copii a unei valori provenind din memoria principală, care a fost activ referită de către program, memoriile cache amortizează timpul cerut pentru copierea conținutului unei locații de memorie de la o memorie principală mai lentă într-o memorie cache mai rapidă. Datorită faptului că multiplele copii ale unei locații de memorie partajată pot fi rezidente simultan în câteva cache-uri diferite, memoriile cache private introduc problema *coerenței* în care există posibilitatea pentru ca diferite copii cache-ate să aibă valori diferite în acelaşi moment de timp. Este responsabilitatea mecanismului de menținere a coerenței cache – SAC-MCH – de a asigura că ori de câte ori un procesor citeşte o locație de memorie el recepționează valoarea corectă.

Există două aspecte importante referitoare la problema coerenței cache. Prima se referă la *modelul sistemului de memorie* prezentat programatorului. Al doilea aspect important îl reprezintă *mecanismul* utilizat de către sistem pentru menținerea coerenței între memoriile cache şi memoria principală.

1.3.1 Modele de consistență

Una dintre definițiile unui SMP ce utilizează memorii cache coerente este aceea a unui sistem care garantează că "... *valoarea returnată a unei instrucțiuni Load este întotdeauna valoarea dată de cea mai din urmă instrucțiune Store ce a operat asupra aceleiași adrese*" [CF78]. Dificultatea cu această definiție este că semnificația lui "cel mai din urmă"-nu este precis definită atunci când instrucțiunile de încărcare și de memorare apar în diferite procesoare ce rulează asincron, unul în raport cu celălalt. Datorită întârzierilor și temporizărilor în diferite porțiuni ale RIN ce interconectează binomul procesor – memorie și în cadrul însăși a procesoarelor și memoriilor, fiecare procesor și fiecare modul de memorie poate observa o ordonare diferită a evenimentelor. *Modelul de consistență* al unui SMP definește punctul de vedere al programatorului referitor la ordonarea în timp a evenimentelor ce apar în diferite procesoare. Aceste evenimente includ citiri și scrieri în și din memorie, operații de sincronizare.

Din punctul de vedere al programatorului referitor la sistemul de memorie, modelul de *consistență secvențială* definește o ordonare strictă a secvenței de execuție a operațiunilor de memorie permise în cadrul unui procesor și între procesoare. Se spune că un SMP este consistent secvențial dacă "... *rezultatul oricărei execuții a unui program este același ca și în cazul în care operațiile tuturor procesoarelor au fost executate într-o anumită ordine secvențială și operațiile fiecărui procesor individual apar în această secvență în ordinea specificată de către programul său*" [Lamp79]. Această strictă ordonare a acceselor la memorie impune o penalizare severă în performanță prin limitarea drastică a suprapunerilor permise între operațiile de memorie livrate de către un procesor individual și de către alte procesoare.

Modelul de consistență cu *ordonare slabă* [DSB88] relaxează ordinea garantată a evenimentelor modelului de consistență secvențială pentru a permite o mai mare suprapunere a scrierilor și citirilor din memorie. Cu acest model doar accesele la memorie către variabilele de sincronizare definite de programator sunt garantate că vor apare într-o ordine secvențial consistentă. În rest, toate referirile la memorie produse de procesoare diferite pot apare în oricare ordine arbitrară.

Modelul de *consistență prin eliberare* [GGH91] relaxează și mai mult constrângerile de ordonare asupra variabilelor de sincronizare, divizând operația de sincronizare în operații separate de *achiziționare* și *eliberare*. Operația de *achiziționare* este furnizată de către un

procesor când el doreşte să obţină acces exclusiv la un anumit obiect din memoria partajată. Pentru a prveni interferenţa cu alt procesor ce poate în mod curent să aibă un acces exclusiv la obiectul partajat, procesorul trebuie să aştepte ca operaţiunea de achiziţionare să se termine înainte de iniţierea oricărei referiri la memoria partajată. Operaţia de *eliberare*, pe de altă parte, este utilizată pentru furnizarea unui acces exclusiv la obiectul partajat din memorie. Această fracţionare a operaţiei de sincronizare în două faze separate, permite acestui model de consistenţă să obţină o mai mare suprapunere a operaţiilor de memorie efectuate de toate procesoarele decât în cazul modelelor de consistenţă secvenţiale, respectiv de ordonare slabă [TH90], [GGH91], [GGHM91], [ZB92], [Marq89].

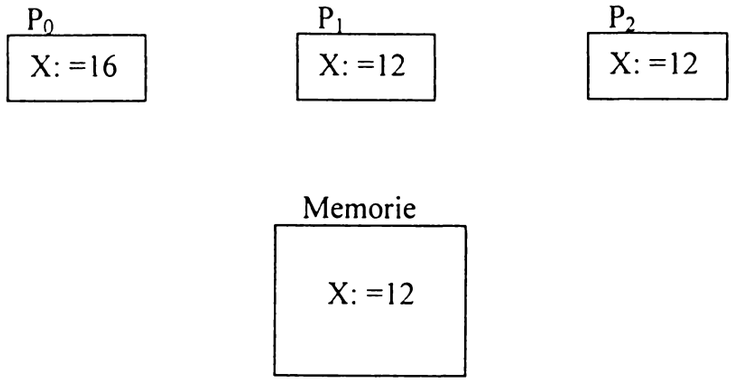
1.3.2 Coerenţa memoriilor cache

Referitor la coerenţa memoriilor cache, trebuie subliniat faptul că atunci când o locaţie de memorie este scrisă de către oricare procesor, faptul că valoarea din acea locaţie a fost schimbată trebuie să fie comunicat tuturor celorlalte procesoare care deţin o copie cache a acestei locaţii pentru a se asigura că nici unul dintre ele nu utilizează o versiune perimată. Coerenţa este menţinută de către scheme de asigurare a coerenţei memoriilor cache (SAC-MCH). Se consideră un sistem cu trei procesoare, fiecare cu o memorie cache privată în care se execută secvenţa de citiri şi de scrieri prezentată în fig. 1.8 a). După ce primele două citiri au fost efectuate la t_2, cache-urile ambelor procesoare P_o şi P_1 vor conţine valoarea "12" pentru variabila înmagazinată în locaţia X a memoriei ca şi în fig. 1.8 b). La momentul t_3 procesorul P_0 scrie în această locaţie schimbând valoarea în "16". Într-un sistem fără SAC-MCH, această valoare va fi actualizată doar în cache-ul lui P_0 astfel că atunci când P_1 reciteşte X la momentul t_4, el va citi vechea valoare de "12" din cache-ul propriu, situaţie prezentată în fig. 1.8 c).

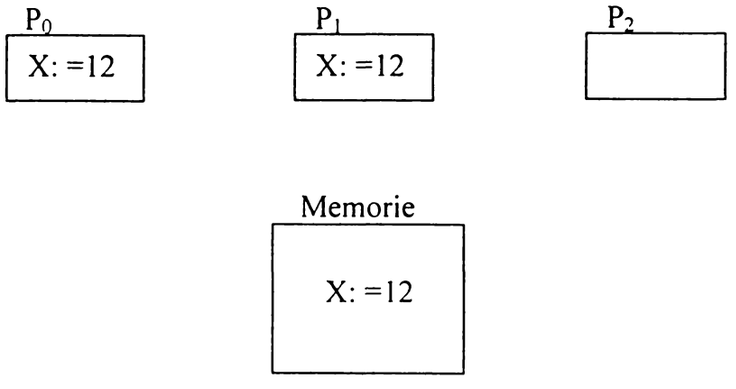
Similar, P_2 de asemenea va citi vechea valoare deoarece memoria principală nu a fost actualizată cu ultima valoare scrisă de P_0. SAC-MCH este necesar pentru a asigura ca valorile perimate ale lui X în cache-urile altor procesoare şi din memoria principală, de exemplu valoarea "12", nu vor fi propagate în viitoarele operaţii de citire.

Timp	P_0	P_1	P_2
t_1	citeşte X		
t_2		citeşte X	
t_3	scrie 16,X		
t_4	citeşte X	citeşte X	citeşte X

a) Secvenţe de citiri şi scrieri



c) Conţinuturile memoriilor cache după citiri la timpul t_4



b) Conţinuturile memoriilor cache după citire la timpul t_2

Fig. 1.8 Un exemplu de problemă de coerenţă a memoriei cache

Se crează o relaţie între modelul de consistenţă de memorie şi SAC-MCH. Această relaţie se referă în primul rând la faptul că SAC-MCH asigură că toate memoriile cache privesc toate operaţiunile de scriere la un bloc specific în aceeaşi ordine logică. Modelul de consistenţă defineşte programatorului ordinea de scrieri în diferite blocuri, ca şi când ar fi percepute de către fiecare din procesoare. Aceasta înseamnă că dacă programatorul urmează regulile modelului de consistenţă pentru sistemul ce urmează să fie utilizat, SAC-MCH forţează ca valoarea returnată de către oricare operaţiune LOAD să fie valoarea garantată de către modelul de consistenţă.

1.3.3 Relaţia între modelele de consistenţă şi coerenţa memoriilor cache

Într-un sistem ce garantează consistenţa secvenţială, SAC-MCH asigură ca efectele fiecărei operaţiuni de scriere la o locaţie din memoria partajată să fie propagate către toate cache-urile înainte de următoarea scriere la aceeaşi locaţie de către orice procesor care ar putea urma. Aceste accese sunt *puternic ordonate* [DSB88]. Pe de altă parte, într-un sistem bazat pe un model de consistenţă slab ordonat, doar accesele la variabilele de sincronizare predefinite sunt puternic ordonate. Ordonarea acceselor la locaţiile memoriei partajate care sunt efectuate de diferite procesoare, pot apare în oricare ordine. Astfel, modelul de consistenţă cu ordonare slabă asigură că valorile de date în memoriile cache sunt coerente doar în punctele de sincronizare.

1.3.4 Scheme de asigurare a coerenţei memoriilor cache

Pentru a rezolva poblema coerenţei memoriilor cache au fost propuse o varietate de scheme de menţinere a coerenţei. Soluţia optimă pentru un sistem multiprocesor dat depinde de câţiva factori ca şi mărimea sistemului exprimată în număr de procesoare, utilizarea anticipată a sistemului şi costul dorit al acestuia. În principiu, în literatură se descriu trei tipuri principale de scheme de asigurare a coerenţei memoriilor cache (SAC-MCH). SAC-MCH orientat pe *monitorizare* (*snooping*) se bazează pe un mediu de interconectare partajat între procesoare şi modulele de memorie ca de exemplu magistrala comună, care permite fiecărui procesor să monitorizeze toate tranzacţiile la memoria partajată. Un procesor monitorizează toate referirile la memoria celorlalte procesoare şi poate detecta când un bloc care este cache-at a fost schimbat de către alt procesor. Apoi

el invalidează [GVW89], [Good83] propria sa copie cache-ată de o astfel de manieră încât următoarea referire la acest bloc va forța un eşec de acces şi valoarea curentă va fi obținută din memoria principală sau din altă memorie cache. În mod alternativ, procesorul poate să actualizeze direct copia sa cache-ată cu o nouă valoare disponibilă pe magistrală. Deoarece magistrala partajată difuzează efectele fiecărei operații de scriere într-o locație de memorie partajată tuturor memoriilor cache, în acelaşi ciclu în care se produce scrierea propriu-zisă, acest SAC-MCH implementează un model de consistență puternic ordonată.

În timp ce aceste mecanisme de monitorizare sunt relativ simplu de implementat, magistrala partajată poate să devină o limitare severă a performanței. Pentru a reduce situația conflictuală de pe o singură magistrală, sistemul Wisconsin Multicube [Good88] a propus o rețea *n* dimensională de magistrale cu procesoarele localizate în punctele de intersecție ale magistralelor şi cu modulele de memorie la punctele terminale. Magistralele adiționale furnizează o lățime de bandă a memoriei mai mare în detrimentul unui protocol de menținere a coerenței mult mai complicat. O altă abordare [Arch88] grupează procesoarele în jurul unor magistrale mai mici separate şi menține coerența printre procesoarele fiecărui grup. O ierarhie adițională de magistrale este introdusă pentru a menține coerența între grupuri. O altă abordare adaugă o magistrală specială de menținere a coerenței [BLA89], [Marq89] pentru a deplasa traficul de actualizare a coerenței de la citirea datelor din magistrala memoriei primare.

Deoarece toate aceste scheme utilizează magistrale adiționale pentru a creşte lățimea de bandă între procesoare şi memoria partajată, performanța lor – în ultimă instanță – va fi limitată de către posibilitatea apariției de conflicte de acces pe magistrală când există prea multe procesoare şi de asemenea limitarea apare şi datorită dificultății tehnologice de a construi aceste magistrale de mare viteză. Prin urmare, SAC-MCH bazate pe monitorizare sunt limitate la sistemele multiprocesor de talie redusă.

O altă metodă de a preveni problema saturării magistralei constă în a înlocui magistrala cu o rețea de interconectare ca de exemplu RIN *omega*, RIN *plasă* sau un *hipercub*. Aceste rețele furnizează o lățime de bandă mai mare între procesoare şi memorii decât aceea a unei magistrale partajate dar în acelaşi timp măresc timpul de întârziere în accesul la memorie. Această întârziere mai mare intensifică dorința de a utiliza memorii private prin eliminarea mecanismului prin care procesoarele monitorizează tranzacțiile la

memoria partajată. Atât mecanismele hardware bazate pe *director* cât şi abordările *orientate pe compilator* constituie alternative pentru menținerea coerenței în aceste sisteme.

În cadrul unui SAC-MCH bazat pe *director* [Lil93] un procesor trebuie să comunice cu un director comun ori de câte ori acțiunea procesorului poate cauza o inconsistență între cache-ul său şi alte cache-uri şi memorie. Directorul menține informația referitoare la procesoarele ce au copia blocurilor deoarece diferite procesoare pot avea o copie a aceluiaşi bloc cache-at în acelaşi moment de timp. Înainte ca un procesor să poată să scrie un bloc, el trebuie să solicite un acces exclusiv la director pentru acest bloc. Înainte ca directorul să asigure acest acces exclusiv, el trimite un mesaj către toate procesoarele însoțit de o copie cache-ată a blocului, forțând fiecare procesor să-şi invalideze propria copie. După recepționarea validării transferului de către toate aceste procesoare, directorul asigură un acces exclusiv procesorului aflat în ciclul de scriere. Când un procesor încearcă să citească un bloc care este exclusiv într-un procesor diferit, el va trimite o cerere de deservire a eşecului în acces către acest director. Directorul va trimite apoi un mesaj către procesorul cu copia exclusivă, forțându-l să rescrie noua valoare în memorie. După recepționarea acestei noi valori, directorul transmite o copie a blocului către procesorul solicitant. Schemele bazate pe director diferă în cantitatea de informație ce pot să o mențină referitoare la blocurile partajate şi în ce măsură sunt utilizate operații de invalidare sau de actualizare pentru a asigura coerența.

Aceste scheme bazate pe director implementează un model de consistență puternic ordonată. Un model de consistență slab ordonată poate fi implementat cu un director prin deținerea unor informații referitoare la întârzierea procesorului, aflat într-o operațiune de scriere, atunci când el accesează o variabilă de sincronizare. Deoarece modelul de consistență slabă întârzie operațiunile de scriere ale procesorului doar în cazul variabilelor de sincronizare, el prezintă o performanță mai ridicată decât a modelului puternic ordonat.

SAC-MCH orientat pe *compilator* determină în timpul compilării care blocuri cache pot să devină perimate. Instrucțiuni speciale sunt apoi inserate în codul generat în scopul de a fi executate de fiecare dintre procesoare pentru a preveni ca să utilizeze posibile date perimate. Unul dintre cele mai simple dintre aceste mecanisme [Weid86] utilizează

invalidarea nediscriminată a datelor cache-ate în scopul întăririi coerenței prin intermediul unui model de consistență slab ordonat.

1.3.5 Factori ce afectează mecanismele de menținere a coerenței memoriilor cache

Cel mai important element în alegerea unui SAC-MCH îl constituie în mod uzual performanța sa exprimată în reducerea întârzierii medii atunci când se extrag date din memorie. Un alt factor important îl reprezintă costul implementării, exprimat în cât de multă memorie este necesară pentru a înmagazina blocul cache ce deține informația de partajare şi de către complexitatea părții de comandă. Diferitele SAC-MCH au diferite abordări în cost şi performanță, ceea ce face dificilă evaluarea lor. Parametrii ce afectează SAC-MCH pot fi rezumați astfel:

- *strategia de detectare a coerenței* ce reprezintă strategia prin care SAC-MCH detectează un posibil acces incoerent la memorie. Acest acces poate fi detectat *dinamic* în timpul rulării sau *statistic* în timpul compilării.
- *strategia de întărire a coerenței* obținută prin actualizare sau invalidare şi care este utilizată pentru a se asigura că intrările unei memorii cache ce conține informație perimată nu vor fi referite niciodată de către un procesor.
- *precizia informației de partajare a blocului* ce poate fi schimbată în raport cu costul de implementare şi de performanță a SAC-MCH.
- *talia blocului cache* ce afectează într-o oarecare măsură performanța sistemului de memorie.

1.4 REȚELE DE INTERCONECTARE

Rețeaua de interconectare (RIN) reprezintă o componentă esențială a SMP. Deoarece performanța unei aplicații ce rulează pe un astfel de sistem depinde într-o mare măsură de performanța accesului la memoria situată la distanță, *latența* RIN joacă un rol extrem de important. Prin urmare, proiectarea componentelor RIN trebuie să fie studiată şi evaluată de o astfel de manieră încât să furnizeze o *latență minimă* şi o *mare lățime de bandă*.

1.4.1 Caracteristici

O rețea de interconectare este caracterizată de: *topologie*, *algoritm de rutare*, *strategie de comutare* şi *mecanism de comandă a fluxului de date* [CSG99].

- *Topologia* reprezintă structura de interconectare a grafului rețelei; ea poate fi *regulată* sau *neregulată*. Cele mai multe SMP utilizează structuri regulate. Adeseori se face o distincție între rețelele directe şi cele indirecte; RIN *directe* au câte un nod conectat la fiecare comutator în timp ce RIN *indirecte* au nodurile conectate doar la un subset specific de comutatoare care formează muchiile rețelei. Cele mai multe dintre SMP utilizează strategii mixte, aşa că distincția cea mai critică este între cele două tipuri de noduri: nodurile procesoare ce generează şi deplasează traficul, în timp ce comutatoarele permit doar traversarea pentru acest trafic. Exemple de rețele directe sunt multe dar cele mai cunoscute sunt *hipercubul* utilizat în SGI Origin 2000, rețeaua *plasă* 2D utilizată la sistemul Intel Paragon şi rețeaua *tor* 3D utilizată la CRAY T3E. RIN indirecte sunt construite utilizând nivele de comutatoare de comunicație între nodurile sursă şi cele destinație. Exemple de rețele indirecte includ numeroase variante constructive ale rețelelor de interconectare ca şi Omega, Delta, Baseline, Benes, etc.
- *Algoritmul de rutare* determină care mesaje pot parcurge graful rețelei. Algoritmul de rutare restricționează setul căilor posibile la un set mai redus de căi permise.
- *Strategia de comutare* determină modul în care datele dintr-un mesaj traversează calea de parcurs. Există două strategii de comutare de bază: în *comutarea de circuit* se stabileşte o cale de la sursă la destinație şi ea este menținută până când mesajul este transferat prin circuit. În *comutarea de pachet* mesajul ce trebuie transmis este fragmentat în pachete mai mici ce se înlănțuie secvențial. Pachetele sunt rutate individual de la sursă la destinație, fiind rutate independent datorită informațiilor de rutare care sunt conținute. Comutarea de pachet permite o mai bună utilizare a resurselor RIN deoarece conexiunile şi tamponul sunt ocupate doar pe timpul traversării de către pachet. Această strategie *store and forward* se aplică la nivele intermediare de elemente de comutare.
- În tehnica comutării prin *canale virtuale*, se înmagazinează un pachet la un nivel intermediar de comutatoare doar dacă calea corespunzătoare de ieşire este ocupată. Se utilizează tehnica de rutare *wormhole* [NK93], [CSG99], [Dal92] în care mesajele sunt divizate într-un bloc ce conține informații de rutare şi câteva

blocuri de date. Aceste blocuri sunt transmise prin RIN într-o manieră *pipeline*, în care blocurile de date urmează calea creată de blocul conducător.

- *Mecanismul de comandă al fluxului* determină când un mesaj, sau fragmente din el, traversează ruta stabilită. În particular, comanda fluxului este necesară ori de câte ori două sau mai multe mesaje tind să utilizeze aceeaşi resursă a RIN, de exemplu un canal, în acelaşi timp. Unul din fluxurile din trafic poate fi stopat, întârziat în tampoane, deturnat pe o cale aternativă sau pur şi simplu îndepărtat. Fiecare dintre aceste opțiuni reclamă cerințe specifice în proiectarea comutatorului şi influențează aspecte ale subsistemului de comunicare. Unitatea minimă de informație ce poate fi transferată de-a lungul unei conexiuni este denumită *flit* (flow control unit).

1.4.2 Performanță de comunicare de bază în RIN

Este foarte important să se înțeleagă relația care există între performanță şi funcționalitate la nivelul subsistemului global de comunicare. În cazul RIN această discuție trebuie purtată din perspectiva latenței şi a lățimii de bandă.

Latența

Timpul pentru a transfera *n* bytes de informație de la o sursă la o destinație, are patru componente după cum urmează:

$$Timp(n)_{S-D} = \text{Timp inițializare transfer} + \text{Întârziere de rutare} + \text{Grad de ocupare a canalului} + \text{Întârziere de conflict} \quad (1.1)$$

Timpul de inițializare a transferului reprezintă timpul consumat de procesor pentru inițializarea unui transfer prin RIN. El poate fi un cost fixat dacă procesorul pur şi simplu dă comanda de inițializare a comunicării sau poate fi liniar cu *n* dacă procesorul trebuie să copieze datele înainte de a inițializa transferul.

Ocuparea canalului reprezintă timpul consumat pentru date în parcurgerea prin cea mai lentă componentă a căii de comunicare. De exemplu fiecare conexiune ce este traversată în RIN va fi ocupată pentru un timp n/BW unde BW este lățimea de bandă a conexiunii.

Întârzierea în rutare reprezintă timpul consumat în RIN în mod distinct pentru identificarea următorului comutator ce trebuie traversat.

Întârzierea generată de conflict reprezintă timpul consumat în rezolvarea situației conflictuale generată de solicitarea simultană a aceleiaşi căi de comunicare în RIN. Întârzierea în rutare şi ocuparea canalului formează aşa-numita *latență neîncărcată* a rețelei pentru dimensiuni de mesaje tipice [CSG99].

Latența neîncărcată pentru un pachet de *n* byte, incluzând antetul, într-o rutare *store and forward* este:

$$T_{sf}(n,h) = h\left(\frac{n}{b} + \Delta\right) \tag{1.2}$$

unde:

h – distanță de rutare ce reprezintă numărul de canale din calea de parcurgere

b – lățimea de bandă a liniei dintr-un canal

Δ – întârziere adițională în rutare

Această ecuație sugerează că topologia RIN este un element extrem în determinarea latenței RIN deoarece topologia determină în mod fundamental distanța de rutare *h*. În realitate situația este mult mai complicată depinzând de mai mulți factori printre care strategia de comutare este determinantă. De exemplu, în literatură [CS99] se citează latența neîncărcată în RIN în unități de timp de rețea, τ, pentru un mesaj de *n* byte ce parcurge distanța *h* într-o RIN comutată în circuit ca fiind:

$$T_{CS}(n,h) = \frac{n}{b} + h\Delta \tag{1.3}$$

În această ecuație se observă situația în care la creşterea lungimii mesajului, distanța de rutare şi prin urmare topologia, devine o fracțiune nesemnificativă a latenței de comunicare.

În ceea ce priveşte latența în RIN cu comutare de pachet, se observă că un mesaj lung poate fi fragmentat în câteva pachete mici ce parcurg rețeaua într-o manieră *pipeline*. În acest caz, latența neîncărcată este:

$$T_{sf}(n,h,n_p) = \frac{n-n_p}{b} + h\left(\frac{n_p}{b} + \Delta\right) \tag{1.4}$$

unde n_p reprezintă dimensiunea fragmentelor de mesaj.

Această abordare a fost adoptată software în rețelele de comunicare tradiționale ca de exemplu Internet.

Discuția precedentă referitoare la latența de comunicare se referă la sitația când un mesaj curge de la sursă la destinație fără a întâlni alt traflc în cale. În această situație RIN poate fi văzută simplu ca şi o structură pipeline. Diferitele strategii de comutare şi de rutare schimbă structura efectivă a pipeline-ului în timp ce topologia, lățimea de bandă a conexiunii şi fragmentarea determină adâncimea şi timpul la nivel de etaj al RIN.

Desigur, rațiunea pentru care RIN sunt atât de interesante din punct de vedere arhitectural este aceea că ele nu sunt simple structuri pipeline ci mai curând rețele întrețesute de porțiuni de mai multe pipeline-uri. Motivația principală pentru utilizarea unei RIN în locul unei magistrale este aceea de a permite multiple transferuri de date de pot apărea simultan. Aceasta înseamnă că un mesaj care parcurge rețeaua poate să intre în coliziune cu altele şi să apară un conflict pentru resurse. Comportamentul în cazul situației conflictuale depinde de câteva fațete ale proiectului rețelei – topologia, strategia de comutare şi algoritmul de rutare – dar elementul esențial îl reprezintă faptul că la orice moment dat un canal poate fi ocupat doar de un mesaj. Dacă două mesaje tind să utilizeze acelaşi canal simultan, unul trebuie să fie îndepărtat. În mod tipic, fiecare comutator furnizează câteva mijloace de arbitrare pentru canalele de ieşire. Prin urmare un comutator va selecta unul dintre pachetele ce sosesc la intrare şi care concurează pentru fiecare ieşire iar celelalte vor fi înlăturate pentru o tratare ulterioară în aceeaşi manieră.

În rețelele de comunicare din calculatoarele paralele, un pachet este mai repede blocat în tampon decât îndepărtat; această procedură presupune existența unei tranzacții între portul de ieşire şi portul de intrare de-a lungul unei conexiuni, ceea ce reprezintă în ultimă instanță un control al fluxului la nivel de conexiune.

Există o numeroasă literatură care tratează atât de importantul, însă în acelaşi timp atât de delicatul subiect al latenței, dar credem că în lucrarea de referință [CSG99] sunt sintetizate principalele aspecte legate de latență şi de lățimea de bandă într-o corectă şi performantă proiectare a RIN.

Lățimea de bandă

Lățimea de bandă este un parametru critic în evaluarea programelor paralele, pe de o parte datorită faptului că o lățime de bandă mai mare diminuează gradul de ocupare, iar pe de altă parte datorită faptului că o lățime de bandă mai mare reduce probabilitatea de conflict şi, în ultimă instanță, datorită faptului că porțiuni din program pot să prelucreze

un volum mare de date fără să aştepte ca transmisia datelor individuale să se termine. Deoarece RIN se comportă ca şi structurile pipeline, este posibil să se furnizeze lăţime de bandă mai mare chiar şi când latenţa este mare.

Este util să se privească lăţimea de bandă din două puncte de vedere: lăţimea de bandă "globală" disponibilă la toate nodurile din cadrul RIN şi lăţimea de bandă individuală "locală" disponibilă la un nod. Dacă volumul de comunicare total al unui program este de M bytes iar lăţimea de bandă a comunicării din RIN este B bytes/secundă atunci este evident că timpul de comunicare este cel puţin M/B secunde. Pe de altă parte, dacă toată comunicarea se produce spre sau dinspre un singur nod, această estimare este prea optimistă, în sensul că timpul de comunicare va fi determinat de lăţimea de bandă a unui singur nod.

Lăţimea de bandă totală a tuturor canalelor în reţea este Cb bytes/secundă sau Cw biţi/ciclu sau C unităţi fizice de comunicare/ciclu. Dacă fiecare din cele N noduri livrează un pachet la fiecare M cicluri cu o distanţă medie de rutare h, atunci fiecare pachet ocupă în medie h canale pentru $l=n/w$ cicluri iar sarcina totală a RIN este: Nhl/M unităţi fizice/ciclu. Utilizarea medie a canalului este:

$$\rho = M\frac{C}{Nhl} < 1 \qquad (1.5)$$

O modalitate de a privi această formulă este una în termeni de număr de conexiuni/nod, C/N, ce reflectă lăţimea de comunicaţie disponibilă în medie pe fiecare nod. Lăţimea de comunicaţie este consumată într-o proporţie directă cu distanţa de rutare şi cu talia mesajului. Numărul de legături/nod este o proprietate statică a topologiei. Distanţa de rutare medie este determinată de topologie, de algoritmul de rutare, de configuraţia de comunicaţie a programului şi de maparea programului în sistem. O bună localitate a comunicaţiei poate utiliza un h mic în timp ce o comunicaţie aleatoare va traversa distanţa medie care se poate întinde pe întreg diametrul reţelei. Dimensiunea mesajului este determinată de comportamentul programului şi de modelul comunicaţiei.

În practică, câţiva factori limitează utilizarea canalului ρ ce devine mult subunitar. Sarcina poate să nu fie perfect echilibrată pe toate căile de comunicare ale RIN. Chiar dacă sarcina este echilibrată, algoritmul de rutare poate să reducă utilizarea conexiunilor pentru o configuraţie particulară de comunicare utilizată de program. Şi chiar în situaţia când toate conexiunile sunt utilizabile şi sarcina este echilibrată, pot apare variaţii

stochastice în ceea ce priveşte sarcina şi conflictul pentru resurse de nivel inferior. Toţi aceşti factori afectează *punctul de saturare* ce reprezintă lăţimea totală a canalului ce poate fi furnizată în mod util. Aşa cum se ilustrează în fig.1.9, dacă solicitarea de lăţime de bandă plasată pe reţea de către procesoare – denumită *lăţime de bandă oferită* – este moderată, latenţa rămâne coborâtă şi lăţimea de bandă furnizată creşte cu lăţimea de bandă oferită.

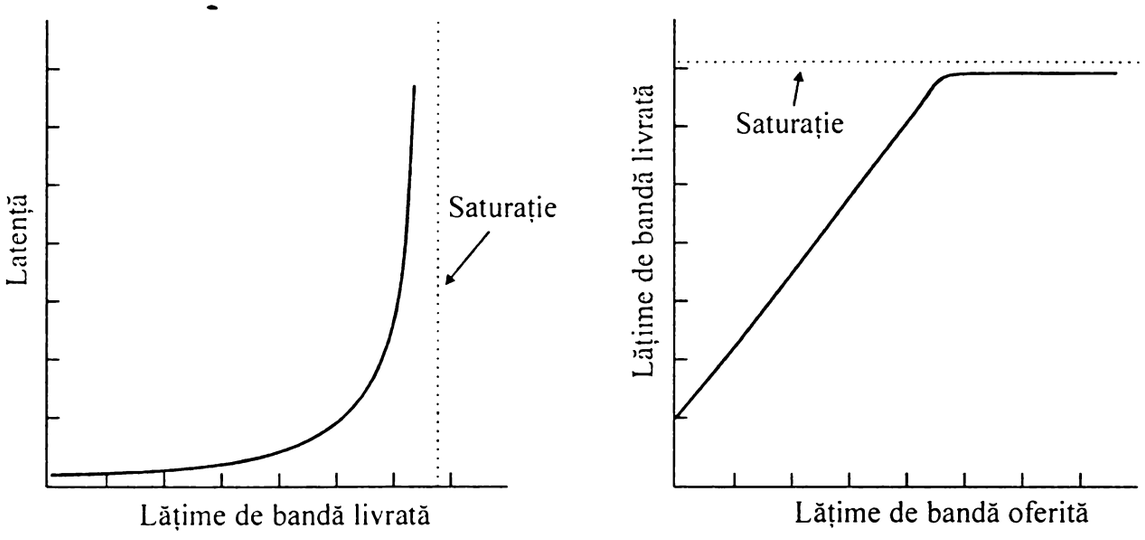


Fig. 1.9 Comportamentul la saturaţie pentru o reţea de interconectare tipică

La un anumit punct, solicitarea pentru o lăţime de banda superioara nu face decat sa crească conflictul pentru resurse iar latenţa creşte dramatic. Creşterea lăţimii de bandă oferită nu creşte lăţimea de bandă furnizată. Pentru a preveni această situaţie de saturare a reţelei se limitează fie solicitările plasate de către procesoare, fie se asigură o lăţime de bandă de comunicare amplă.

Modelarea performanţei la reţelele maşinilor paralele demonstrează că latenţa şi lăţimea de bandă a reţelelor reale depind sub toate aspectele de schema reţelei. Modelarea performanţei reţelelor este ea însăşi o arie bogată cu o literatură voluminoasă [BYA89], [BINK97], [DB94], [KS83], [YLL90], [BRA89]. Aceste studii consideră reţeaua ca fiind izolată de context şi sunt bazate pe modele analitice ce presupun modele simple ale traficului de date prin reţea. Este dificil să se formalizeze toate detaliile unei reţele într-un simplu model analitic sau într-unul de simulare. Prin urmare, aceste modele nu iau în considerare diferitele tipuri de mesaje, traficul de invalidare asociat cu coerenţa memoriilor cache şi cu sincronizarea, acestea fiind specifice unui mediu multiprocesor. În plus, modelele prezentate evaluează reţeaua în termeni de *capacitate de trecere* şi de

întârziere a mesajului, în loc de *timp de răspuns* al aplicației. În fine, rețelele de interconectare au evoluat semnificativ odată cu introducerea unor tehnici revoluționare cum ar fi *canalele virtuale* [Dal92].

Este important de adăugat că performanța nu este singurul criteriu dominant în proiectele de rețele. Alte două criterii critice sunt *costul* şi *toleranța la defect* asupra cărora se va insista în capitolele următoare.

1.5 TOLERANȚA LA LATENȚĂ

Există diferențe de ritm în ceea ce priveşte creşterea vitezei microprocesoarelor în comparație cu timpul de acces al memoriilor utilizate. În timp ce viteza a crescut mai mult de 10% pe decadă, timpul de acces a crescut doar pe jumătate. Prin urmare, latența accesului la memorie exprimată în termeni de cicli de procesor creşte cu mai mult de 20% într-o perioadă de 10 ani. În sistemele bazate pe magistrală, latența creşte datorită procedurii de monitorizare a magistralei. În sistemele cu memorie distribuită, latența RIN, interfața rețelei se adaugă la timpul de accesare a memoriei locale din nod. Memoriile cache ajută la reducerea frecvenței acceselor cu latență ridicată dar ele nu reprezintă o soluție universală: ele nu reduc comunicația inerentă şi programele au o rată de eşecuri semnificativă.

În proiectarea SMP, unul dintre scopurile protocoalelor de menținere a coerenței este acela de a reduce frecvența evenimentelor cu latență mare şi cu solicitări de lățime de bandă impuse în mediul de comunicare în timp ce se asigură un model de programare convenabil. Scopul proiectării suportului hardware în cadrul SMP este acela de a reduce latența accesului de date în timp ce se menține o lățime de bandă scalabilă mare. În mod uzual se poate ameliora lățimea de bandă prin tehnici hardware, de exemplu utilizând conexiuni mai largi sau topologii mai generoase. Dar latența rămâne limitarea fundamentală.

În proiectarea arhitecturii unui SMP există trei căi pentru a reduce latența efectivă a accesului de date – primele două căzând în responsabilitatea sistemului iar a treia în responsabilitatea aplicației.

1. *Reducerea timpului de acces la fiecare nivel al unei ierarhii extinse de memorie*. Această tehnică pretinde acordarea unei atenții deosebite în ceea ce priveşte eficientizarea fiecărui pas din calea de acces. Interfața procesor-cache

poate fi proiectată să funcţioneze foarte strâns. Controllerul cache trebuie să acţioneze rapid în cazul unui eşec de acces pentru a reduce penalizarea în timp atunci când accesează nivelul următor. Interfaţa cu RIN poate fi strâns cuplată cu nodul şi proiectată pentru a forma, livra şi manipula rapid tranzacţiile prin reţea. Reţeaua propriu-zisă poate fi proiectată pentru a reduce întârzierile de rutare, timpul de transfer şi întârzierile datorită congestiei de trafic. Printr-o proiectare atentă trebuie să se înţeleagă însă că nu pot fi depăşite latenţele inerente tehnologiei. Evident, tolerarea latenţei prin această tehnică adaugă costuri suplimentare.

2. *Structurarea sistemului pentru a reduce frecvenţa acceselor cu latenţă ridicată*. Aceasta este sarcina de bază a replicării automate ca şi aceea ce se produce în memoriile cache unde se iau în considerare avantajele localităţii spaţiale şi temporale în configuraţia de acces a programului, în sensul că cele mai importante date trebuie să fie cât mai aproape de procesor. Se poate face replicarea mult mai efectivă prin adaptarea structurii sistemului multiprocesor; de exemplu prin furnizarea unei cantităţi substanţiale de capacitate de înmagazinare fiecărui nod al sistemului.
3. *Structurarea aplicaţiei pentru a reduce frecvenţa acceselor cu latenţă ridicată*. Aceasta presupune descompunerea şi atribuirea sarcinii de calcul procesoarelor în scopul reducerii comunicării inerente şi structurării configuraţiilor de acces pentru a creşte localitatea spaţială şi temporară.

În plus faţă de comunicare şi accesul de date, există alte evenimente cu un potenţial de latenţă ridicată ca de exemplu sincronizarea pentru care trebuie făcute eforturi similare. Aceste eforturi în ameliorarea latenţei sistemului şi a aplicaţiei, deseori nu sunt suficiente. În acest sens o altă abordare [CSG99] este aceea de a *tolera latenţa remanentă*. Aceasta înseamnă estomparea latenţei din drumul critic al procesorului suprapunând-o cu calculul sau cu alte evenimente cu latenţă ridicată. Procesorului i se permite să efectueze alte sarcini utile sau chiar acces de date şi comunicare în timp ce evenimentul cu latenţă ridicată este în evoluţie. Cheia tolerării latenţei este în ultimă instanţă *paralelismul* deoarece activităţile suprapuse trebuie să fie independente una de cealaltă.

1.5.1 Cerințe fundamentale, beneficii și limitări ale toleranței la latență

Cerințe

Tolerarea latenței necesită un *extraparalelism*, o *lățime de bandă mărită* și, în cele mai multe dintre cazuri, un *hardware mult mai sofisticat* și *protocoale elaborate*.

- *Extraparalelism.* Deoarece activitățile suprapuse trebuie să fie independente una față de cealaltă, paralelismul în aplicație trebuie să fie mai mare decât numărul de procesoare utilizate. Un paralelism suplimentar poate fi explicitat sub forma unor fluxuri adiționale, ca și în sistemele multiflux, sau poate să existe în cadrul unui flux. Chiar comunicarea a două cuvinte din același proces în paralel implică o lipsă a unei serializări totale dintre ele, și prin urmare un extraparalelism.
- *Lățime de bandă mărită.* În timp ce toleranța la latență poate să reducă timpul de execuție, nu se reduce însă cantitatea de comunicație. Aceeași comunicare efectuată într-un timp mai mic, înseamnă o rată de comunicare mai mare și, prin urmare, se impun lățimi de bandă mai mari în arhitectura de comunicație.
- *Un hardware mai sofisticat și protocoale mai elaborate.* Cu excepția cazului producerii unor mesaje mai mari, procesorului trebuie să i se permită să efectueze o operație cu latență mai mare înainte ca operația să se termine.

Aceste cerințe implică ca toate tehnicile de tolerare a latenței să aibă un cost relativ semnificativ. Prin urmare, proiectantul trebuie să utilizeze tehnici algoritmice pentru a reduce frecvența evenimentelor cu latență ridicată înainte de a se baza pe tehnici de tolerare a latenței. Cu cât mai puține sunt evenimentele cu latență ridicată, cu atât mai puțin agresivă va fi tendința de a estompa această latență.

Beneficii potențiale

O simplă analiză poate oferi limitele în beneficiile de performanță ce pot fi așteptate de la tolerarea latenței. Preuspunându-se că nu se utilizează tehnici de tolerare a latenței, timpul de execuție, așa cum este el văzut de un procesor, are următorul profil: T_C cicli sunt consumați în calcul local, T_{OV} cicli sunt consumați în procesarea mesajului, T_{OCC} cicli în asistarea comunicării și T_I cicli în așteptarea transmisiei mesajului în RIN. Dacă se presupune că celelalte resurse pot fi perfect suprapuse cu activitatea din procesorul principal atunci creșterea potențială de viteză poate fi determinată prin aplicarea simplă a legii lui Amdahl. Procesorul poate fi ocupat pentru T_C+T_{OV} cicli; latența maximă ce

poate fi estompată de către procesor este T_l+T_{OCC} prin urmare creşterea maximă de viteză datorită estompării latenței este:

$$\left(1+\frac{T_{OCC}+T_l}{T_C+T_{OV}}\right) \tag{1.6}$$

Această limită reprezintă un prag superior deoarece se presupune o perfectă suprapunere a resurselor şi nu este impus nici un cost suplimentar datorită toleranței la latență.

Cât de multă latență poate fi efectiv estompată depinde de mulți factori ce implică aplicația şi arhitectura. Caracteristici de aplicație relevante includ structura comunicației şi cât de multe alte activități pot fi suprapuse cu ea.

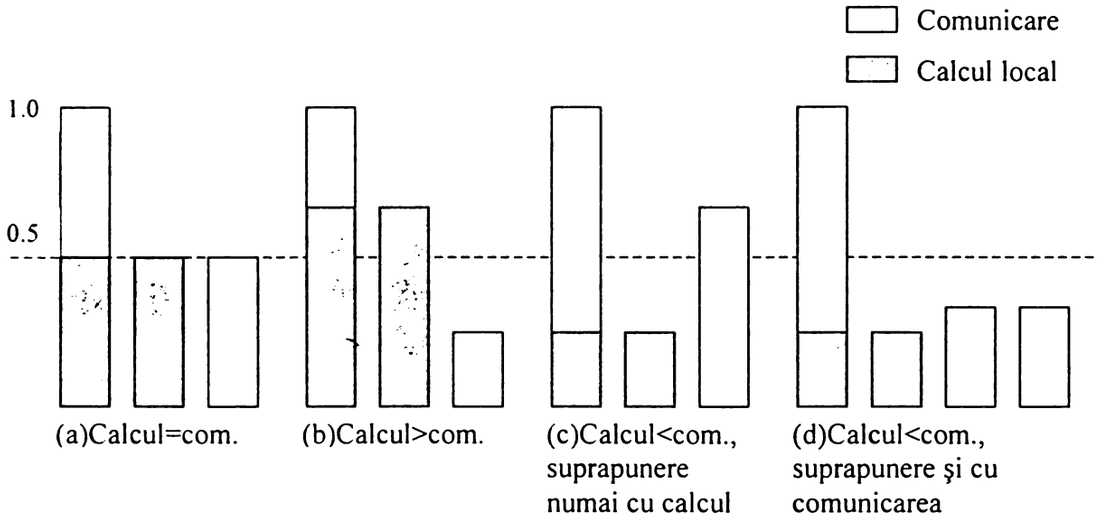


Fig.1.10 Limitări şi beneficii ale toleranței la latență

Limitări

Limitările majore pot fi divizate în trei clase: *limitări de aplicație*, *limitări ale arhitecturii de comunicație* şi *limitări de procesor*.

Limitări de aplicație. Cantitatea de timp de calcul independent ce este disponibilă pentru suprapunere cu latența poate fi limitată, astfel că nu toată latența poate fi estompată iar procesorul va trebui să stagneze pentru o anumită perioadă de timp. Chiar dacă programul are suficientă sarcină utilă şi extraparalelism, structura programului poate să facă dificil pentru sistem sau pentru programator identificarea operațiunilor concurente şi să orchestreze suprapunerea.

Limitări ale arhitecturii de comunicație. Arhitectura de comunicație poate să restricționeze numărul de mesaje sau numărul de cuvinte ce pot fi extrase dintr-un nod la

un moment dat şi, prin urmare, parametrii de performanţă ai arhitecturii de comunicaţie pot limita latenţa ce trebuie să fie estompată.

În literatură [CSG99] se tratează pe larg subiecte legate de limitarea latenţei ce poate fi mascată datorită specificităţilor arhitecturale.

Limitări ale procesorului. În SMP-MP ce utilizează mecanisme de asigurare a coerenţei memoriilor cache, localitatea spaţială asigurată prin blocuri cache lungi permite extragerea la un moment dat a mai multor cuvinte de comunicaţie. Pentru a estompa latenţa, un procesor şi subsistemul său cache trebuie să permită apariţia mai multor eşecuri în acces la memoria cache în mod simultan. Acest lucru este însă costisitor aşa că sistemele procesor-cache au limitări relativ reduse în ceea ce priveşte numărul de eşecuri extrase.

Este clar din cele prezentate că transformarea unei arhitecturi de comunicaţie într-una eficientă reprezintă o operaţiune foarte importantă pentru tolerarea efectivă a latenţei.

În continuare menţionăm că o tratare sistematică a relaţiei complexe dintre arhitectură, toleranţă la latenţă şi performanţă excede scopul acestei dizertaţii, ea putând însă genera subiecte majore de reflexie pentru proiectanţii de sisteme de calcul paralele.

În concluzie, trebuie să remarcăm că în situaţia creşterii discrepanţei dintre vitezele procesoarelor şi timpii de acces la memorie şi timpii de comunicaţie, toleranţa latenţei va deveni un subiect critic pentru viitoarele multiprocesoare. Au fost dezvoltate multe tehnici de tolerare a latenţei şi fiecare îşi are avantaje şi dezavantaje relative. Toate aceste tehnici se bazează pe concurenţa în exces în programe de aplicaţie în cadrul numărului de procesoare utilizate şi toate tind să crească cererile de lăţime de bandă plasată pe arhitectura de comunicaţie.

Pentru multiprocesoarele coerente din punct de vedere cache, tehnicile de tolerare a latenţei sunt suportate hardware atât de procesor cât şi de sistemul memoriei cache şi conduc la un spaţiu generos de alternative de proiectare. Multe dintre aceste tehnici de toleranţă a latenţei suportate de către hardware sunt de asemenea aplicabile la uniprocesoare. Tehnici ca: *planificarea dinamică, modele de consistenţă a memoriei relaxate*, *preaducerea datelor din memorie*, etc. sunt frecvent întâlnite în arhitecturile microprocesoarelor contemporane. Cea mai generală tehnică de mascare a latenţei, cea care presupune crearea de fluxuri multiple, nu este încă populară din punct de vedere comercial deoarece ea nu a fost încă testată pe uniprocesoare.

În pofida acestui spațiu generos de alternative de proiectare multe dintre problemele cotidiene ale toleranței latenței sunt de asemenea probleme susținute prin tehnici software. Automatizarea şi simplificarea suportului software cerut pentru asigurarea toleranței la latență este o sarcină departe de a fi fost rezolvată definitiv. În ultimă instanță, raportul dintre tehnicile de tolerare a latenței suportate prin resurse hardware şi cele suportate software reprezintă un subiect deschis în arhitecturile paralele.

1.6 TOLERANȚA LA DEFECT

Indiferent de tendințele arhitecturale abordate în proiectarea şi construcția SMP, *toleranța la defect* va reprezenta unul dintre principalele obiective urmărite de proiectanți. Noile tehnologii, indiferent cât de performante sunt, nu asigură o siguranță în funcționare absolută. Prin urmare, luarea în considerare a toleranței la defect în stabilirea indicelui de performanță reprezintă o necesitate stringentă ce nu poate fi ignorată în nici un fel.

Fiabilitatea şi *disponibilitatea* sistemelor de calcul constituie aspecte majore în proiectarea lor. Relativ recenta descoperire a unei erori de proiectare în microprocesorul Pentium a revelat complexitatea măsurilor de asigurare a fiabilității în utilizarea microprocesoarelor avansate.

Oportunitatea pentru o recuperare rapidă în cazul apariției unei defecțiuni este consistent ajustată de apariția microprocesoarelor de mare viteză, dar apar noi provocări în ceea ce priveşte asigurarea unei sincronizări fabile. O altă provocare pentru proiectarea SMP tolerante la defect o reprezintă creşterea utilizării sistemelor masiv paralele. În dorința de a obține cea mai mare performanță posibilă cu ajutorul arhitecturilor inovative, aceste sisteme sunt adeseori bazate pe tehnologii neprobate. Nu este surprinzător faptul că diferențele în dependabilitate, prezentate în fig. 1.11 sunt atât de mari. Un cost de necontestat al lipsei de toleranță la defect este pierderea puterii de calcul în calculul de mare performanță. O mică fracțiune din timpul operațional pierdut datorită defectelor ar putea genera rapid o pierdere în performanță care ar transforma SMP într-un singur calculator.

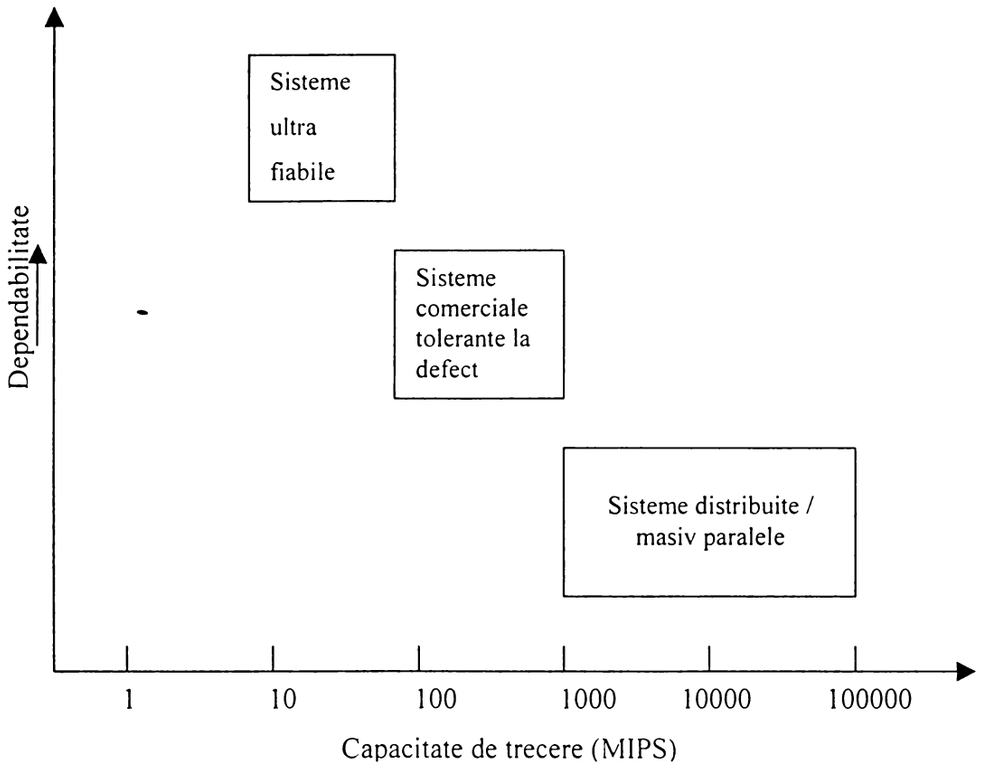


Fig. 1.11 Oferta de SMP în functie de dependabilitate/performantă

Prin urmare, trebuie asigurată toleranța la defect fără o pierdere semnificativă în performanță.

Toleranța la defect în SMP complexe trebuie să incorporeze atât tehnici hardware cât şi software. Cerințele de dependabilitate sunt în principal ierarhizate astfel, în funcție de nivelul căruia i se adresează:

- La nivel de *circuit* şi de *tehnologie*, ținând cont de tehnicile de proiectare a circuitelor pentru a ameliora testabilitatea şi fiabilitatea modulelor individuale.
- La nivel de *arhitectură a nodului*, ținând cont de proiectarea capsulelor VLSI ce implementează un nod de procesare.
- La nivel de *arhitectură între noduri*, specificându-se modul în care aceste noduri sunt conectate împreună şi maniera în care sistemul va fi reconfigurat în prezența procesoarelor defecte. Se ia în considerare topologia de interconectare.
- La nivel de *sistem de operare*, ținând cont de crearea şi exploatarea *punctelor de verificare* – PV, de *rulare înapoi* după ce o porțiune din sistem a fost identificată ca fiind defectă. Sarcina procesorului defect se distribuie altor procesoare.
- La nivel de *aplicație* unde utilizatorul verifică rezultatele calculului.

În mod tipic, toleranța la defect este căutată la fiecare nivel al proiectului. Defectele nedetectate la nivelele inferioare trebuie să fie manipulate la nivele superioare.

1.6.1 Implicații ale toleranței la defect în arhitecturile MIMD

În SMP-MP procesoarele sunt *anonime* în sensul că, *in extremis*, toate datele şi instrucțiile staționează într-o memorie comună disponibilă fiecărui procesor. SMP-MP dispun de memorii cache pentru a micşora latența şi a creşte lățimea de bandă. Cu cât talia memoriilor cache creşte rolul memoriei centrale se reduce iar SMP-MP tinde să se comporte ca şi un SMP-TM. Memoria cache reduce astfel anonimatul procesorului. Consistența memoriilor cache ridică probleme speciale în ceea ce priveşte toleranța la defect deoarece în timpul defectărilor, este afectată coerența cache între diferite memorii cache dispuse pe procesoare diferite. În Capitolul 4, în cadrul secțiunii ce tratează recuperarea din defect la SMP-MP, va fi discutat acest subiect în detaliu. O altă problemă importantă ce apare la SMP-MP, o reprezintă *propagarea defectului*. Un procesor defect poate potențial să scrie în orice locație a memoriei centrale şi prin urmare să afecteze conținutul memoriei, mai puțin în situația în care se efectuează o verificare înainte de fiecare scriere în memoria principală.

Datorită faptului că SMP-TM comunică prin mesaje, sunt necesare în primul rând verificări ale corectitudinii transmiterii mesajelor. Acestea sunt mai puțin frecvente decât cele efectuate asupra execuției instrucțiilor. Astfel, din punct de vedere al protecției la erori, SMP-TM se comportă mai bine.

Abordările de bază specifice în ceea ce priveşte toleranța la defect în SMP sunt două. În *redundanța statică* în care N copii ale fiecărui procesor sunt utilizate iar gradul minim de replicare este triplicarea.

În *redundanța dinamică*, prima dată este detectat procesorul defect care ulterior este înlocuit cu unul de rezervă.

2. ANALIZA ARHITECTURILOR SISTEMELOR DE CALCUL DIN CLASA MIMD

2.1 ASPECTE GENERALE

Paralelismul este un termen general utilizat pentru caracterizarea unei varietăți de simultaneități ce apar în calculatoarele moderne. Avantajul major oferit de paralelism îl reprezintă viteza crescută a sistememelor de calcul bazate pe el.

Procesarea paralelă include atât *algoritmi paraleli* cât şi *arhitecturi paralele*.

Un algoritm paralel poate fi văzut ca şi o colecție de module independente dintre care unele pot fi executate simultan. Modulele comunică între ele în timpul executării unui algoritm. Deoarece algoritmii evoluează într-un mediu paralel din punct de vedere hardware, care constă din module de procesare ce schimbă între ele date, este o cale naturală abordarea relației dintre *spațiul algoritmului* şi *spațiul hardware*. Cu cât această relație este mai bună şi mai nuanțată, cu atât mai bine se pot proiecta sisteme de calcul paralel.

Criteriile pricipale după care se pot caracteriza algoritmii sunt :

- *Granularitatea modulului* (*module granularity*);
- *Controlul concurenței* (*concurrence control*);
- *Mecanismul de utilizare a datelor* (*data mechanism*);
- *Geometria de comunicare* (*comunication geometry*);
- *Dimensiunea algoritmului* (*algoritm size*).

Arhitecturile paralele pot fi caracterizate utilizând următoarele criterii :

- *Complexitatea procesorului*;
- *Modul de operare*;
- *Structura memoriei*;
- *Rețeaua de interconectare*;
- *Numărul de procesoare şi talia memoriei*.

În literatură [Mold 93] se explicitează relația dintre caracteristicile din cele două spații.

În tabelul din fig. 2.1 se prezintă succint aceste relații :

Algoritmi paraleli	*Arhitecturi paralele*
Granularitatea modulului	Complexitatea procesorului
Controlul concurenței	Modul de operare
Mecanismul de date	Structura memoriei
Geometria de comunicare	Rețeaua de interconectare
Talia algoritmului	Numărul de procesoare şi talia memoriei

Fig. 2.1 Relația dintre caracteristicile algoritmilor paraleli şi arhitecturile paralele

În literatură [Dasg 89], [Hwan 93], [Prad 96] etc se utilizează mai multe taxonomii pentru clasificarea sistemelor de calcul.

Ne interesează evoluția taxonului MIMD – Multiple Instruction Multiple Data – definit prima dată de Flynn (1966) şi apoi nuanțat de Hwang-Briggs (1984). Aceştia modifică clasificarea lui Flynn introducând subtaxonii:

- MIMD–T (Multiple Instruction Multiple Data – Thightly) – Sistem strâns cuplat;
- MIMD–L (Multiple Instruction Multiple Data – Loosely) – Sistem slab cuplat.

Această partajare s-a făcut în funcție de modul de utilizare a memoriei de către procesoare, respectiv de gradul ei de utilizare în procesul de comunicare.

Organizarea sistemelor de calcul din subclasa MIMD – T este prezentată în fig. 2.2, iar a celor din subclasa MIMD – L în fig. 2.3.

Se observă că în primul caz mediul de conectare – *magistrala* – este între procesoare şi memorie, în timp ce în al doilea caz memoria este ataşată fiecărui procesor şi mediul de conectare este *rețeaua de interconectare* ce face conectarea modulelor combinate.

Există serioase dezbateri asupra modului de organizare a memoriilor în sistemele paralele de talie mare. Dezbaterea se centrează, şi nu întotdeauna justificat, pe o falsă dihotomie: *memoria partajată* versus *memoria distribuită*. Memoria partajată înseamnă un singur spațiu de adresă, presupunând implicit o comunicare prin instrucții LOAD/STORE. In opoziție, în raport cu o singură adresă se află spații multiple de adrese private, ceea ce implică comunicare prin SEND/RECEIVE. Memoria distribuită se referă la localizarea fizică a memoriei. Dacă memoria fizică este divizată în module, cu câteva plasate lângă fiecare procesor, atunci memoria fizică este distribuită.

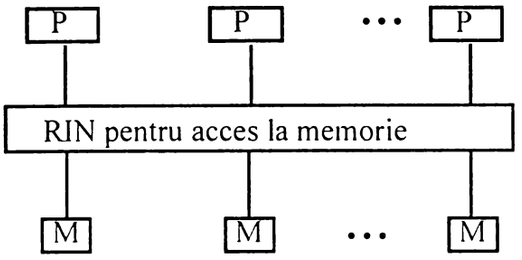


Fig 2.2 SMP cu memorie partajată, din subtaxonul MIMD-T

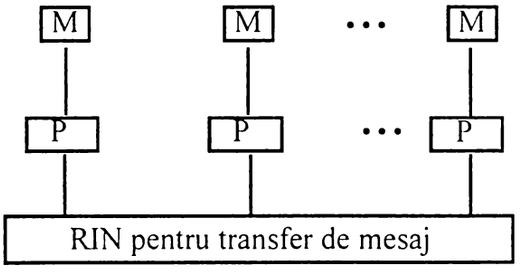
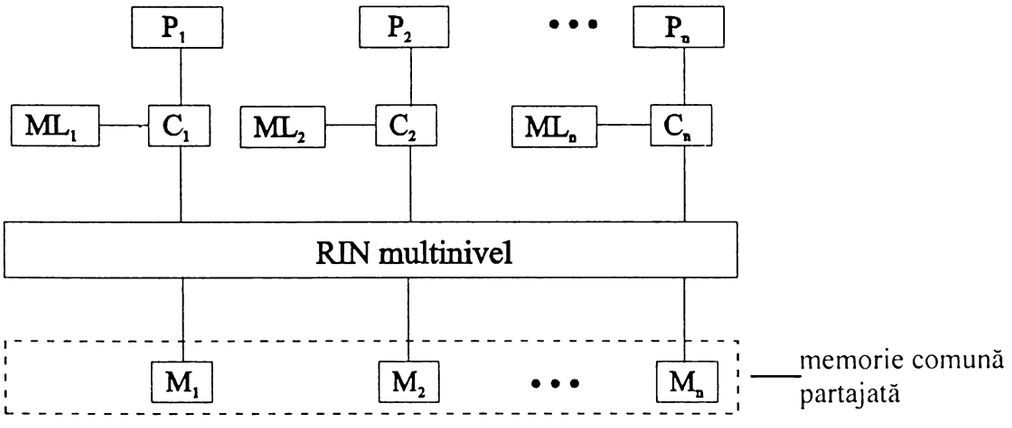


Fig. 2.3 SMP cu memorie distribuită, din subtaxonul MIMD-L



P - procesor
M - modul de memorie
C - memorie cache privată
ML - memorie locală

Fig. 2.4 SMP-MP cu arhitectură tip "sală de dans"

Opusul memoriei distribuite îl constituie *memoria centralizată*, unde timpul de acces la o locație de memorie fizică este același pentru toate procesoarele deoarece fiecare acces se face prin rețeaua de conectare ca și în fig. 2.4. Această arhitectură este –după unii autori– denumită *sală de dans* (*dance hall*) cu procesoarele dispuse pe o latură iar memoriile pe alta. Analogia este mai mult decât evidentă.

Spațiul singular de adresă versus spații multiple de adresă și memorie distribuită versus memorie centralizată sunt subiecte ortogonale: *Mașinile MIMD pot avea un singur spațiu de adresă și o memorie distribuită sau spații cu adresă privată multiple și o memorie fizică centralizată.*

Pentru simplificare însă, vom considera uzual sistemele din subclasa MIMD – T ca având o memorie globală partajată prin intermediul căreia procesoarele își asigură *comunicarea* și *coordonarea*.

Le vom denumi *sisteme multiprocesor cu memorie centrală partajată* (SMP-MCP) sau pe scurt *multiprocesoare*. Un multiprocesor se consideră a fi *fiabil pentru multiprocesare* (*multiprocessing reliable*) dacă și numai dacă poate fi utilizat atât pentru operații MIMD cât și pentru M/SISD (Multiple/Single Instruction Single Data) conform [Hwan 93].

Reamintim că atât sistemele M/SISD cât și cele SISD sunt cazuri degenerate ale celor MIMD.

Exemple de calculatoare din subclasa MIMD-T sunt: C.mmp construit la Carnegie Mellon University, Multimax de la Encore Computer, FX de la Alliant, etc.

In sistemele din subclasa MIMD-L, memoria principală este partiționată și atașată procesoarelor. Acestea împart același spațiu adresă de memorie. Vor fi denumite *sisteme multiprocesor cu memorie centrală distribuită* (SMP-MCD).

Exemple de arhitecturi MIMD-L sunt Cm* de la CMU, Butterfly de la BBN Laboratories, RP3 de la IBM ș.a. Atât calculatoarele MIMD-T cât și cele MIMD-L vor fi denumite *multiprocesoare*, *sisteme multiprocesor cu memorie partajată* și abreviate SMP-MP.

Multicalculatoarele se caracterizează prin faptul că memoria nu este partajată. Interacțiunea dintre procesoare se produce prin mesaje ce sunt transferate între procesoarele sursă și destinație (noduri). Aceste mesaje traversează legătura ce conectează direct două noduri și pot să traverseze prin câteva noduri într-o manieră

înmagazinează şi înaintează (*store and forward*) înainte de a-şi atinge destinaţia. Multicalculatoare – *sisteme multiprocesor ce transferă mesaje* (SMP-TM) se bazează uzual pe topologii ca şi *inel*, *arbore*, *stea*, *hipercub*, etc.

Exemple de multicalculatoare sunt: Caltech Cosmic Cube, Intel IPCS1, AMETEK System 1, Intel Tuchstone ş.a.

Prin urmare o clăsificare a sistemelor paralel distribuite este [BYA 89] prezentată în fig. 2.5.

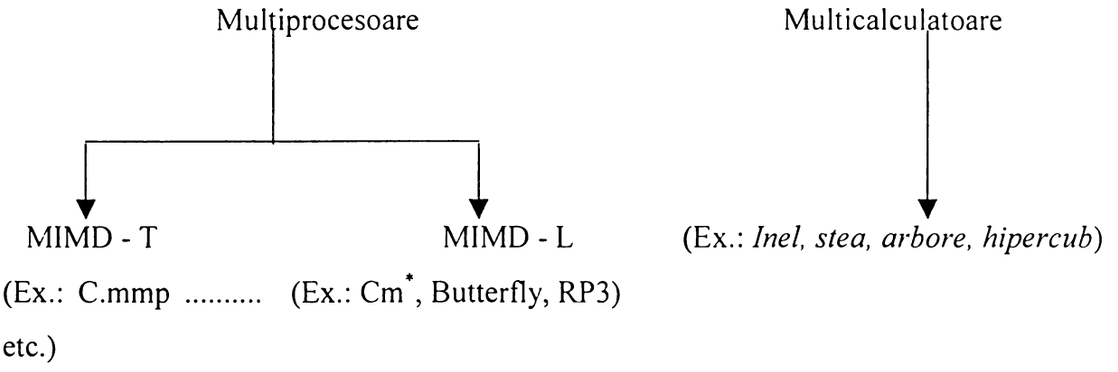


Fig. 2.5 O clasificare a sistemelor paralel/distribuite.

Clasificarea nu include *ariile de calculatoare* (*array computers*) şi cele cu *prelucrare pe loturi* (*pipeline computers*) cât şi *reţelele locale de calculatoare* (*Local Area Networks*). Ariile de calculatoare şi cele cu prelucrare pe loturi sunt cu prelucrare paralelă dar nu distribuită în timp ce LAN fac parte din prelucrarea distribuită dar nu şi din cea paralelă.

Proiectarea oricărei arhitecturi multiprocesor implică rezolvarea unui set fundamental de probleme, unele dintre ele fiind noi pentru contextul multiprocesării în timp de altele sunt moştenite din domeniul uniprocesor.

Cele mai importante dintre aceste probleme sunt:

1. *Granularitatea paralelismului* ce trebuie suportat de multiprocesor.
2. *Mecanismele de sincronizare* pentru accesul la memoria partajată şi comunicarea între procese.
3. *Latenţa memoriei* şi rezolvarea sa.
4. Problema *reţelei de interconectare*.
5. *Strategiile de planificare* a procesorului/procesoarelor.

6. *Limbajele de programare concurente* şi *tehnicile de programare*.

Problemele de la 1 la 4 sunt specific arhitecturale. Arhitecturile sistemelor paralel / distribuite pot fi analizate şi sub alte aspecte cum ar fi:

- *expandabilitatea* – măsură a abilităţii sistemului de a prelua sarcini mărite;
- *capacitatea* – proprietatea reţelei de a putea mări cantitatea de date transferate atunci când se adaugă noi procesoare la sistem;
- *flexibilitatea* – posibilitatea de adaptare a sistemului la o modificare a fluxului de date.

Opinăm însă că o abordare modernă a sistemelor paralel/distribuite trebuie să se facă prin prisma *dependabilităţii*, definită simplu în [Prad96] ca fiind calitatea serviciilor oferite de un sistem de calcul.

Sistemele paralel/distribuite sunt adecvate pentru implementarea *toleranţei la defecte* (*fault-tolerance*) datorită resurselor hardware multiple din componenţă. Comportarea *rezistentă la defecte* (*fault resilient*) decurge din conectivitatea nodurilor şi a muchiilor grafului asociat sistemului. In multe situaţii toleranţa este posibilă doar cu reducerea performanţelor până la o limită admisibilă. Are loc o aşa numită *degradare graţioasă* (*graceful degradation*).

Intr-un sistem distribuit, diagnoza permite localizarea defectelor astfel încât sistemul să poată fi reconfigurat în jurul unităţilor operaţionale.

Degradarea graţioasă se obţine prin *configurarea automată* şi *capacitatea de recuperare* furnizate cu ajutorul unei mentenanţe on-line.

2.2 GRANULAŢIA DE PARALELISM

Paralelismul în calcule poate fi examinat la câteva nivele depinzând de complexitatea dorită. Acestea sunt: *program* (*job*), *sarcină* (*task*), *proces* (*process*), *variabilă* (*variable*) şi *bit*. Criteriul cel mai uzual de clasificare a sistemelor de calcul paralel este *granularitatea* (*granularity*) interacţiunilor dintre activităţile care sunt executate în paralel.

Considerând un sistem cu N procesoare în care o parte a timpului de lucru al fiecărui procesor este afectată schimburilor de date şi a informaţiilor de control cu alte procesoare se poate scrie:

$$T^c = \sum_{i=1}^{N} t_i^c \quad \text{şi} \quad T^w = \sum_{i=1}^{N} t_i^w$$

unde:

t_i^c - timpul consumat de P_i pentru comunicare;

t_i^w - timpul consumat de P_i pentru calcul;

T^c - timpul consumat de sistem pentru comunicare;

T^w - timpul consumat de sistem pentru calcul.

Considerând că nu există o lege de repartizare a timpului de prelucrare pentru fiecare procesor, se poate scrie *timpul total de prelucrare*:

$$T = T^c + T^w . \tag{2.1}$$

Granulația denumită şi *granulație de prelucrare* (*grain of computation*) este:

$$\gamma = \frac{T^w}{T^c} . \tag{2.2}$$

Granularitatea exprimă caracterul reunit al unui program paralel, a arhitecturii pe care se execută şi a dimensiunii problemei de rezolvat. Datorită acestui caracter reunit, dacă se modifică un parametru – de exemplu problema de rezolvat – se modifică şi granularitatea pentru o anumită perioadă de timp.

Din această cauză, pe lângă *granularitatea potențială* (*potential granularity*) se introduce şi *granularitatea actuală* (*actual granularity*) ca şi o medie a timpului afectat pentru două comunicări succesive:

$$\gamma' = \frac{T^w}{S} \tag{2.3}$$

unde:

$$S = \sum_{i=1}^{N} S_i$$

reprezintă numărul total de comutări de la calcul la comunicare pentru toate procesoarele P_i.

In general procesarea pe un sistem de calcul paralel se situează între două extreme:

- durate scurte de calcul + durate scurte de comunicare;
- durate lungi de calcul + durate lungi de comunicare.

In primul caz se defineşte o *granularitate fină* (*fine granularity*) iar în al doilea caz o *granularitate grosolană* (*coarse granularity*). Deşi granularitatea este un parametru complex, unele clase de sisteme favorizează un anumit timp de granulaţie. Cu cât mai fin este gradul granulaţiei cu atât creşte costul sistemului de comunicare şi deci costurile de comunicaţie.

Arhitectura din subclasa MIMD – T (*tightly coupled*) se caracterizează prin granulaţie fină ($\gamma < 10$) iar cele din clasa MIMD – L (*loosely coupled*) prin granulaţie grosieră ($\gamma > 10$). Evident granularitatea poate varia de la sistem la sistem.

Prin urmare multiprocesoarele din subclasa MIMD – T dispun de un *control centralizat* în timp ce multicalculatoarele din subclasa MIMD – L funcţionează pe principiul *autonomiei cooperative*.

2.3 LATENŢA MEMORIEI ŞI REZOLVAREA EI.

Prin *latenţă* (*latency*) se înţelege timpul consumat între o solicitare făcută de un procesor la memorie şi recepţionarea de către procesor a datelor asociate solicitării.

Latenţa memoriei poate reprezenta un potenţial de strangulare a traficului şi trebuie tratată în cel mai serios mod.

In cazul SMP-MP problema latenţei devine acută datorită:

- prezenţei mai multor procesoare ce pot solicita accesul la acelaşi modul de memorie, ceea ce poate conduce la *conflict de memorie* (*memory contention*);
- prezenţei RIN care introduce o întârziere inerentă între un procesor ce solicită un acces la memorie şi primirea datei;
- potenţialul conflictual în orice componentă particulară a RIN propriu-zise.

Cu cât mai mare va fi talia SMP-MP – în termeni de număr de procesoare – cu atât mai mare va fi latenţa medie.

In accest paragraf se vor prezenta câteva tehnici de *evitare* sau de *tolerare* a efectelor latenţei memoriei.

2.3.1 Comutarea de proces

O metodă posibilă de mascare a efectului latenței este de a efectua un proces sau un comutator "*context*" (*context switch*) ori de câte ori un proces P_i solicită un acces la memorie: P_i este suspendat şi procesorul care-l execută pe P_i este reafectat la unul din procesele P_j aflat în starea de *aşteptare pentru reluare* (*waiting for run*).

Abordarea acestei metode într-un context multiprocesor ridică următoarea problemă. Starea procesului suspendat P_i, conținută în registrele procesorului, va trebui ea însăşi sa fie salvată şi starea procesului P_j va trebui să fie încărcată în registrele acestui procesor în scopul efectuării unui comutator de context.

Dacă starea procesului este salvată în memorie, atunci însăşi această procedură va implica acces la memorie ceea ce va agrava problema latenței. Se poate evita această situație dacă se prevăd seturi multiple de registre de procesor care să mențină stări multiple de proces. Evident creşte complexitatea sistemului.

Pentru ca această metodă să fie rentabilă este necesar ca tot timpul să existe un număr suficient de procese concurente disponibile: *Ori de câte ori un proces efectuează o solicitare de acces la memorie trebuie să existe un set nevid de procese ce aşteaptă să fie executate.*

2.3.2 Utilizarea arhitecturilor de tip LOAD/STORE

In literatură [Dasg 89] se citează cu titlu de exemplu o metodă de reducere a latenței utilizată la calculatorul CRAY-1, constând din utilizarea instrucțiilor de tip LOAD/STORE: *Singurele instrucții ce fac referire la memorie sunt instrucții de tip LOAD/STORE; toate celelalte instrucții ce se referă la date utilizează registrele procesorului.*

Această observație se aplică şi maşinilor RISC.

De fapt, utilizarea unui set larg de registre programabile (sau a unui sistem local de înmagazinare a informației) sugerează noțiunea de memorie cache, singura excepție fiind aceea că registrele sunt programabile. Rezumând, problema se pune ca fiecare procesor să aibă memoria sa locală şi orice modificare de date, copie a unei locații din memoria principală, să fie actualizată în orice altă memorie locală unde ea se află.

2.3.3 Coerența memoriilor cache (CCH)

În SMP-MP o metodă de a reduce *latența* memoriei partajate şi - *în extenso* - a întregului sistem constă în utilizarea memoriilor cache - C - asociate fiecărui procesor - amplasate într-un sistem ierarhizat.

Copii multiple ale aceluiaşi bloc de memorie din MP pot fi rezidente în mai multe memorii cache private. Modificarea de către un procesor a oricărei copii a acestui bloc partajat în memoria sa cache privată, va schimba aceste date partajate într-o valoare absolută în memoria principală sau în oricare alta C.

Un sistem de memorie este *coerent* dacă valoarea returnată în urma execuției unei instrucții LOAD este întotdeauna identică cu valoarea scrisă în aceeaşi adresă de către ultima instrucție STORE. [CF78].

Sursele inconsistențelor datelor pot fi mai multe.

În fig. 2.6 se prezintă schematic structura unui sistem multicache, având n memorii cache C_i pentru $i = 1, ..., n$, şi o memorie principală partajată de toate procesoarele P_i.

Fie:

X - adresă fizică a unui bloc din memoria principală

y_i - adresa acestui bloc în C_i

Problema coerenței apare atunci când:

1. Un bloc X este memorat în C_i la adresa y_i şi în C_j la y_j. O inconsistență între y_i şi y_j poate apare deoarece:
   - Un proces P ce migrează de la procesorul P_i spre P_j, modifică blocul y_i într-un fel şi blocul y_j în alt fel;
   - Un proces ce rulează în P_i modifică y_i, şi alt proces din P_j modifică diferit y_j.
2. Un bloc X nu este apt să se sincronizeze în dinamica modificărilor cu memoriile C. De exemplu, o copie a lui X este în y_i, dar când y_i este modificat, X nu este actualizat. Dacă blocul X este solicitat în P_j, atunci ca şi un rezultat al erorii, X este copiat în C_j, dar acesta nu este ultimul X.

Diferite variante de configurare a SMP-MP cu memorii cache sunt prezentate în fig. 2.6.a, 2.6.b şi 2.6.c.

Cel mai popular protocol pentru a menține CCH, se numeşte *snooping*. În fig. 2.7 se prezintă un SMP-MP ce utilizează o singură magistrală şi în care se evidențiază modul de acces al memoriilor cache la memoria principală.

Toate dispozitivele de comandă ale memoriilor cache (DCC) monitorizează magistralele pentru a determina dacă în C există copii ale blocului protejat.

Acestă monitorizare – *snooping* - este utilizată pentru SMP-MP cu o singură magistrală, arhitecturi tipice pentru anii '80. Maşinile din acea perioadă au evoluat de la uniprocesoare prin adăugarea de procesoare multiple la o singură magistrală. Apoi s-au adăugat memoriile cache pentru ameliorarea performanțelor şi apoi s-au proiectat DCC ce să monitorizeze informația pe magistrala partajată.

Menținerea CCH are două componente: *citiri* şi *scrieri*. Copiile multiple nu constituie problemă pentru citire, dar un procesor trebuie să aibă un acces exclusiv atunci când scrie un cuvânt. Procesoarele trebuie să conțină cea mai recentă copie la citirea unui obiect, astfel că toate procesoarele trebuie să obțină noile valori după o scriere. Prin urmare protocoalele de monitorizare trebuie să localizeze toate C care protejează un obiect ce trebuie scris. Consecința unei scrieri a unei date protejate este fie *invalidarea* tuturor celorlalte copii, fie *actualizarea* copiilor partajate cu valoarea ce a fost scrisă.

Biții de stare dintr-un bloc cache sunt utilizați pentru *protocoale de monitorizare - snooping-* şi informația este utilizată pentru monitorizarea activităților magistralei. La un eşec în citire toate C verifică să vadă dacă ele au o copie a blocului solicitat şi apoi efectuează acțiuni specifice, ca de exemplu furnizarea datelor către C care a eşuat la citire. Similar, la o scriere, toate C verifică existența copiei la fiecare din ele şi apoi acționează, fie invalidând, fie actualizând copia lor cu noua valoare.

Monitorizarea interferează rar cu accesele procesorului datorită existenței unui port suplimentar ataşat fiecărui P - prin care se investighează porțiunea de etichetă referitoare la stare.

Protocoalele de monitorizare sunt:

- *invalidare de scriere /(write-invalidate)*: Procesorul care efectuează o operație de scriere provoacă invalidarea tuturor copiilor în alte C înainte de schimbarea copiei sale locale; apoi se actualizează data locală până când alt procesor o solicită. Procesorul care scrie trimite un semnal de invalidare pe magistrală, şi toate C verifică

existența propriilor copii; dacă este aşa, ele trebuie să invalideze blocul conținând cuvântul. Astfel, această schemă permite existența mai multor procesoare ce citesc, dar doar a unuia ce scrie;

- *actualizare de scriere / (write-update)* : Procesorul care scrie transmite noua dată prin magistrală; Toate copiile sunt apoi actualizate cu noua valoare. Această schemă cunoscută sŭb numele de *difuzare de scriere*, transmite continuu comenzi de scriere spre data partajată în timp ce procedura *invalidare de scriere*, şterge toate celelalte copii, astfel că există doar o copie locală pentru scrieri ulterioare.

Protocolul *actualizare de scriere* este similar cu protocolul *write – through* deoarece toate scrierile parcurg magistrala pentru a actualiza copiile datelor partajate. Protocolul *invalidare de scriere* utilizează magistrala doar la prima scriere pentru a invalida celelalte copii, şi prin urmare scrierile următoare nu vor afecta traficul magistralei. În consecință protocolul *invalidare de scriere* are avantajul de a reduce accesele la magistrală, deci o reducere a traficului, în timp ce *actualizare de scriere* (*scrie – actualizează*) are avantajul de a face ca noile valori să apară în memoriile C, prin urmare se poate reduce latența.

SMP–MP comerciale bazate pe magistrală unică partajată utilizează protocolul *invalidare de scriere* ca şi protocol standard de asigurare a CCH.

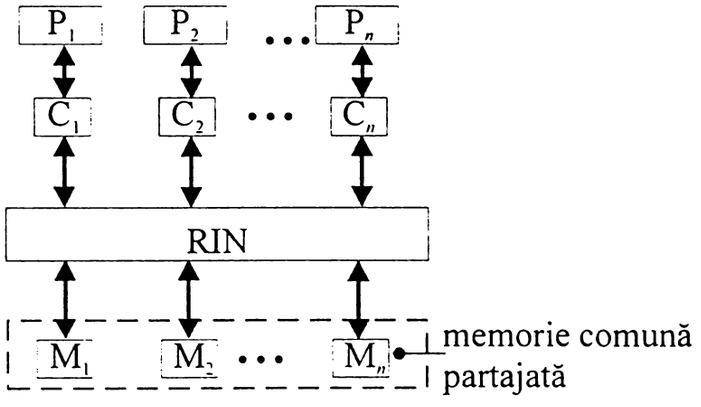


Fig. 2.6.a SMP - MP cu MCH private

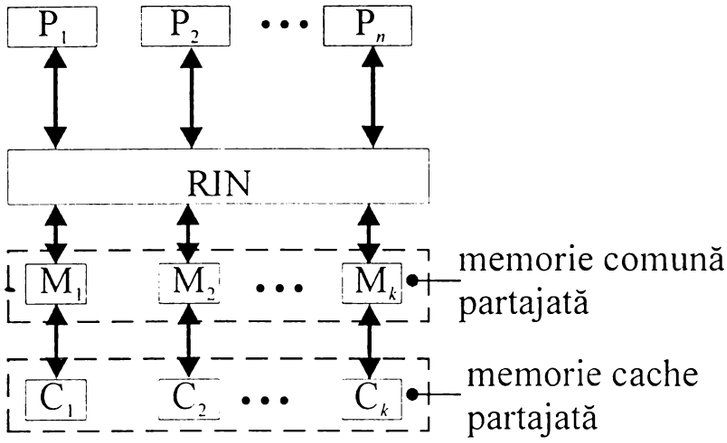


Fig. 2.6.b SMP - MP cu memorii cache partajate

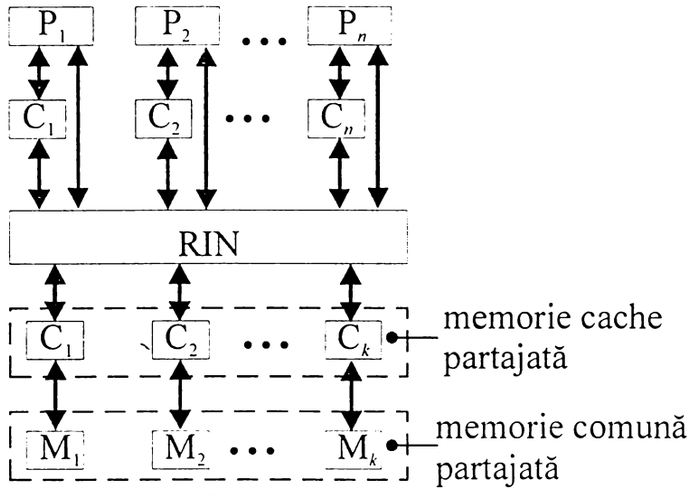
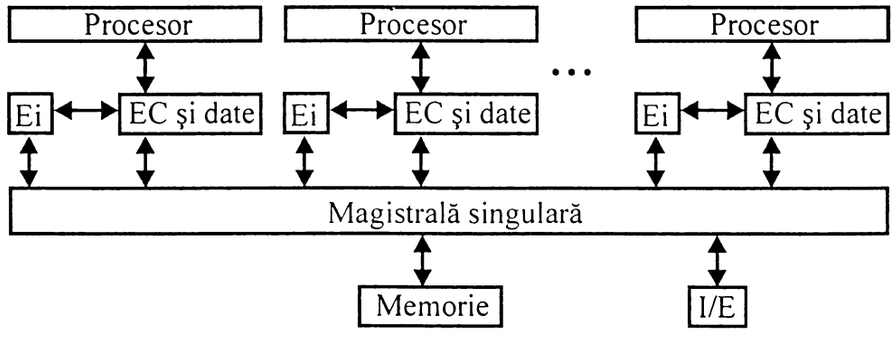


Fig. 2.6.c SMP - MP având memorii cache private şi partajate



Ei - etichetă pentru înregistrare
EC - etichetă pentru cache-are

Fig. 2.7 Structura unui SMP-MP cu magistrală unică ce utilizează tehnica *snooping* pentru asigurarea coerenței

Un exemplu de protocol *invalidare de scriere*

În fig. 2.8 se prezintă o diagramă de stări finite pentru un protocol *invalidare de scriere* bazat pe o politică de "*rescriere*". Fiecare bloc cache este în unul din cele trei stări:

1. *citeşte numai*: Acest bloc cache este "curat" (nemodificat) / (clean) şi poate fi partajat de alte procesoare;
2. *citeşte / scrie*: Acest bloc este "viciat" (modificat) / (dirty) şi nu este partajat;
3. *invalid*: Acest bloc nu are date valide.

Cele trei stări sunt duplicate în figură, pentru a arăta tranziţii bazate pe acţiuni ale procesorului în opoziţie cu tranziţii bazate pe operaţiuni de magistrală.

În realitate există doar un singur automat cu stări finite pentru fiecare C_i, cu stimuli provenind fie de la P_i, fie de pe magistrală.

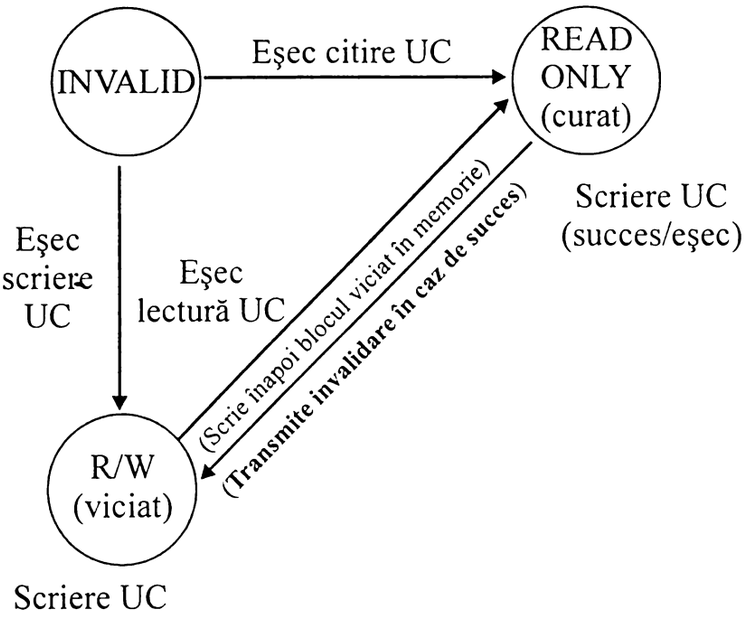
Tranziţii de stare ale unui C_i apar la eşecul în citire, la eşecul în scriere sau în scrieri reuşite. Succesele la citire nu schimbă starea lui C_i.

Să analizăm ce se întâmplă în cazul unui eşec la citire.

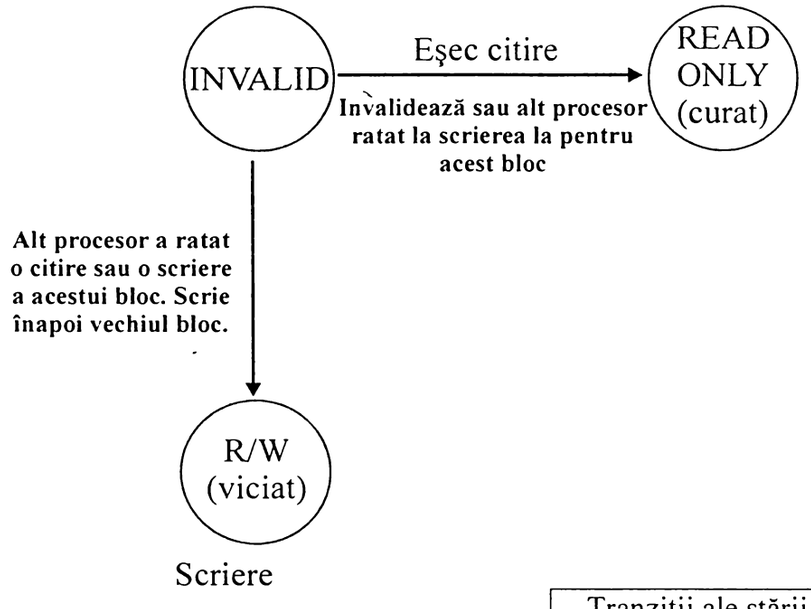
Cînd P_i are un eşec la citire, el va schimba starea blocului din C_i în Citeşte Numai şi va rescrie vechiul bloc dacă acesta a fost în starea Citeşte/Scrie (viciat). Toate celelalte memorii cache din restul procesoarelor monitorizează eşecul la citire pentru a observa dacă acest bloc este în memoriile C private. Dacă unul are o copie şi este în starea Citeşte/Scrie, atunci blocul este scris în M şi, în acest protocol, blocul este poziţionat în starea Invalid. Eşecul în citire este apoi satisfăcut citind din memorie.

Pentru a scrie un bloc, procesorul câştigă accesul la magistrală, trimite un semnal de invalidare, scrie în acest bloc şi îl plasează în starea Citeşte / Scrie. Datorită faptului că alte memorii cache monitorizează magistrala, toate C verifică să vadă dacă au copii ale acestui bloc; dacă da, atunci îl invalidează.

Evident, după cum se observă, există multe variante de scheme de asigurare a CCH. Ele vor fi discutate ulterior.



Tranziţii ale stării lui C_i utilizând semnale de la P_i



Tranziţii ale stării lui C_i utilizând semnale provenind din magistrală

Observaţii:

1. Săgeţile simple şi textele nesubliniate specifică acţiuni ce se produc în memorii C fără coerenţă.
2. Săgeţile şi textele îngroşate reprezintă acţiuni adăugate pentru asigurarea CCH.

Fig. 2.8. Diagrama de stări pentru un protocol *invalidare de scriere*

Scheme de asigurare a coerenței la memoriile cache (SAC- MCH)

Pentru a păstra coerența memoriei cache este necesar să ne asigurăm că toate copiile păstrate în C_i ale unei locații din M sunt identice.

Protocoale realizate hardware sunt utilizate în arhitecturile cu magistrale partajate. În general acestea utilizează un mecanism de transmitere asigurat de magistrala partajată pentru a menține coerența memoriilor cache multiple (MCHM).

În aceste protocoale, fiecare operație de scriere într-o locație din M - partajată între procesoare - este transmisă prin magistrală pentru celelalte procesoare ca să *invalideze* sau să *actualizeze* propriile lor copii ale locației, dacă ele sunt prezente.

Dacă într-o memorie ierarhizată dintr-un SMP - MP coexistă copii multiple ale aceleiaşi date, accesele la aceste copii trebuie să fie manipulate şi/sau coordonate atent pentru a preveni posibilitatea accesării unei copii pierdute.

Reamintim că un sistem este *coerent*, dacă valoarea returnată la o instrucție de tip LOAD este întotdeauna identică cu valoarea scrisă la aceeaşi adresă de către ultima instrucție STORE.

În fig. 2.9 se prezintă o taxonomie a diferitelor scheme de asigurare a coerenței (SAC-MCH).

În literatură [SLM89], [CBZ95], [Wils87] se inventariază câteva din scheme propunându-se soluții de extindere la SMP - MP de talie mare.

Aceasta taxonomie se bazează pe ideea că dacă întărirea coerenței nu este importantă, atunci este indiferent dacă invalidările sunt efectuate de un procesor local (de ex. auto-invalidare) sau de către alte procesoare (invalidări induse).

Schemele sunt în continuare clasificate în funcție de solicitarea de-a avea sau nu resurse partajate şi, dacă este aşa, ce tip de resurse trebuie alocate pentru mărirea coerenței.

A. SAC- MCH bazate pe ne-invalidare

În literatură [SLM 89] este indicată metoda *difuzare de scriere / write-broadcast* în care MCH private sunt menținute coerente prin forțarea dispozitivelor de comandă ale MCH să informeze alte MCH din sistem asupra fiecărei scrieri de-o astfel de manieră încât fiecare MCH să poată actualiza propria sa copie, dacă o astfel de copie există.

Cea mai simplă schemă dintre cele bazate pe metoda *difuzare de scriere*, este *difuzare de scriere cu scriere prin tot spațiul. / write-broadcast with write-through.*

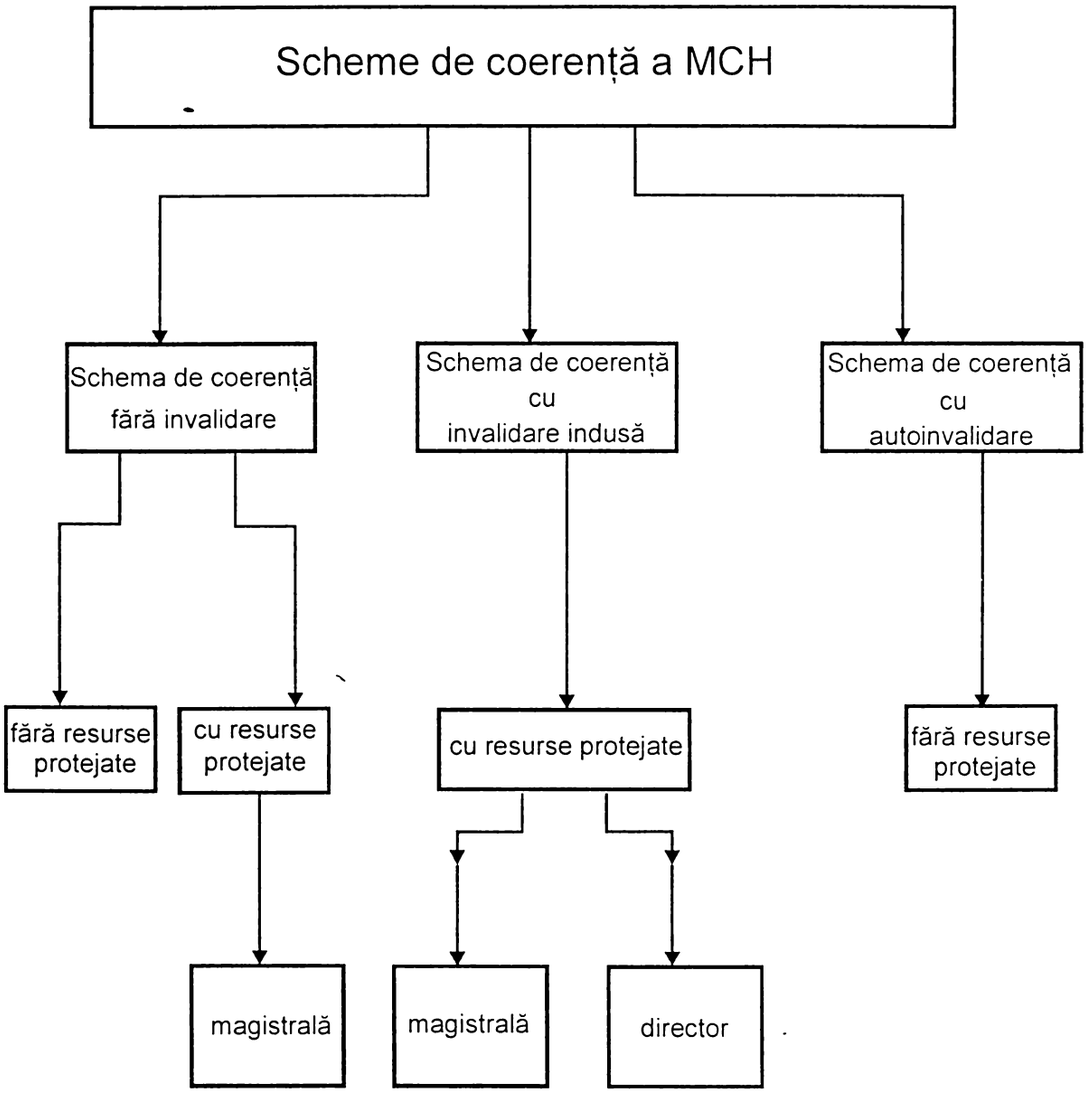


Fig. 2.9 Taxonomia schemelor de asigurare a coerenței memorilor cache

În această schemă, toate scrierile sunt imediat reflectate în MP - memoria globală partajată. Fiecare MCH din sistem urmăreşte aceste tranzacții *scrie-în-memorie* şi îşi actualizează propria sa copie. Această procedură de urmărire implică existența unui mediu cu magistrală partajată.

Este de notat faptul că traficul de scrieri va satura curând magistrala de memorie chiar şi în condiţiile existenţei unui număr redus de procesoare, ceea ce limitează această abordare.

Pentru a reduce utilizarea magistralei de memorie, multe SAC-MCH bazate pe tehnica *difuzare de scriere* sunt proiectate pentru a diminua numărul de cicluri difuzare de scriere. De exemplu, operaţii *difuzare de scriere* sunt efectuate doar atunci când blocul ce este scris se regăseşte şi în alte MCH.

Chiar dacă schema *difuzare de scriere* a fost utilizată doar în SMP-MP bazate pe magistrală, ea poate fi utilizată la orice SMP-MP. Protocolul va fi *bazat pe director / directory-based* cu dispozitivele de comandă ale memoriei având responsabilitatea aducerii la zi a copiilor MCH în cadrul procesului de scriere.

După recepţionarea unei cereri de scriere de la un procesor, controler-ul de memorie va trimite cereri de actualizare a intrării cache corespunzătoare către MCH ce au o astfel de intrare.

Acest lucru se întâmplă după recepţionarea semnalelor de confirmare a transferurilor de la toate MCH spre care s-au transmis semnale de actualizare pe care controler-ul de memorie le trimite ca şi confirmare pentru cererea de scriere a procesorului solicitat. Un astfel de protocol este ineficient şi nu va mai fi luat în considerare.

B. SAC-MCH bazate pe invalidare indusă.

În schemele de asigurare a coerenţei bazate pe *invalidare - indusă/induce invalidation* intrările în MCH private pot fi invalidate prin mesaje generate de alte procesoare. În această clasă de scheme de asigurare a coerenţei, resursele partajate sunt uzual solicitate pentru a decide dacă este nevoie de invalidare, pentru a decide pe cine să invalideze şi/sau să actualizeze invalidările. În cele ce urmează se vor analiza schemele de asigurare a coerenţei MCH din această clasă în concordanţă cu tipul de resursă partajată pe care acestea le solicită.

B.1 Scheme bazate pe magistrală

Într-un SMP cu magistrală partajată cu o schemă de asigurare a coerenţei bazată pe *invalidare indusă*, tranzacţii de scriere sunt transmise pe magistrală. Fiecare controler al

MCH scanează magistrala şi invalidează o intrare de mixare în MCH, dacă o astfel de intrare există. Magistrala este de asemenea utilizată să alimenteze cea mai actuală dată fie din memorie fie din MCH privată la un eşec în citire.

În această clasă de scheme, o stare este uzual asociată cu fiecare bloc cache în fiecare MCH.

În mod uzual starea indică când un bloc este valid, partajat sau exclusiv, şi/sau dacă memoria este de actualizat în raport cu blocul cache corespunzător ("nemodificat") sau nu ("modificat").

De exemplu, starea locală în schemele de asigurare a coerenței de tip *write-once* poate fi una din următoarele : (1) *invalidă*, (2) *nemodificat-partajată*, (3) *nemodificat -exclusivă*, (4) *nemodificat-exclusivă.*

Toate controler-ele MCH în sistem lucrează împreună, fiecare cu fiecare şi cu memoria pentru a asigura un sistem de memorie coerent.

Cele mai multe dintre aceste scheme permit ca multiple copii ale unui bloc " nemodificat " să fie conținute în MCH private, dar cel mult un MCH privat poate conține o copie "modificată ".

B.2 Scheme bazate pe director

Într-o schemă de asigurare a coerenței MCH cu *invalidare indusă bazată pe director*, (*directory-based*) câte o intrare *director* (*index*), ce este memorată în controlerul memoriei, este asociată fiecărui bloc de memorie. Această intrare codifică starea blocului. Starea este utilizată pentru a decide dacă există o solicitare pentru invalidări în cadrul unei tranzacții date de scriere în bloc şi, dacă este aşa, să localizeze MCH privată care are o copie a blocului ce trebuie invalidată.

Este de asemenea utilizată pentru a indica dacă blocul de memorie corespunzător este perimat (stale) sau nu, şi dacă este aşa, să localizeze MCH privată care este garantată să aibă cea mai recentă copie a blocului. În plus față de stare din directorul global, în mod uzual se asociază o stare locală cu fiecare MCH privat.

Această stare locală este utilizată să permită unei MCH private să proceseze cele mai multe cereri de la procesorul asociat fără implicarea altor acțiuni globale.

În [CF 78], Censier şi Feautrier au propus o SAC- MCH cu invalidare indusă, bazată pe director. În schema lor, fiecare intrare director constă dintr-*un bit de semnalizare a stării* "modificat " (dirty) şi o *etichetă de n-bit*, unde *n* este numărul MCH private din sistem. Al *i*-lea bit din eticheta unui bloc de memorie este poziționat dacă MCH privată corespunzătoare are o copie validă a blocului. Bitul de semnalizare indică dacă blocul de memorie este actualizat sau nu.

Starea locală poate fi una din următoarele trei stări :

(1) *invalidă* (*invalid*);

(2) *nemodificat-partajat* (*clean-shared*);

(3) *modificat -exclusiv* (*dirty-exclusive*).

La o citire reuşită, data solicitată este imediat furnizată procesorului solicitant şi nici o acțiune adițională nu este solicitată.

- Dacă există o scriere reuşită şi MCH are o copie *modificat-exclusiv* a blocului, scrierea poate să continue local fără să implice nici o ac[illegible]e globală. Dar dacă există o scriere reuşită şi MCH are o copie *nemodificat -partajat* a blocului, MCH va transmite o cerere către controlerul MCH pentru a invalida copii ale blocului prezente în alte MCH private.

La un eşec la citire, bitul de semnalizare şi eticheta de prezență, asociată cu blocul de memorie solicitat sunt examinate pentru a vedea care MCH privată are copii valide ale blocului. Dacă o oarecare MCH privată are o copie *viciat-exclusiv* a blocului solicitat, blocul este rescris în memoriea globală şi utilizat să furnizeze date spre MCH solicitant. Biții "viciați" ai blocului sunt puşi pe 0 în directorul general şi în starea locală a MCH ce are o copie "viciată". Stările locale în aceea MCH şi în cea solicitantă devin *nemodificat-partajate*. În alte cazuri (de ex., când nici una sau mai multe MCH au copii *nemodificat-partajate* ale blocului), blocul solicitat este furnizat de memoria globală şi este încărcat în MCH ca şi *nemodificat-partajat*.

La un eşec la scriere se testează din nou bitul de semnalizare şi eticheta de prezență pentru a se vedea dacă una din MCH private are o copie validă a blocului solicitat.

Există trei cazuri ce trebuie analizate.

Dacă nici una din ele nu are o copie validă, blocul este furnizat de memoria globală. Dacă cel puțin o MCH privată are o copie *nemodificat-partajată* a blocului, toate copiile blocului sunt invalide şi blocul solicitat este furnizat de memoria globală.

Dacă o MCH privată oarecare o copie *modificat-exclusivă* a blocului, copia modificată este rescrisă în memoria globală înainte ca ea să fie invalidată. Noul bloc scris este utilizat pentru a alimenta date spre MCH solicitant.

În toate cele trei cazuri menționate anterior, bitul de semnalizare global al stării modificate aparținând blocului solicitat este poziționat şi blocul solicitat este încărcat în MCH solicitant ca şi *modificat-exclusiv*.

Pentru a limita cantitatea informației globale de stare conținută în director, Archibald şi Baer [SLM 89] propun o SAC- MCH bazată pe director denumită *schema cu director format din doi biți*.

În această schemă, informația directorului global constă din doi biți ce codifică una din cele patru stări posibile :

(1) Blocul nu este prezent în nici o MCH;

(2) Blocul este într-un MCH precis şi nu este modificat;

(3) Blocul este prezent în nici - unul sau în mai multe MCH şi nu este modificat;

(4) Blocul este prezent exact într-un MCH şi este modificat ("viciat").

Deoarece directorul nu mai conține identități ale MCH conținând un bloc dat în schema propusă de autori, sunt necesare transmisii pentru a efectua invalidări şi pentru a solicita o rescriere a blocului modificat.

C. SAC-MCH bazate pe auto-invalidare

Chiar dacă SAC- MCH bazate pe invalidări induse pot fi eficiente în asigurarea unei rate înalte de succes datorită abilității lor de-a urmării dinamic traseul stării fiecărui bloc, traficul rețelei generate pentru solicitări de invalidare şi pentru manipularea informației de stare locală/globală poate fi substanțial.

În SAC- MCH cu *auto-invalidare*, intrările în MCH pot fi invalidate doar de procesoare locale şi nici o informație manipulată global nu este asociată cu blocuri MCH sau cu blocuri din memoria globală. Aceasta elimină traficul suplimentar în detrimentul unei gestionări mai puțin eficiente a MCH. În a decide ce şi când să invalidezi pentru a

menține concret coerența, SAC- MCH cu *auto-invalidare* utilizează câteva informații din cadrul structurii programului paralel obținute în timpul compilării. Aceste scheme sunt uneori menționate ca fiind scheme asistate software datorită dependenței lor de compilator.

În literatură [SLM 89] se propune o *schemă cu invalidare selectivă rapidă.*

Fiecare referire la o operație de citire este marcată în cadrul compilării ca fiind fie *citire din memorie* (MCH poate să conțină o copie perimată a datei de citit) sau citire din MCH (la o apelare reuşită, se garantează că intrarea MCH este actualizată).

În fig. 2.10 se prezintă două condiții posibile pentru un acces perimat.

În prima condiție, o intrare cache este încărcată printr-o scriere de către P_i şi devine perimată când alt P_j transmite o scriere către aceeaşi locație de memorie. Dacă lui P_i îi este permis să acceseze copia perimată în propria MCH, el ar viola criteriul de corectitudine a coerenței MCH. În mod similar, un acces perimat este posibil când intrarea cache este încărcată printr-o operațiune eşuată de citire şi devine perimată printr-o scriere provenind de la alt procesor. De notat că aceste condiții pretind ca fiecare acces la o variabilă partajată să fie marcată ca *citire din memorie* odată ce variabila a fost scrisă de două ori de către diferite procesoare.

Suplimentar cu schema de date pentru accese la date perimate, *un bit de schimb* se asociază cu fiecare cuvânt. Scopul lui principal este de-a permite reținerea referințelor *citeşte din memorie* ce apar mai mult de o singură dată la un moment dat într-un ciclu. Biții de schimb a MCH sunt poziționați la începutul fiecărui ciclu.

O citire din cache (*cache-read*) este executată ca de obicei, indiferent de starea bitului de schimb. La o *citire din memorie* (*memory-read*) dacă există un eşec sau un acces reuşit cu starea bitului de schimb modificată, atunci cuvântul este încărcat în cache şi se inițializează bitul de schimb. Această situație implică ca fiecare prin acces la o variabilă partajată după începutul unui ciclu să fie servit de memoria globală odată ce variabila a fost scrisă de două ori de către procesoare diferite. Aceasta limitează scalabilitatea schemei. La o *citire din memorie*, când a avut loc un acces reuşit şi bitul de schimb asociat a fost inițializat, solicitarea de acces este satisfăcută de către cache. În plus, fiecare scriere reuşită inițializează biții de schimb asociați şi actualizează memoriile principale şi cache. În [SLM 89] se prezintă o schemă de asigurare a coerenței cu auto-

invalidare care permite operarea memoriilor cache în interiorul secțiunilor critice. În această schemă, operarea memoriilor cache în interiorul secțiunilor critice a fost posibilă prin utilizarea identificatorilor de tipul *one time identifier* (OTI) ce sunt garantați să fie unici în timpul existenței lor. Identificatorul OTI este utilizat pentru a se distinge intrările de actualizare ale memoriilor cache (încărcate în cache în secțiunea critică curentă) de acelea perimate (încărcate în cache în timpul secțiunilor critice trecute).

În acest scop, fiecare cache are un câmp OTI şi un OTI distinct este folosit pentru fiecare execuție de secțiuni critice. La fiecare acces cache, OTI curent este comparat cu câmpul OTI în blocul cache corespunzător.Dacă câmpurile se potrivesc, solicitarea este satisfăcută de către memoria cache. Astfel, blocul este încărcat din memoria globală şi câmpul OTI este poziționat ca fiind OTI curent. În tabelul din fig.2.11, citat în [SLM89], se prezintă un sumar a SAC - MCH cu *auto-invalidare* discutate în acest paragraf.

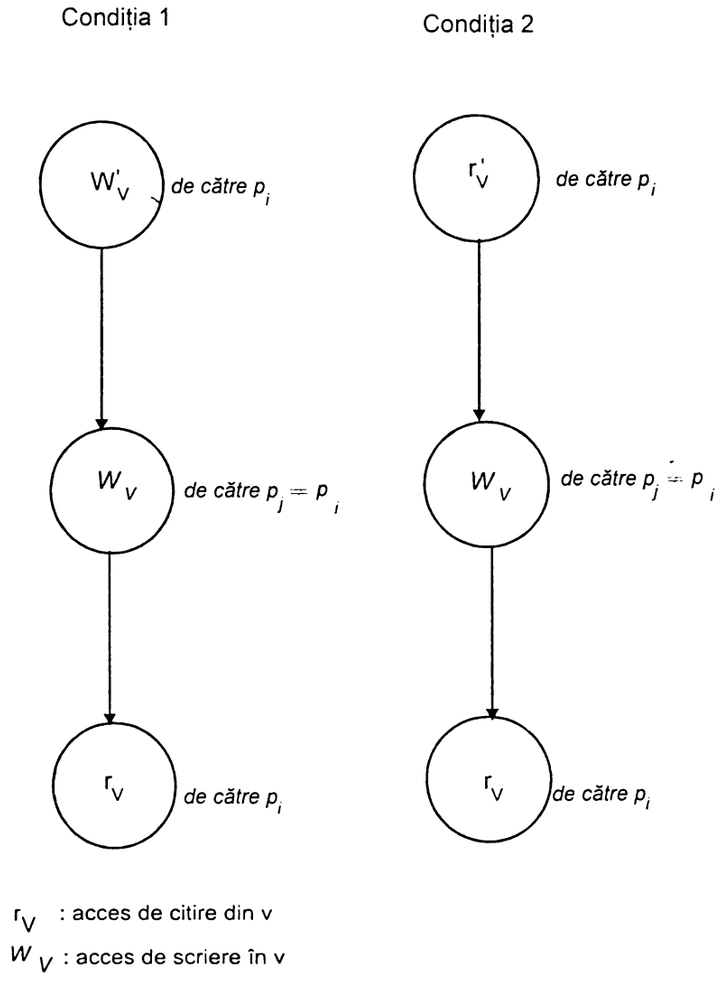


Fig. 2.10 Condiții pentru un acces la un bloc cache perimat

	Veidenbaum	McAuliffe	Lee	Veidenbaum Cheong	Cytron	Smith
unitate de întărire a coerenței	toate variabilele	fiecare variabilă	fiecare variabilă	fiecare referință	fiecare referință	fiecare pagină
regiune de întărire a coerenței	buclă	unitate de calcul (UC)	secțiune	program	program	secțiune critică
Detecția incoerenței	structură cu buclă încuibată	utilizare de variabile în UC	utilizare de variabilă într-o secțiune	analiză de flux	analiză de dependență de date	OTI
loc de invalidare	frontiere ale buclelor	frontiere ale UC-urilor	frontiere ale secțiunilor	frontieră de bucle	fiecare referință	sfârşit al secțiunii critice
politică de actualizare a memoriei globale	scrie prin	rescrie	rescrie	scrie prin	schemă hibridă	scrie prin

Fig. 2.11 Sumar al SAC-MCH cu *autoinvalidare*

2.4 REȚELE DE INTERCONECTARE

2.4.1 Terminologie

O rețea de interconectare RIN reprezintă o conexiune complexă de comutatoare şi legături ce permite comunicarea de date între procesoare şi memorii în SMP-MP - multiprocesoare - şi între elemente de procesare în SMP -TM - multicalculatoare.

Reamintim că în subclasa SMP-MP toate procesoarele P partajează o memorie comună în timp ce la SMP-TM fiecare procesor are memoria sa proprie cât şi alte resurse de I/E. De aceea vom utiliza denumirea de *element de procesare* -PE.

Există o relație directă între arhitectura unui sistem MIMD şi topologia RIN utilizată.

SMP-MP utilizează RIN de tip *crossbar*, *magistrală multiplă*, *multinivel* etc. în timp ce SMP-TM utilizează topologii ca şi : *stea*, *inel*, *arbore*, *hipercub*.

Compararea, clasificarea şi ordonarea RIN se fac în scopul evaluării performanțelor şi a stabilirii tipurilor de aplicații. Se utilizează mai multe criterii.

În literatură [BYA 89], [Prad 96], [Popa 96] etc. se admite că trebuiesc luate în considerare :

- filozofia temporizării;
- metodologia de comutare;

- strategia de comandă.

De aici rezultă principalele criterii de analiză :

- *modul de operare;*
- *tehnica de comutare;*
- *strategia de comandă;*
- *topologia;*
- *gradul de interconectare;*
- *complexitatea;*
- *gradul de blocare;*
- *capacitatea de reconfigurare.*

Modul de operare se referă la natura comunicării între moduri : *sincronă* şi *asincronă.*

Modul de operare sincron presupune existența unui ceas central care fixează paşii în care se produce transferul de date.

Varianta asincronă asigură comunicăriile utilizând protocoale hand-shaking cu interblocare. Se asigură o bună expandabilitate şi modularitate dar cresc dificultățile de proiectare.

Tehnica de comutare stabileşte modul de direcționare a informației. Există două tehnici utilizate: *comutare de circuite* şi *comutare de pachete.*

Comutarea de circuite presupune ca înainte de începerea transferului comutatoarele să fie astfel poziționate încât să stabilească o cale de comunicare între transmițător şi receptor. Aceasta rămâne nemodificată pe toată durata efectuarii transferului. Datorită existenței unui timp de stabilire a conexiunii cu implicare directă asupra latenței RIN, metoda este potrivită pentru blocuri lungi. Ele pot fi de dimensiuni variabile.

Comutarea de mesaje, pachete de date, presupune secționarea blocului ce trebuie transmis în entități de lungime fixă - pachete - care se transferă prin RIN după procedura "*memorează şi transmite (store - and forward)*".

Un comutator care recepționează pachetul îl memorează şi caută să stabilească o conexiune spre nivelul următor. Dacă nu găseşte o cale liberă şi un comutator liber, reține pachetul şi aşteaptă. În caz contrar, existând o cale liberă, pachetul este transmis la nivelul următor. Tehnica este utilizată pentru mesaje scurte.

Strategia de comandă defineşte modul prin care semnalele de comandă direcționează fluxul de date generat în RIN şi stabileşte modul de comandă a comutatoarelor.

Informația de comandă, care poziționează comutatoarele, poate fi generată de un controler central sau poate fi conținută în mesajul ce este transferat prin RIN.

În consecință controlul poate fi *centralizat* sau *distribuit*. Controlul centralizat, datorită existenței controlerului central, introduce întârzieri în stabilirea căii de comunicație pentru că elementul de control trebuie să cunoască toate cererile procesoarelor şi să rezolve un eventual conflict. De asemenea introduce o strangulare a sistemului (system bottleneck) cu efecte negative directe asupra dependabilității sistemului.

Controlul distribuit descentralizează decizia de comutare la nivelul comutatoarelor în sensul că se atribuie un controler simplificat fiecărei componente a sistemului.

În aplicații ale multiprocesoarelor, comanda unui RIN *crossbar* este uzual centralizată iar comanda unui RIN-MN (Multiple Level Interconection Network) este descentralizată.

Bazat pe caracteristicile operaționale prezentate mai sus, RIN pot fi clasificate în opt categorii diferite pentru o topologie dată.

Schema de clasificare este prezentată în fig. 2.12

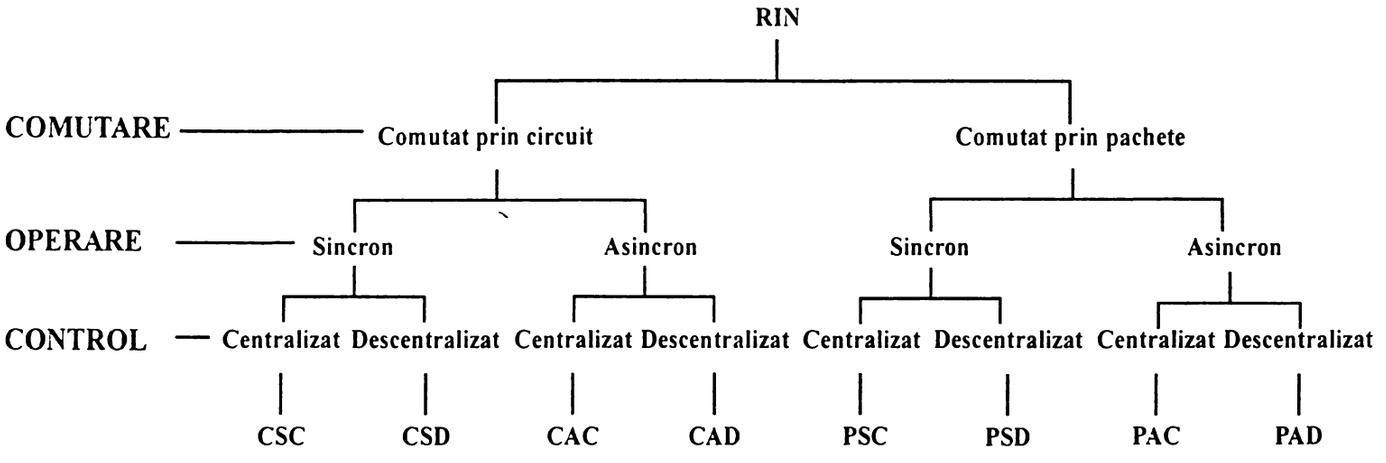


Fig. 2.12 Clasificare a RIN

Topologia definește forma unui RIN. Aceasta poate fi decrisă printr-un graf în care *nodurile* sunt comutatoarele iar *arcele* reprezintă legăturile dintre ele. De fapt trebuie notat că un nod reprezintă un cuplu procesor-memorie-comutator. Comutatoarele apar însă ca noduri ale grafului.

Topologiile pot fi *regulate* şi *neregulate*. Din punct de vedere a facilităților de reconfigurare ele pot fi *statice* sau *dinamice*.

RIN statice (static interconnection network) sunt acelea care permit numai legături fixe (uni sau bidirecționale) între două noduri. Aceste RIN sunt tipice pentru multicalculatoare adică SMP – TM.

RIN statice se pot clasifica în funcție de criteriul spațial în topologii:

- pe o dimensiune;
- pe două dimensiuni;
- pe trei dimensiuni;
- pe mai mult de trei dimensiuni.

Exemplu de topologii RIN statice:

monodimensională	-	*magistrala comună*;
bidimensională	-	*inel*, *stea*, *arbore*, *plasă*, *sistolă*
tridimensională	-	*cub*
multidimensională	-	*hipercub*

În fig. 2.13 se prezintă structura unui nod ce este interconectat în RIN.

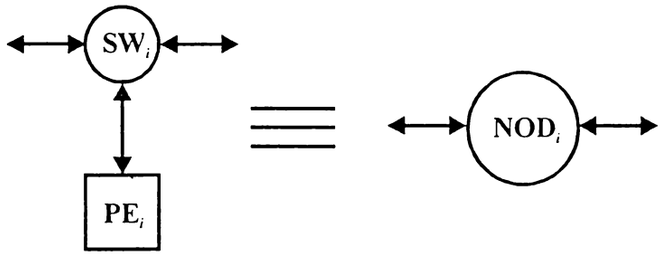
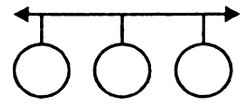


Fig 2.13 Structura unui nod de interconectare la RIN

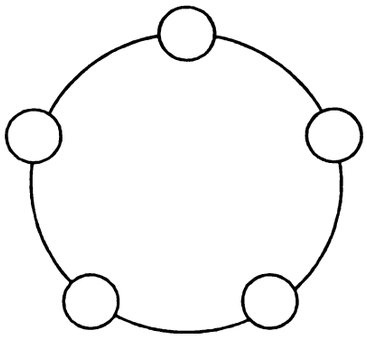
În fig. 2.14 se prezintă câteva exemple de RIN statice. Acestea sunt *magistrala comună*, *inel*, *stea*, *arbore*, *plasă*, *sistolică*, *cub* 3*D*, *hipercub*. Se remarcă diferențele în topologie între aceste tipuri de interconectări. La o extremitate se găseşte RIN – *magistrală comună* în care toate nodurile comunică între ele prin intermediul unei magistrale partajate în timp, iar la cealaltă se găsesc RIN de tip *cub*, respectiv *hipercub* în care fiecare nod este interconectat cu celelalte.

În cadrul sistemelor multiprocesor cu memorie partajată, cea mai simplă conexiune o reprezintă magistrala comună a cărei analiză se va face ulterior. Sistemele de calcul distribuite sunt caracterizate prin RIN *plasă* şi RIN *cub*, respectiv *hipercub*.

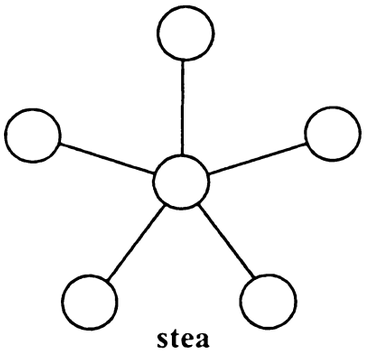
Rețelele cu arii locale – LAN – sunt conectate în general prin intermediul RIN-*stea* respectiv RIN-*inel* în timp ce RIN-*sistolică* face obiectul unei clase separate de maşini paralele.



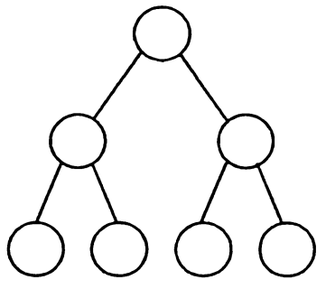
magistrală comună



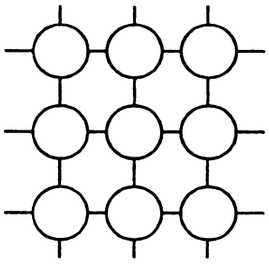
inel



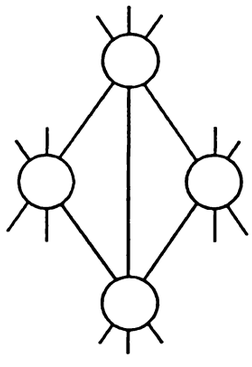
stea



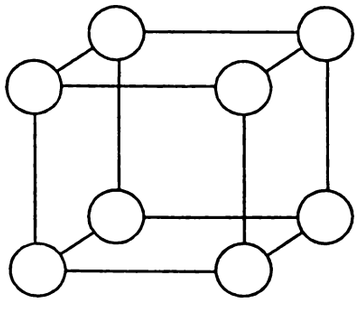
arbore



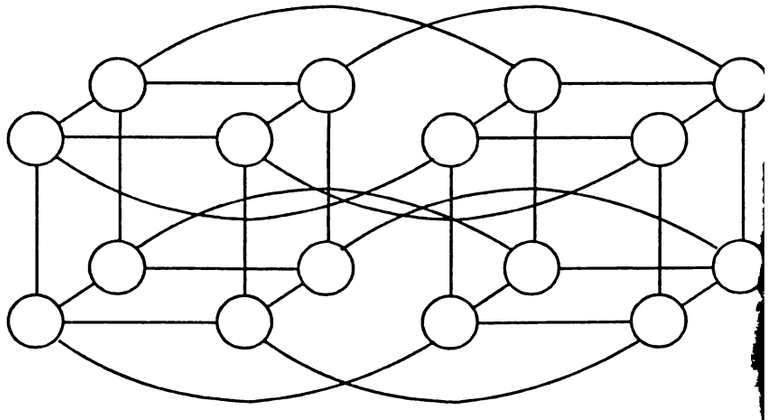
plasă



sistolică



cub 3D



hipercub 4D

Fig. 2.14 RIN statice. Exemple

Topologiile dinamice se caracterizează prin existența comutatoarelor active ce stabilesc calea de comunicație între două elemente de procesare.

RIN dinamice pot fi configurate în topologii pe:

- un nivel
- mai multe nivele
- tip *crossbar*.

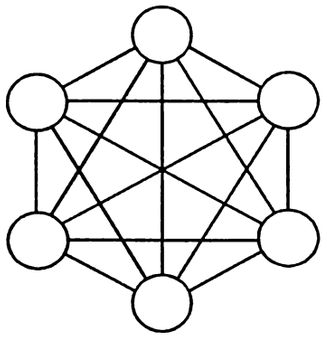
Topologiile pe un nivel având dispuse comutatoarele pe acesta, sunt cu recirculare în sensul că un mesaj ca să ajungă la destinație poate să parcurgă de mai multe ori RIN.

Topologiile organizate pe mai multe niveluri sunt formate din cupluri de *conexiuni de permutare* şi *etaje de comutare* dispuse pe niveluri succesive. Exemple sunt RIN de tip *banyan, delta, omega.*

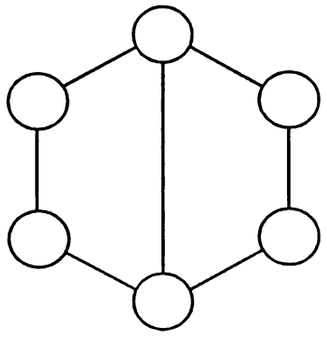
Topologia *crossbar* asigură căi de comunicație separate pentru fiecare pereche de parteneri. Este cea mai complexă RIN dar şi cea mai scumpă.

Gradul de interconectare împarte topologiile RIN în două: cu *interconectare totală* şi cu *internonectare parțială.*

În fig. 2.15 se prezintă exemple de topologii cu cele două tipuri de interconectare.



RIN cu interconectare totală



RIN cu interconectare parțială

Fig. 2.15 Exemple de interconectări

Complexitatea este un criteriu de analiză în strânsă corelare cu costul RIN.

Numărul de comutatoare poate fi determinat pe baza următoarelor tipuri de funcții:

- $f(N)$ - depinde liniar de numărul N al partenerilor;
- $f(N \times \log N)$ - depinde logaritmic de numărul partenerilor;
- $f(N^2)$ - dependență patratică.

RIN de tip *stea, inel, plasă, cub* şi cele *dinamice* pe un singur nivel au o complexitate dată de *f(n)*.

În a doua categorie intră RIN dinamice pe mai multe nivele. Baza logaritmului este dată de numărul intrărilor şi ieşirilor comutatorului. Pentru un comutator cu M intrări şi M ieşiri funcția va fi de tipul $f(N \times \log_M N)$.

În literatura de specialitate [Hwan 93] se consideră că aceste tipuri de RIN optimizează raportul performanță/cost.

RIN a căror complexitate şi cost a topologiei sunt determinate de $f(N^2)$ au performanțe maxime oferind conectivitate totală în timp minim. Datorită costului mare, devin nerentabile pentru un N mare.

2.4.2 Sisteme multiprocesor cu memorie partajată orientate pe magistrală

Cea mai simplă conexiune între procesoare şi alte unități funcționale în cadrul unui SMP – MP o reprezintă *magistrala partajată* (*shared-bus*).

Magistrala reprezintă o cale de comunicare unică la care se conectează unități funcționale care pot fi:

- procesoare;
- memorii;
- porturi de I/E.

Magistrala comună este o unitate pasivă neexistând comutatoare. Controlul transferului de date/instrucții se face de către interfețele existente la fiecare unitate conectată pe magistrală.

În fig. 2.16 se prezintă configurația unui SMP – MP orientat pe magistrală.

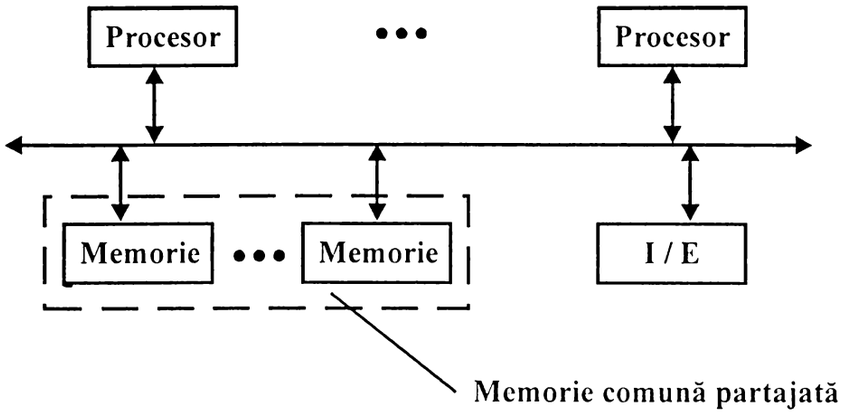


Fig. 2.16 SMP – MP orientat pe magistrală

Magistrala este *comună* pentru că la ea se conectează toate resursele hard ale sistemului şi este *partajată în timp* pentru că la un moment dat doar două unităţi funcţionale sunt logic conectate între ele efectuându-se un transfer de date.

Structura magistralei este construită în jurul conceptului *master-slave*, unde o unitate funcţională de tip *master* preia controlul magistralei şi a unităţii funcţionale de tip *slave*, după ce i-a decodificat adresa şi efectuează transferul. Această procedură - bazată pe un protocol de tip *hand-shake* – permite conectarea modulelor de diferite viteze prin intermediul magistralei. Ea este prin urmare *asincronă*.

De asemenea, o caracteristică importantă a magistralei este abilitatea sa de a permite conectarea mai multor unităţi funcţionale de tip master în vederea multiprocesării.

Un procesor se poate găsi în două ipostaze: *master* sau *slave*. O memorie nu poate fi decât *slave* iar o unitate funcţională de I/E, de obicei în postură de *slave*, poate fi *master* doar dacă are facilităţi DMA.

Dacă mai multe unităţi master solicită accesul la magistrală este necesară *arbitrarea* solicitarilor pentru a se evita situaţia de *conflict*.

Conflictul poate apare dacă:

- două sau mai multe unitaţi *master* doresc să comunice cu aceste unităţi *slave*;
- două sau mai multe unităţi *master* necesită aceeaşi cale de comunicare.

Dat fiind faptul că toate transferurile se efectuează prin magistrala comună, aceasta limitează drastic performanţele sistemului.

Debitul global al sistemului (*overall throughput*) este limitat de apariția *strangulării la nivel de magistrală* (*bus-bottleneck*), produs de solicitari simultane multiple ale masterilor.

În paragraful 3.2 vom detalia analiza de performanțe ale acestei conexiuni.

Este evident că reducerea traficului pe magistrală presupune o mai mare utilizare a resurselor proprii ataşate fiecărui procesor. Prin urmare, crearea unui sistem ierarhizat de memorii între procesoare şi magistrala comună devine o necesitate.

În [Coif 94] sunt prezentate mai multe soluții de ierarhizare a memoriilor.

Din punct de vedere fizic pot exista, în raport cu un procesor, *memorii locale* şi *memorii externe*.

Un procesor poate adresa memoria sa locală prin intermediul unui singur nivel de conectare- *magistrala locală*- şi o memorie externă prin intermediul mai multor nivele de conectare – *magistrale locale* şi *magistrala globală*. Pot exista memorii care să nu poată fi adresate de unele procesoare.

Din punct de vedere logic, memoriile se împart în *private* şi *comune*. Cele private pot fi adresate doar de procesoarele la care sunt conectate.

Procesoarele execută programe rezidente în memoriile locale. Ele cooperează prin transmiterea de mesaje prin intermediul memoriei comune partajate. Memoria comună (MC) poate fi implementată fie utilizând una sau mai multe module de memorie externă tuturor procesoarelor, fie distribuind-o în partea neprivată a memoriilor locale (ML).

Un procesor poate fi în una din următoarele stări:

- *activă* – execută un program din memoria sa privată;
- *de adresare* – cooperează cu alte procesoare utilizând MC;
- *de aşteptare* – aşteaptă să adreseze o MC;
- *blocată* – un procesor este blocat de un alt procesor care adresează segmentul de MC din memoria sa locală – ML.

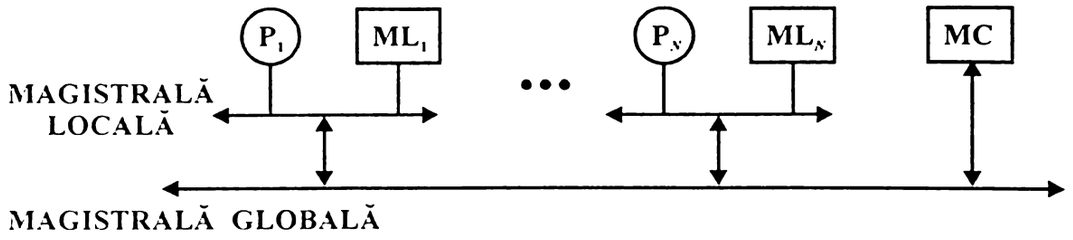
Structura 1 este prezentată în fig. 2.17 a). Se caracterizează prin existența unei memorii comune MC, externe oricărui procesor şi adresabilă doar prin magistrala globală. Fiecare procesor P dispune de o memorie locală ML, neadresabilă decât de procesorul la care este ataşată. Conflictul apare doar la adresarea memoriei MC.

Structura 2 este prezentată în fig. 2.17 b). Se observă distribuirea MC. Fiecare ML este configurată din două zone: una privată MP şi una comună MC. Fiecare P este conectat la ML proprie prin intermediul unei magistrale locale. Pentru ca un P_i să adreseze o zonă comună din ML_j ataşat unui alt procesor P_j se vor utiliza: magistrala locală a lui P_i, magistrala globală şi magistrala locală a lui P_j. Un procesor care primeşte accesul la magistrala globală deține prioritate pentru accesul la orice resursă. El devine prioritar în raport cu alte procesoare.

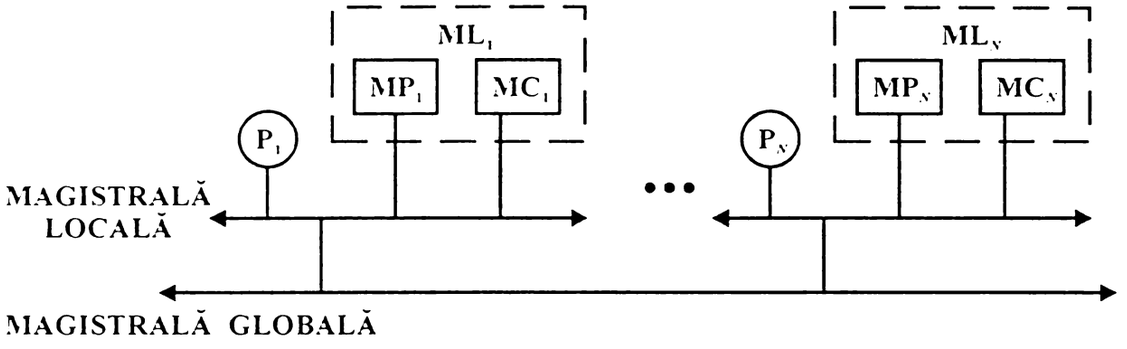
Dacă aceste procesoare erau în stare *activă* ele trec în *stare blocată*. Dacă erau în stare de *aşteptare* rămân în continuare în această stare dar eliberează magistrala locală. Acest protocol previne blocarea şi ameliorează performanțele.

Structura 3 este prezentată în fig. 2.17 c). Este o ameliorare a structurii 2 prin utilizarea unei memorii *dual-port* pentru a implementa zona comună a memoriilor locale ale fiecărui procesor. Zonele comune devin direct adresabile prin magistrala globală. Se scurtează timpul de transfer. Nu apare conflict nici la nivelul magistralelor locale şi nici la cel al memoriei dual-port care suportă două accese simultane. Conflictul apare doar la nivelul magistralei globale. In această structură un procesor nu poate adresa zona comună a propriei ML decât prin intermediul magistralei globale.

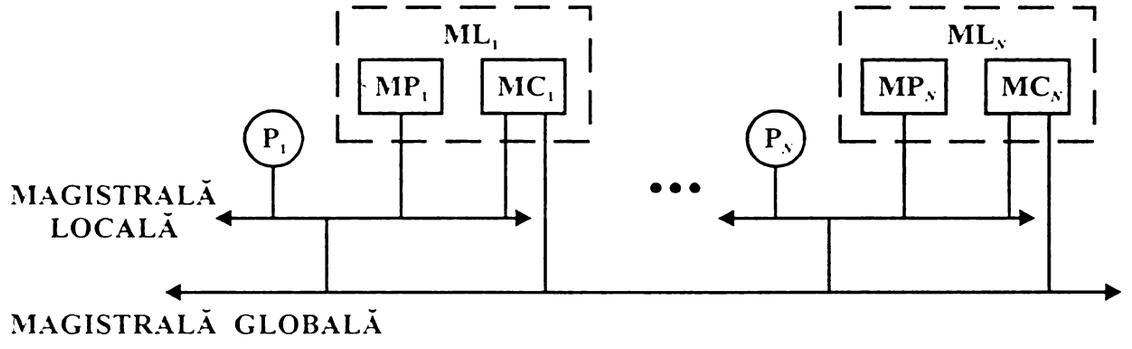
Structura 4 este prezentată în fig. 2.17 d). Reprezintă o variantă a structurii 3 în cazul în care nu există disponibilă o memorie dual-port. Zonele de memorie comună sunt externe tuturor procesoarelor. Ca şi la structura 3 procesoarele nu pot adresa zonele comune ale propriilor memorii prin intermediul magistralei globale ci dispun de o cale dedicată acestui scop. Doar un singur P_i poate adresa la un moment dat o memorie comună. Conflictul poate apare la nivelul memoriilor comune şi a magistralei globale. Prioritatea este dată accesului ce vine de pe magistrala globală.



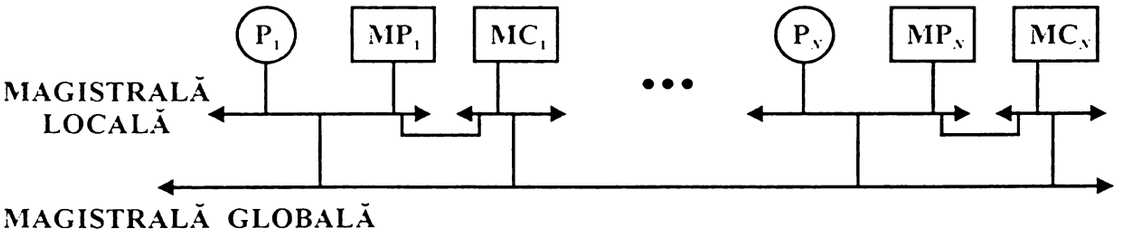
a) Structura 1



b) Structura 2



c) Structura 3



d) Structura 4

Fig. 2.17 Memorii ierarhizate pentru optimizarea traficului pe magistrală

Structura generală a unei magistrale cuprinde fire pe care circulă următoarele tipuri de semnale:

- adrese;
- date;
- control;
- întreruperi;
- acces la magistrală;
- tensiuni electrice de referință, etc.

În [Coif 94] sunt prezentate mai multe tipuri de magistrale dar ne vom referi doar la MULTIBUS, familia de magistrale dezvoltată de firma INTEL şi care face obiectul standardului IEEE 796.

La MULTIBUS spațiul de memorie maxim admisibil este 1MO iar cel de I/E de 64 K port-uri.

Modulele ce se pot conecta pe magistrală sunt de tip *master* şi *slave*. Un modul *master* poate comanda liniile de adrese şi de control şi deci poate prelua controlul magistralei. Un modul *slave* nu are această posibilitate. In cazul existenței mai multor masteri este necesară arbitrarea. Sunt permise două tipuri de arbitrări: serială şi paralelă. Tactul magistralei secvențiază operația de arbitrare.

Se pot efectua transferuri de date pe 8 sau 16 bit, existând două linii *A*o şi */BHEN* care indică lungimea cuvântului transferat. Există patru posibilități:

- transfer pe 16 bit;
- transfer de octet par, pe jumătatea inferioară a liniilor de date;
- transfer de octet impar, pe jumătatea superioară a liniilor de date;
- transfer de octet impar, pe jumătatea inferioară a liniilor de date.

O caracteristică a MULTIBUS-ului este că poate insera o memorie ROM suprapusă peste una RAM sau peste alta ROM. Facilitatea este utilă în cazul existenței unei rutine *bootstrap*, în cazul plasării porturilor în spațiul de memorie sau pentru testare.

MULTIBUS suportă două tipuri de întreruperi: *vectorizate* şi *nevectorizate*. Există 8 linii pentru cereri de întreruperi. Cele vectorizate cer transferul unui vector de la un *master* către un *slave* utilizând secvența de acceptare indicată de activarea liniei */INTA*. Cele

nevectorizate presupun că localizarea sursei de întrerupere se face în cadrul masterului fără a mai fi necesar transferul unui vector.

MULTIBUS este compusă din 73 de linii după cum urmează:

- 20 de adrese;
- 16 de date;
- 1 pentru inițializare;
- 2 pentru inhibare;
- 1 pentru stabilirea, împreună cu Ao, a lungimii cuvântului transferat;
- 7 pentru arbitrare şi control;
- 5 pentru comanda transferurilor de date;
- 9 pentru sistemul de întreruperi;
- 4 pentru sesizarea stării tensiunii de alimentare;
- 4 auxiliare;
- 4 pentru alimentare: +5V, -5V, +12V, 0V.

Există terminale pentru dezvoltări ulterioare.

Caracteristicile RIN de timp magistrală comună sunt:

- avantaje:
  - simplitate logică şi constructivă;
  - posibilitate de extensie cu module definite şi proiectate în jurul ei (arround it). Famila MULTIBUS este un bun exemplu.
- dezavantaje:
  - saturarea rapidă a magistralei cu repercursiuni negative asupra latenței RIN;
  - toleranța minimă la defect, orice defecțiune conducând la căderea catastrofală a întregului sistem.

RIN de tip *magistrală comună multiplă* păstreză principiul magistralei comune singulare dar îi şi ameliorează performanțele din punct de vedere al toleranței la defecte.

In fig. 2.18 se prezintă schematic structura unui SMP – MP cu magistrală multiplă.

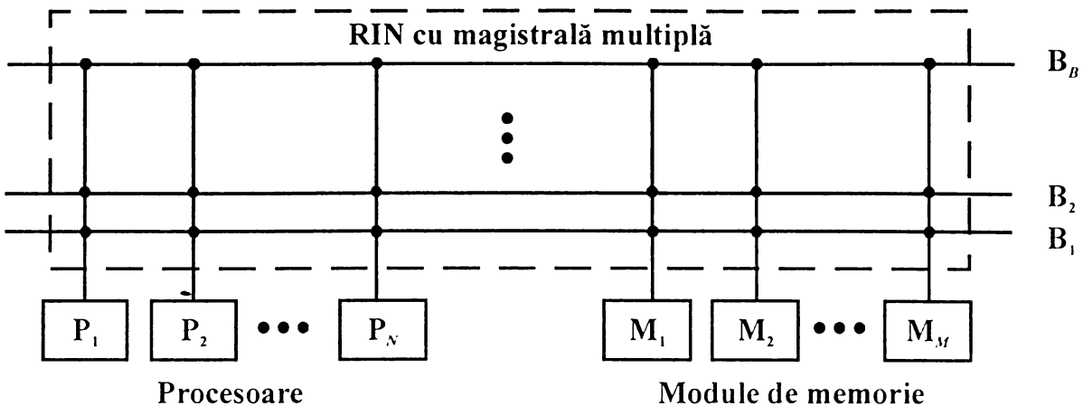


Fig. 2.18 SMP- MP cu magistrală multiplă

Se produce o decongestionare a traficului deci o scădere a latenței RIN şi în acelaşi timp o ameliorare a toleranței la defect. Creşte complexitatea şi costul.

In cap. 3.3 se studiază amănunțit acest RIN din punct de vedere al performanțelor şi se descrie structura unui SMP – MP construit în jurul unei astfel de magistrale.

2.4.3 Rețele de interconectare statice

Rețelele de interconectare statice (*static interconnection network*) sunt acelea care permit numai legături fixe (uni sau bidirecționale) între două noduri. Acestea sunt potrivite pentru conexiuni procesor – procesor, adică în SMP – TM.

In literatură [Prad 96], [Hwan 93] sunt descrise pe larg topologii de RIN-statice. În continuare ne vom referi la topologiile *arbore* (*tree*), *plasă* (*mesh*) şi *hipercub* (*hypercube*) deoarece aici se pot face intervenții semnificative pentru exploatarea potențialului de toleranță la defecte prin reconfigurare.

RIN *arbore*

Sunt într-o oarecare măsură similare cu cele de tip *stea* în sensul că nodul rădăcină (apex) se foloseşte pentru intermedierea transferului de la o ramură a arborelui la cealaltă. În general de la nodul sursă se parcurge arborele în sus până când se întâlneşte un nod comun cu calea până la nodul destinație, după care se parcurge această cale descendent. Numărul de legături al RIN este N-1, adică costurile cresc liniar cu dimensiunea. Defectarea unui nod critic – apex-ul sau unul apropiat lui – are urmări catastrofale asupra întregii RIN.

Într-o rețea regulată cu b ramuri sub fiecare nod şi cu j niveluri, numărul total N de noduri este:

$$N = F(j) = 1 + b + b^2 + \ldots + b^{j-1} = \frac{b^j - 1}{b - 1} \tag{2.4}$$

În literatură [Witt 81] se calculează intensitate traficului de mesaje pe o legătură de deasupra unui nod din nivelul k. Se consideră un timp de N-1 unități în care fiecare nod va trimite în medie câte un mesaj la celălalt. În acest timp fiecare nod din subarborele de sub nodul considerat - care are $j - (k-1)$ niveluri - va schimba o pereche de mesaje între ele şi restul de noduri ale RIN. Prin urmare densitatea $T(k)$ a traficului pe linia de deasupra nodului de nivel k va fi :

$$T(k) = \frac{2F(j)(j-k+1)[N - F(j)(j-k+1)}{N-1} \tag{2.5}$$

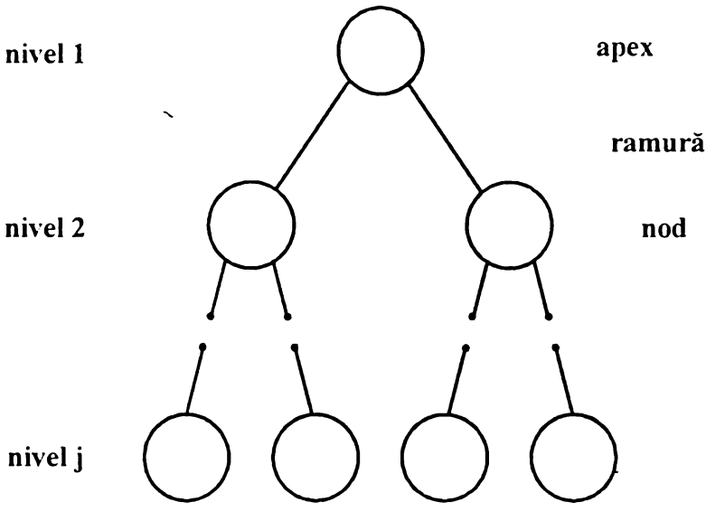


Fig. 2.19 RIN *arbore*

RIN *plasă*

O arie de procesoare bidimensională de tip plasă care interconectează vecinii cei mai apropiați (nearest neighbour mesh) se poate obține dacă fiecare nod se conectează prin câte o legătură la cei patru vecini, iar capetele libere pot fi recirculate pe partea opusă. Dacă pe o latură sunt W procesoare atunci numărul total de procesoare este $N = W^2$ iar în

cazul general al unei *plase regulate* într-un spațiu D - dimensional acest număr este N = W^D.

Distanța maximă de transmitere a unui mesaj în orice direcție este de $W/2$, iar lungimea medie de transmitere a mesajelor este $DW/4$. [Witt 81]. Deoarece fiecare nod se conectează la fiecare din cele D dimensiuni la câte doi vecini, există $2D$ legături la un nod.

Pentru RIN vor exista în total $2DN/2 = DN$ legături.

Densitatea medie a transferului pentru fiecare legătură este :

$$T = \frac{ND(W/4)}{ND} = \frac{W}{4} \tag{2.6}$$

Un nod al rețelei trebuie să dispună de $2D$ porturi de conectare la vecini. Dacă W crește cresc întârzierile în transmiterea mesajelor.

RIN *hipercub*

Structura de tip *hipercub* (*hypercube*) se caracterizează prin prezența a $N = 2^D$ procesoare, interconectate sub forma unui cub binar D dimensional. Fiecare nod are legături directe şi separate la alte D noduri vecine. Adresele a două noduri diferă printr-un bit. Există în total 2^D adrese binare asociate nodurilor.

Hipercubul generalizat (*generalized hypercube*) are lățimea W (noduri pe latură) dar păstrează adiacența adreselor. Numărul total de noduri este $N = W^D$ ca şi la RIN *plasă*. Conectarea legăturilor necesită doar D porturi/nod în timp ce la *plasă* sunt necesare $2D$/nod. Există în total ND/W laturi. Numărul mediu de paşi pentru transmiterea unui mesaj este de $D(W-1)/W$ [Witt 81].

Densitatea traficului pe fiecare latură este :

$$T = \frac{ND(W-1)/W}{ND/W} = W - 1 \tag{2.7}$$

Extensia dimensiunii unui *hipercub* este mai dificilă decât a unei rețele *plasă*, deoarece creşterea numărului de noduri care folosesc o latură W este limitată de T admisibil, iar creşterea lui D presupune mărirea numărului de porturi la fiecare nod.

Hipercubul are proprietatea că el poate fi definit inductiv.

Un *hipercub* de ordinul 0 este un nod, în timp ce *hipercubul* de ordin $n+1$ este construit din două *hipercuburi* de ordin n şi conectând nodurile lor corespunzătoare.

În fig. 2.20 se prezintă un *hipercub* de ordinul 4.

De notat că nodul unui *hipercub* constă dintr-un procesor, memoria sa locală, porturi de comunicare pentru alte noduri şi porturi I/E.

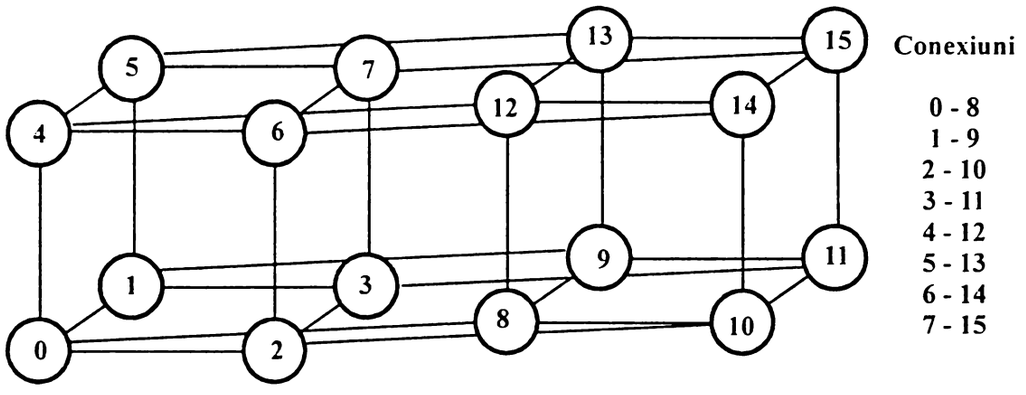


Fig. 2.20 Un hipercub cu 16 procesoare

RIN *hipercub* are câteva proprietăţi interesante :

1. Cu cât numărul de procesoare creşte, creşte logaritmic şi numărul de conexiuni şi hardul asociat şi prin urmare sisteme cu număr mare de procesoare sunt realizabile.
2. Un *hipercub* este un supraset a altor RIN statice ca şi *inel, arbore* etc. deoarece toate acestea pot fi transformate în *hipercub* ignorând unele conexiuni.
3. *Hipercuburile* sunt scalabile, o proprietate ce rezultă direct din proprietatea de recursivitate a lui.
4. Schemele de direcţionare sunt simple. O politică de rutare a mesajului poate fi aceea prin care se trimite un mesaj la vecinul a cărui etichetă binară se potriveşte cu eticheta destinaţiei finale în următorul rang binar.

Lungimea căii de comunicaţie între două noduri este exact numărul de biţi prin care etichetele diferă. Maximum este n iar media $n/2$.

Există numeroase căi posibile între două noduri; această redundanţă se foloseşte pentru creşterea lui T şi a toleranţei la defecte.

RIN *hipercub* a fost utilizată la SMP- TM elaborate de Caltech, AMETER, Intel etc.

În literatură [Mold 93] se face o prezentare sistematică a evoluției acestei topologii până la Intel Touchstone Delta - o familie de supercalculatoare ce şi-a făcut debutul în 1990.
Protocolul de rutare a mesajelor utilizează pentru prima dată tehnica *wormhole* propusă de Dally şi Seity în 1987 şi implementată la SMP-TM AMETEK 2010.
Rutarea *wormhole* diferă de comutarea de pachete ce utilizează tehnica *store-and-forward*. Pachetele sunt manipulate prin comutatoare speciale şi nu software ca la majoritatea sistemelor bazate pe hipercub. Când pachetul din fruntea mesajului găseşte un canal ocupat, pachetele din secvență sunt blocate. Pachetul din fruntea mesajului, uzual constând din câțiva bit, poziționează comutatoarele astfel încât restul pachetelor poate să urmeze fără a se consuma timp. Această tehnică de rutare reduce memoria asociată nodului şi în acelaşi timp software-ul necesar pentru manipularea mesajelor.

2.4.4 Rețele de interconectare dinamice

Într-o RIN dinamică conexiunea între două noduri (procesor – memorie la SMP–MP şi procesor – procesor la multicalculatoare) se stabileşte prin intermediul unor comutatoare electronice denumite *bloc de comutare (switching box)* sau mai simplu, *comutatoare*. Schimbarea stării acestora poate face posibilă realizarea comunicației între diferite perechi de noduri. RIN care admit conexiuni simultane între toate combinațiile de intrări şi ieşiri se numesc *rețele fără blocare*. Ele devin *strict fără blocare* dacă indiferent de conexiunile existente şi fără a le perturba, întotdeauna se poate stabili o cale între oricare intrare nefolosită şi oricare ieşire nefolosită. Unele rețele fără blocare permit efectuarea unei conexiuni I/E fără afectarea celor existente numai dacă acestea din urmă au fost înfăptuite pe baza unui *algoritm de rutare (routing algorithm)* prescris. Aceste RIN sunt considerate a fi *fără blocare într-un sens larg* (*wide sense nonblocking*).
Pentru a reduce numărul de comutatoare majoritatea rețelelor sunt *rețele cu blocare*, adică nu fac posibile toate combinațiile de conexiuni I/E simultan. Dacă căile blocate pot fi totuşi restabilite şi redirecționate prin modificarea stării comutatoarelor, atunci rețeaua se numeşte *rearanjabilă*.
La o RIN dinamică multinivel (RIN-MN), cea mai complexă RIN, se disting următoarele elemente componente:

- *comutatorul*;

- *conexiunea* ce asigură realizarea funcției de interconectare (permutare);
- *nivelul de comutare* sau *etajul de comutare*.

Asociat acestora, se studiază *mecanismul de rutare a informației*.

Comutatorul

Comutatorul stabileşte calea de conectare între o intrare şi o ieşire. Complexitatea comutatorului depinde de tipul de RIN în care este utilizat: mică pentru RIN – *crossbar* şi mare pentru RIN *multinivel*.

Literatura de specialitate [Dasg 89], [Hwan 93], [Popa 96], [BWK 98] ş.a. prezintă *in extenso* mai multe tipuri. Reținem trei de tipul:

patratic, arbitru, distribuitor, prezentate în fig. 2.21

a. patratic **b. arbitru** **c. distribuitor**

Fig 2.21 Comutatoare

Comutatorul de tip *patratic* are acelaşi număr de intrări şi de ieşiri.

Comutatorul de tip *arbitru* (multiplexor) asigură selectarea uneia dintre intrări şi transferul ei la unica ieşire.

Comutatorul de tip *distribuitor* alocă unei intrări o anumită ieşire.

Cel mai utilizat este comutatorul patratic cu două intrări şi două ieşiri, 2 x 2, ce poate fi de două tipuri: cu două sau cu patru stări.

În fig. 2.22 se prezintă un astfel de comutator şi cele patru stări posibile ale sale cu transfer:

- direct;
- în cruce;
- superior;
- inferior.

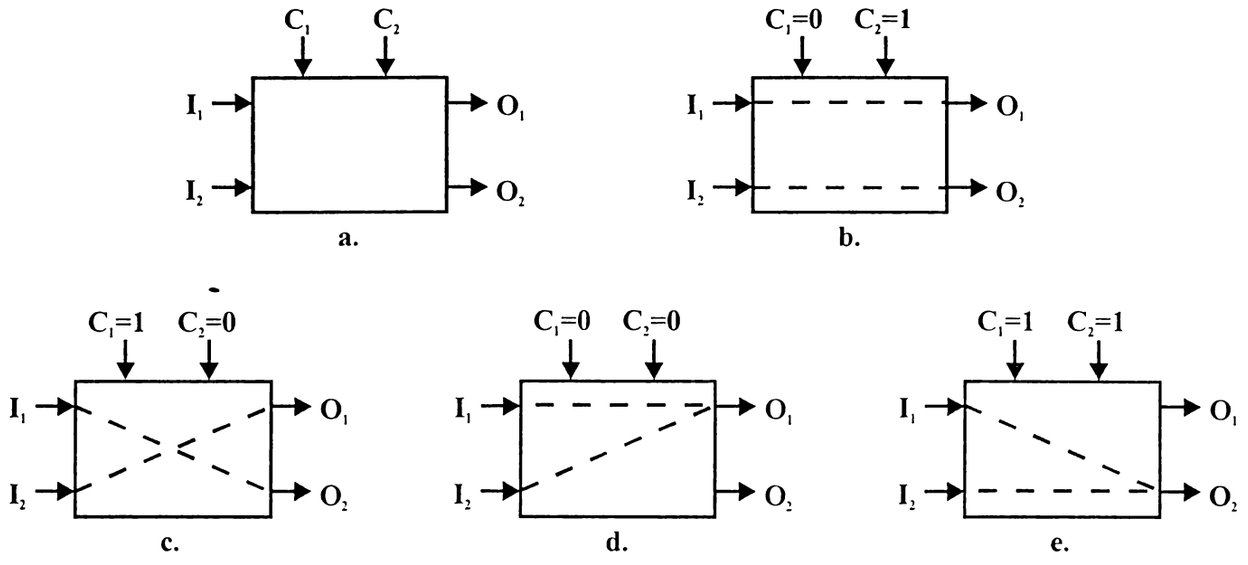


Fig. 2.22 Comutator 2 x 2

a) reprezentarea comutatorului;
b) transfer direct;
c) transfer în cruce;
d) transfer superior;
e) transfer inferior.

Stările descrise la b) şi c) nu sunt conflictuale, asigurând comutatorului atributul de *fără blocare* în timp ce stările prezentate la d) şi e) sunt conflictuale deoarece ambele intrări concură pentru O_1 respectiv O_2. În acest caz comutatorul este *cu blocare*.

Uzual, în cazul comutatoarelor fără blocare, se utilizează o singură intrare de comandă *c* care respectă specificaţiile:

if $c = 0$ **then** /* *conectare directă* */

$new\ O_1 = I_1$ &

$new\ O_2 = I_2$

else /* *conectare încrucişată* */

$new\ O_1 = I_2$ &

$new\ O_2 = I_1$.

În acest caz comutatorul va avea doar două stări ca şi în fig. 2.23.

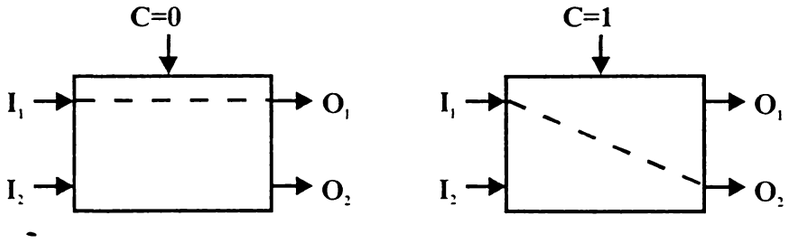


Fig. 2.23 Comanda unei intrări

Pentru evitarea situațiilor conflictuale prezentate mai sus se utilizează comutatoare ce au incorporate tampoane pentru memorarea pachetului ce trebuie transferat. Aceste tampoane sunt memorii FIFO cu o capacitate depinzând de intensitatea traficului şi de lungimea mesajelor.

Funcții de interconectare

Funcțiile de interconectare realizează permutări ale ordinii procesoarelor.

Permutările sunt funcții bijective simple în care fiecare intrare este mapată în una şi numai una dintre ieşiri.

Ele au un rol important în proiectarea RIN. Prin combinarea mai multor bijecții, se pot obține conexiuni complexe.

Când se execută o *funcție de rutare* f într-o RIN, datele de la nodul PE_i sunt transmise la nodul $PE_{f(i)}$.Această rutare a datelor apare simultan pentru toate nodurile active PE_i din set. Un nod inactiv PE poate să recepționeze date de la alt PE când se execută o funcție de rutare, dar el nu poate să transmită date. Este necesară asigurarea unei adrese la fiecare PE din matrice. Se pot utiliza mai multe scheme de adresare. Procesoarele pot fi adresate utilizând câmpuri de adresă de m bit, în acest caz $N = 2^m$. In alte situații când nodurile PE sunt dispuse într-un câmp bidimensional, se utilizează coordonate carteziene.

Fie n o adresă a unui nod PE exprimat printr-un câmp de adresă de m bit:

$$n = (b_m, b_{m-1}, \cdots, b_1).$$

Interconectările din rețea pot fi descrise ca şi permutări asupra reprezentărilor binare ale intrărilor.

In literatură [Mold 93] se descriu mai multe funcții de interconectare.

Schimb

Permutarea *schimb (exchange)* este definită ca:

$$E_k = (b_m, \ldots, \bar{b}_k, \ldots, b_1) \quad 1 \le k \le m \tag{2.8}$$

Structura RIN ce se obține aplicând $E_k(x)$ pentru $m = 3$ se numește *cub de ordinul 3*.

$E_1(x) = \bar{b}_3 b_2 b_1$ corespunde legăturilor orizontale

$E_2(x) = b_3 \bar{b}_2 b_1$ corespunde legăturilor oblice

$E_3(x) = \bar{b}_3 b_2 b_1$ corespunde legăturilor verticale.

În fig. 2.24 se prezintă conexiunile din cubul de ordin 3.

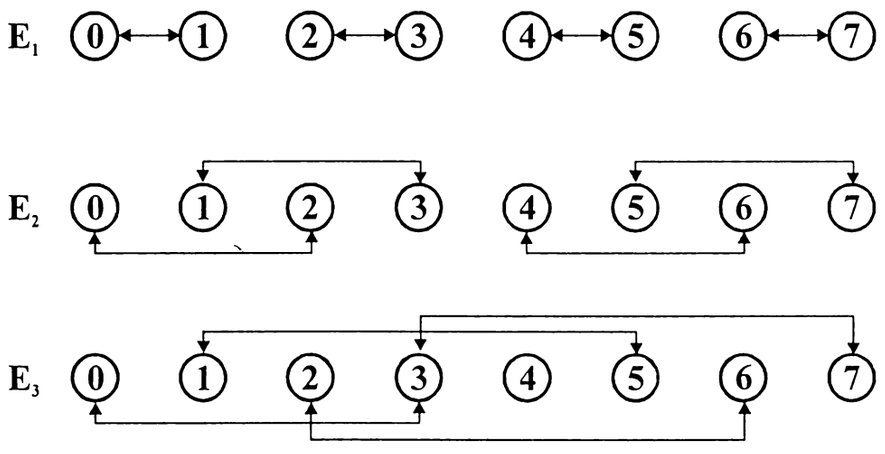


Fig. 2.24 Conexiuni în cubul de ordinul 3

În general într-o RIN bazată pe această funcție de permutare, cele N noduri sunt conectate prin $N \times 2^{N-1}$ legături. Ordinul k al cubului este $k = \log_2 N$.

Amestec perfect

Permutarea *amestec perfect (perfect shuffle)* este definită ca:

$$S(x) = (b_{m-1}, b_{m-2} \ldots, b_1, b_m) \quad \text{- direct} \tag{2.9}$$

$$S'(x) = (b_1, b_m, \ldots, b_3, b_2) \quad \text{- invers} \tag{2.10}$$

RIN bazată pe *S(x)* în cazul $m = 3$ este prezentată în fig. 2.25 a iar pe *S'(x)* în fig. 2.25 b.

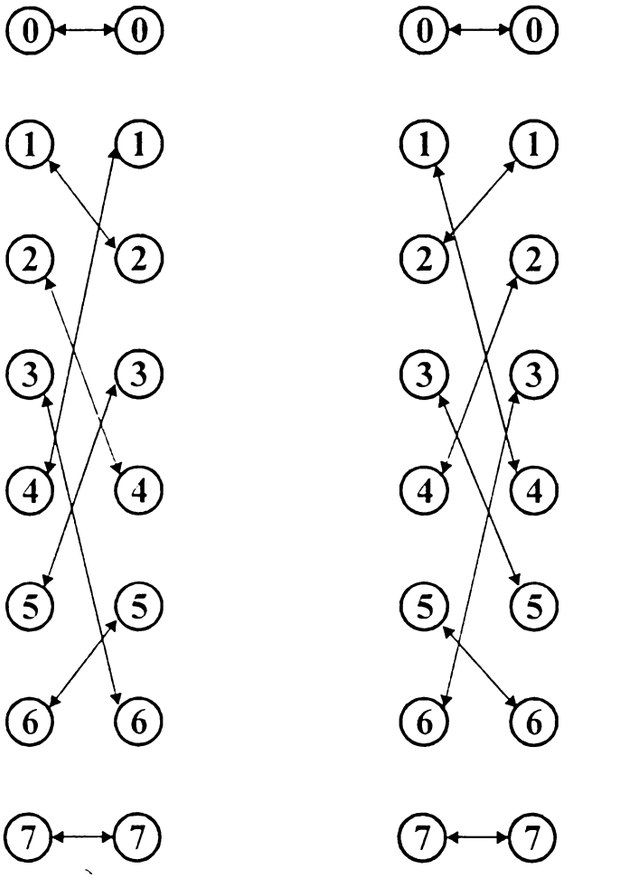


Fig. 2.25 Interconectare *amestec perfect*

Se pot defini subseturi respectiv supraseturi ale lui *S(x)*. A *k*-a subfuncție S_k este:

$$S_k(x) = (b_m, \ldots, b_{k-1}, b_{k-1}, \ldots, b_1, b_k) \tag{2.11}$$

iar a *k*-a suprafuncție $S^k(x)$ este:

$$S^k(x) = (b_{m-1}, \ldots, b_{m-k+1}, b_m, b_{m-k}, \ldots, b_1). \tag{2.12}$$

Fluture

Permutarea *fluture (butterfly)* este definită ca şi:

$$B(x) = (b_1, b_{m-1}, \ldots, b_2, b_m). \tag{2.13}$$

Interconectarea bazată pe această funcție, pentru cazul $m = 3$ este prezentată în fig. 2.26.

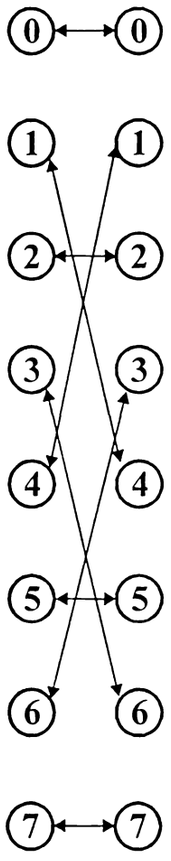


Fig. 2.26 Interconectarea *fluture*

Se pot defini în manieră similară:

$$B_k(x) = (b_m, \ldots, b_{k+1}, b_1, b_{k-1}, \ldots, b_k) \tag{2.14}$$

$$B^k(x) = (b_{m-k+1}, \ldots, b_{m-k+2}, b_m, b_{m-k}, \ldots, b_1) \tag{2.15}$$

Bit oglindit

Permutarea *bit oglindit (bit reversal)* se defineşte:

$$R(x) = (b_1, b_2, \ldots, b_m)\,. \tag{2.16}$$

Pentru cazul $m = 3$ conexiunea este prezentată în fig. 2.27.

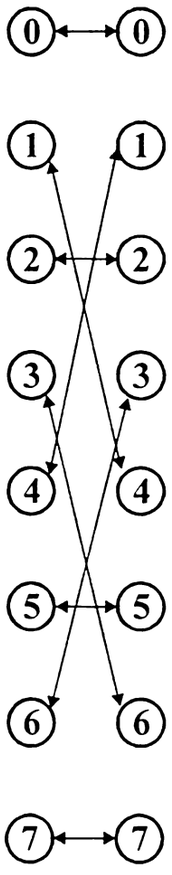


Fig. 2.27 Interconectare *bit-oglindit*

Rețeaua Illiac

Aceasta este utilizată în calculatorul Illiac IV şi se defineşte astfel:

$$\begin{aligned} R_{+1}(x) &= (x+1) \mod N \\ R_{-1}(x) &= (x-1) \mod N \\ R_{+r}(x) &= (x+r) \mod N \\ R_{-r}(x) &= (x-r) \mod N . \end{aligned} \qquad (2.17)$$

Pentru cazul în care $N = 16$ şi $r = 4$, această rețea este prezentată în fig. 2.28.

Ea este o RIN parțial conectată. Când $N = 16$, patru noduri PE pot fi atinse într-un pas, şapte în doi paşi şi unsprezece PE în trei paşi.

În general sunt necesari $I \le \sqrt{N} - 1$ paşi pentru a conecta fiecare nod PE.

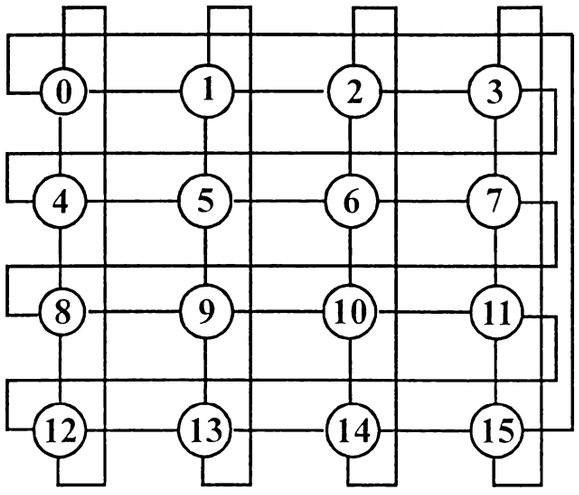


Fig. 2.28 Reprezentarea rețelei Illiac

Deplasare

Permutarea *deplasare (shift)* este definită ca şi:

$$SH(x) = |x + 1|_N \tag{2.18}$$

unde $|\ \ |_N$ înseamnă *modulo N*.

Interconectarea bazată pe deplasare, în cazul m = 3, este prezentată în fig. 2.29.

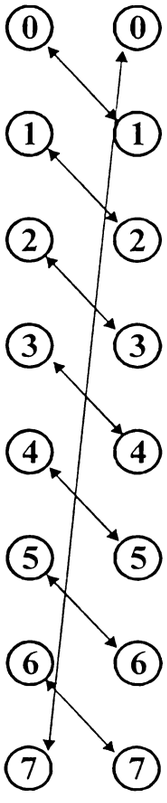


Fig. 2.29 Interconectarea bazată pe *deplasare*

Se pot determina realţii diferite între permutări.
De exemplu pentru m = 3,

$$R_3 = B_3 .$$

Pentru anumite interconectări, ca de exemplu la reţeaua Illiac, este preferabil să se utilizeze direct indexul fiecărui PE_x, pentru $0 \le x \le N-1$. Conexiunile de la PE_x sunt definite de către elementul de procesare destinaţie.

Mecanisme de rutare a informaţiei

Prin *mecanism de rutare a informaţiei* se înţelege totalitatea procedurilor / algoritmilor utilizate pentru controlul transferului informaţiei de la un nod la următorul.
Un astfel de mecanism trebuie să asigure:

- integritatea informaţiei
- calea de date optimă între transmiţător şi receptor
- evitarea saturării căilor de date
- găsirea căii alternative în cazul defectării unei căi de date.

Alegerea mecanismului de rutare este impusă de caracteristicile generale ale RIN.
În continuare se descriu câteva mecanisme de direcţionare:
Rutarea prin saturare. Dacă un nod primeşte un mesaj şi dacă nu-i este destinat îl va transmite simultan către toţi vecinii, cu excepţia transmiţătorului. Avantajul principal îl reprezintă simplitatea iar dezavantajul major creşterea traficului din care o mare parte se datorează metodei.
Rutarea aleatoare constă în modul aleator în care se aleg căile de date. Nodul care primeşte un mesaj ce nu îi este destinat îl va transmite în continuare după o anumită lege de distribuţie. Avantajele şi dezavantajele sunt ca şi la metoda anterioară.
Rutarea fixă presupune generarea unei *matrici de rutare* pentru o topologie dată.
Matricea se construieşte din atâtea linii şi coloane câte noduri sunt, la intersecţia unei linii cu o coloană găsindu-se acel nod care este intermediarul între nodul care corespunde liniei şi cel care corespunde coloanei.
Pentru RIN dinamice, în literatură [Hwan 93], [Popa 96], [Mold 93] se prezintă mai multe mecanisme:

- *Utilizarea adresei destinaţiei pentru stabilirea căii de comunicaţie.*

Procesorul transmițător generează o *etichetă de rutare (routing tag)* care are un număr de biți egal cu numărul de niveluri. Fiecare rang controlează câte un nivel. La intrarea fiecărui nivel *j*, comutatorul verifică al *j-lea* bit al etichetei, începând cu primul nivel, analizând c.m.s. bit. Dacă bitul este 0, comutatorul selectează portul superior; dacă bitul este 1 se selectează portul inferior.

- *Calcularea funcției SAU-EXCLUSIV între adresele sursei și destinației.*

Combinația rezultată va controla poziționarea comutatoarelor. Dacă adresele sursei și destinației sunt:

$$a_n a_{n-1} \cdots a_i \cdots a_1 \text{ respectiv } b_n b_{n-1} \cdots b_i \cdots b_1$$

atunci rezultatul este:

$$c_n c_{n-1} \cdots c_i \cdots c_1$$

unde $c_i = a_i \oplus b_i$ controlează nivelul *i*.

- *Utilizarea unei funcții de control.*

Această funcție are atâtea ranguri câte niveluri de comutatoare există. Valoarea 0 a unui rang va stabili conexiunea directă iar valoarea 1 conexiunea în cruce. Generarea funcției de control este sarcina modulului ce asigură controlul întregii rețele.

A. RIN dinamice pe un singur nivel.

O RIN dinamică pe un singur nivel este formată din:

- *conexiunile de realizare a funcției de interconectare*;
- *nivelul de comutare* conținând comutatoare.

Schema bloc este prezentată în fig. 2.30.

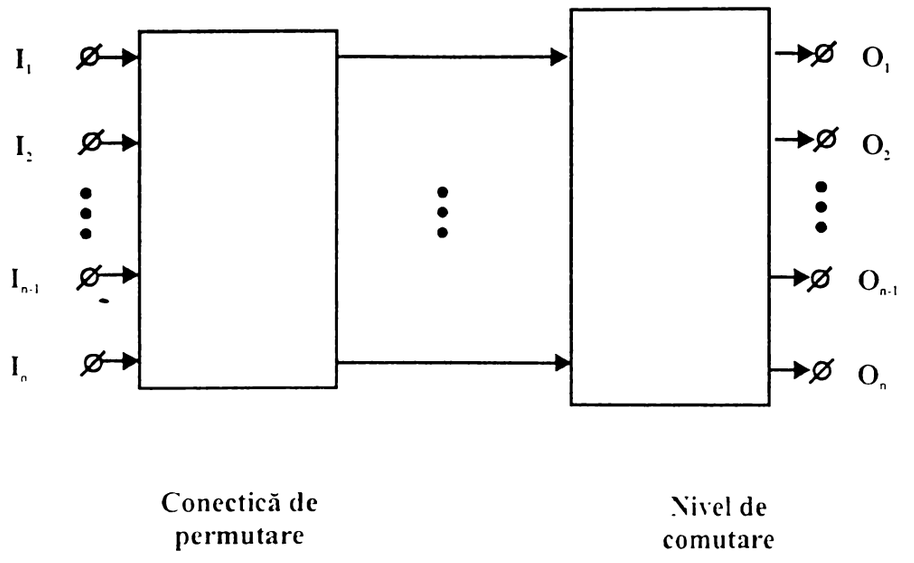


Fig. 2.30 Schema bloc a unui RIN dinamic *cu un singur nivel*.

Un exemplu clasic este aceea a RIN bazată pe funcția de interconectare *amestec perfect* şi varianta *amestec perfect invers*.

Schemele celor două variante de RIN sunt prezentate în fig. 2.31 a şi b.

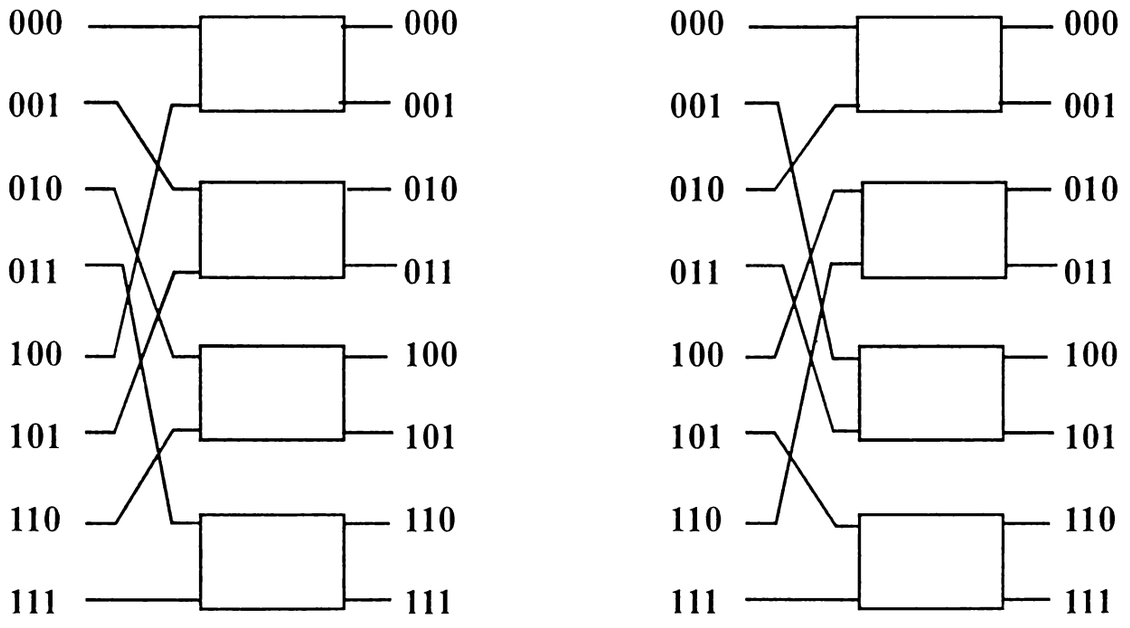


Fig. 2.31 RIN ce realizează funcția *amestec perfect*

RIN dinamice pe un *singur nivel* au o complexitate medie dar oferă posibilitați limitate de conectare, numărul permutărilor posibil la o trecere fiind mic. Conectivitatea totală se poate obține recirculând informația prin rețea, de unde şi denumirea de RIN cu *recirculare*.

Datorită faptului că la fiecare permutare noua adresă se obține din precedenta printr-o rotire rezultă că numărul maxim de recirculări este egal cu $\log_2 m$ unde m reprezintă numărul de ranguri al adresei.

RIN *crossbar*

Este o RIN dinamică pe un singur nivel, deplin conectată (fully connected). Este o rețea fără blocare permițând conectarea simultană a N elemente de procesare PE la M memorii. Accesele simultane la toate modulele de memorie conferă RIN un mare debit de prelucrare a informației. Situația conflictuală poate apare atunci când mai mult elemente de procesare solicită acelaşi modul de memorie.

Dacă $N = M$ atunci se utilizează N^2 comutatoare, ceeea ce face costul $O(N^2)$ prohibitiv față de $N/2 \log N$ a unui RIN *omega*.

Un PE este format dintr-un procesor P cu resurse locale de memorie ML conectate în jurul unei magistrale locale. RIN crossbar se utilizează în SMP–MP cu număr redus de procesoare. Un exemplu îl constituie sistemul C.mmp [Dasg 89] care utilizează 16 module procesoare PDP 11 şi 16 module de memorie partajate având o capacitate totală de 32 Mbyte.

Structura unei RIN *crossbar* de dimensiuni $N \times M$ este prezentată în fig. 2.32.

Intr-o astfel de rețea funcția de comutare şi comanda sa sunt distribuite de-a lungul celor $N \times M$ comutatoare. Este posibilă concentrarea funcției de comutare la fiecare dintre cele M module de memorie, rezultând un *sistem cu memorie multiport*.

Structura generală a unui astfel de sistem este prezentată în fig. 2.33.

Fiecare modul de memorie are n porturi, un port / procesor. Comutatorul asociat fiecărui modul de memorie conține şi logica de arbitrare.

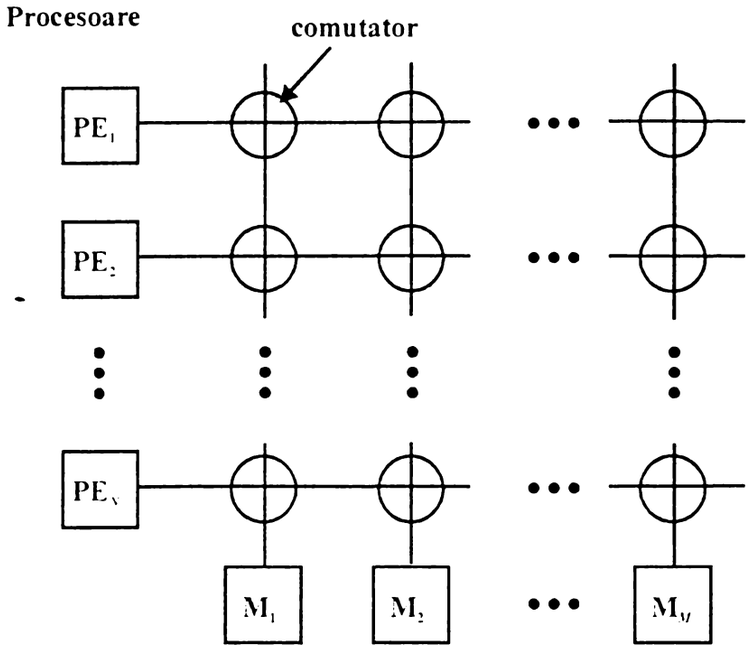


Fig. 2.32 RIN *crossbar*

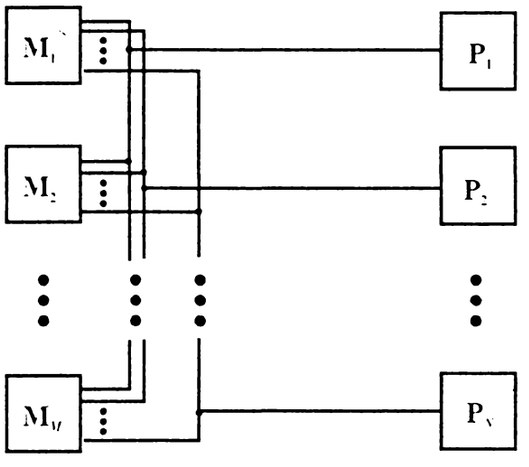


Fig. 2.33 Memorii multiport

B. RIN dinamice multinivel

Asigură un bun compromis performanță / cost permițând stabilirea unei căi de comunicație între intrare şi ieşire într-un timp mai scurt decât cel obținut într-o RIN dinamică mononivel cu N intrări şi N ieşiri, există $\log_2 N$ niveluri de comutare 2 x 2.

Pe un nivel numărul de comutatoare este $N / 2$.

Shcema bloc a unui RIN dinamic *multinivel* este prezentată în fig. 2.34.

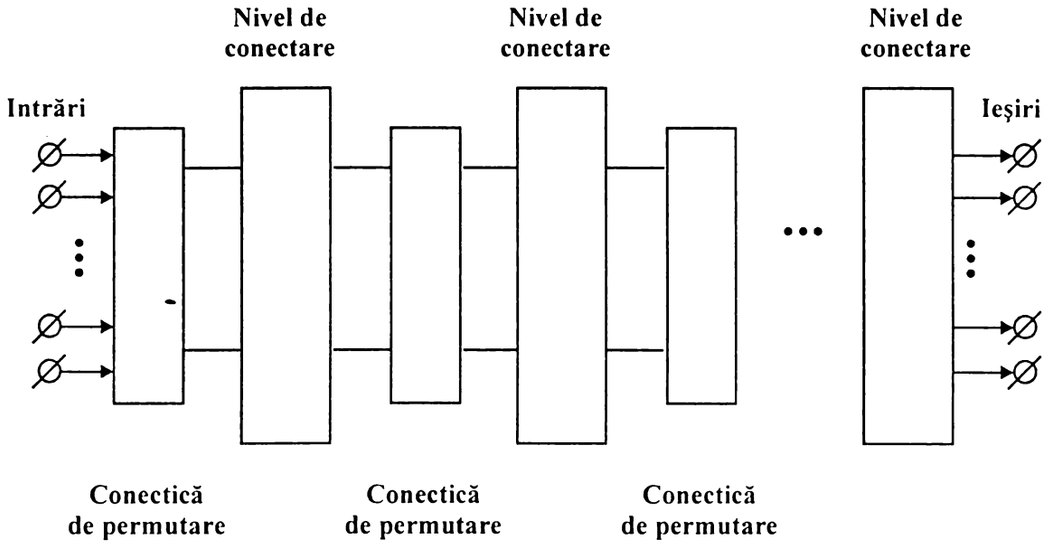


Fig. 2.34 Schema bloc a unui RIN dinamic *multinivel*

Diferențe importante apar între RIN dinamice *pe un singur nivel* şi RIN dinamice *multinivel*:

- O RIN dinamică *multinivel* poate fi utilizată atât la transferul prin comutare de circuite cât şi la cel prin comutare de pachete. O RIN *pe un singur nivel* nu poate transfera decât pachete.
- O RIN dinamică *multinivel* are viteza de transfer mai mică deoarece informația parcurge mai multe niveluri înainte de a ajunge la destinație, pe când la o RIN *pe un singur nivel*, informația sau părți din informație ajung la destinație după fiecare iterație.

RIN *omega*

Această RIN a fost studiată de D. Lawrie şi a fost utilizată în multiprocesorul Cedar şi o variantă ameliorată, la maşina NYU Ultracomputer. Este descrisă pe larg în [Hwa 93], [Mold 93], [MHB 86] ş.a.

In varianta sa inițială conectează N procesoare la N memorii.

Se caracterizează prin:

- lățime de bandă proporțională cu N;
- pentru $N = 2^k$ există $k = \log_2 N$ niveluri identice;
- între două niveluri adiacente conexiunea este de tip *amestec perfect*;
- $N/2$ comutatoare pe fiecare nivel;
- $N \log_2 N$ comutatoare per ansamblu;
- comandă independentă a comutatoarelor pe fiecare nivel;

- latența, logaritmică în N.

Schema unei RIN *omega* 8 x 8 este prezentată în fig. 2.35.

Ea constă din trei niveluri fiecare conținând patru comutatoare 2 x 2.

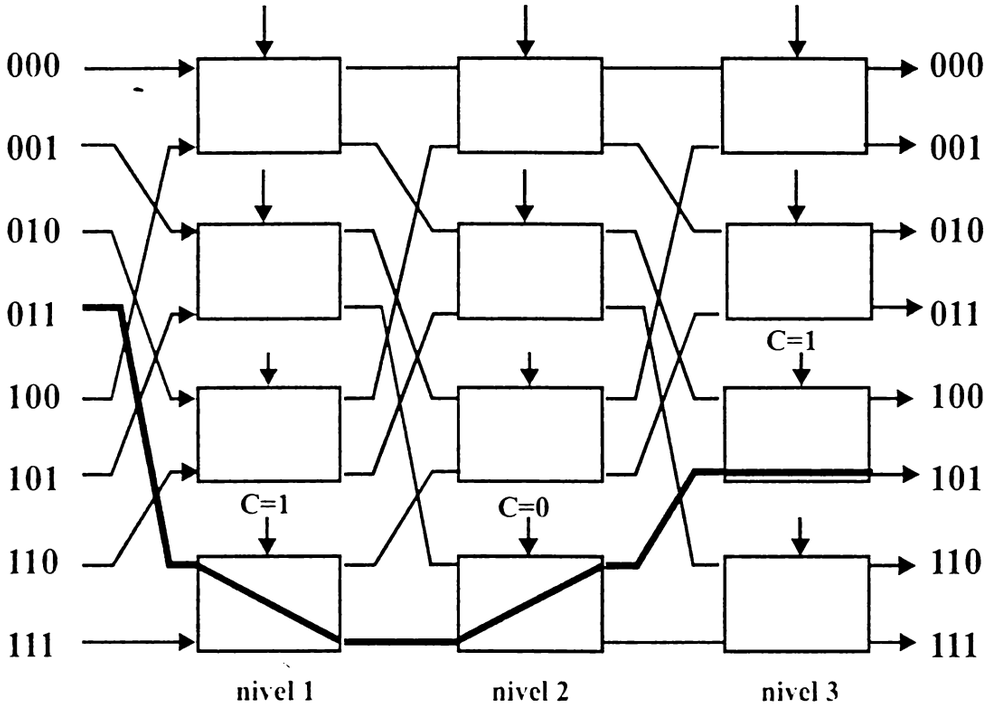


Fig. 2.35 RIN *omega* conectând intrarea 3 cu ieşirea 5

Procesorul conectat la intrarea 3 (011) generează o etichetă care este reprezentarea binară a destinației, în cazul nostru 5 (101). Conectarea unui comutator dispus pe nivelul i este acționat de al i-lea bit al etichetei contorizat de la C.m.s.b. În figură se vede traseul parcurs de un mesaj de la intrarea 3 la ieşirea 5.

În general considerându-se că se transmit date de la dispozitivul de intrare având numărul I la dispozitivul de ieşire având numărul O şi că reprezentarea binară este:

$$I = i_1 i_2 \ldots i_k \quad \text{şi} \quad O = o_1 o_2 \ldots o_k .$$

Se poate formaliza astfel:

For j = 1 ··· k **do**

If O_j = o **then** pune pe 0, *c* de pe nivelul j conectat la I

else pune pe 1, *c* de pe nivelul j conectat la I

endfor

Reamintim că la c = 0 se conectează intrarea cu ieşirea superioară iar la c = 1 cu ieşirea inferioară în cadrul comutatorului.

Inițial, dispozitivul I este conectat la un comutator dispus pe nivelul 1 a RIN. Apoi algoritmul de direcționare permite comutatoarelor relevante de pe nivelurile 1, 2,..., *k* să fie poziționate succesiv astfel ca transmisia să fie efectuată dela I la O.

În RIN *omega* exită doar o singură cale între orice intrare şi orice ieşire. Prin urmare pentru o mapare dată a unui set de intrări la un set de ieşiri, algoritmul va produce un set unic de căi de comunicație pentru a realiza această mapare. Există un potențial conflictual deoarece o anumită pereche de astfel de căi poate să intre în conflict cu alta, deoarece se solicită simultan două stări diferite ale aceluiaşi comutator.

De exemplu maparea intrărilor (0, 4) la ieşirile (4, 7) va produce un astfel de conflict.

Formalizând, o RIN *omega* nu poate produce toate mapările posibile de la intrare la ieşire. În literatură [Hwan 93] se citează câteva clase de mapări ce pot fi realizate cu RIN *omega*.

În fig. 2.36 se prezintă situația conflictuală menționată anterior.

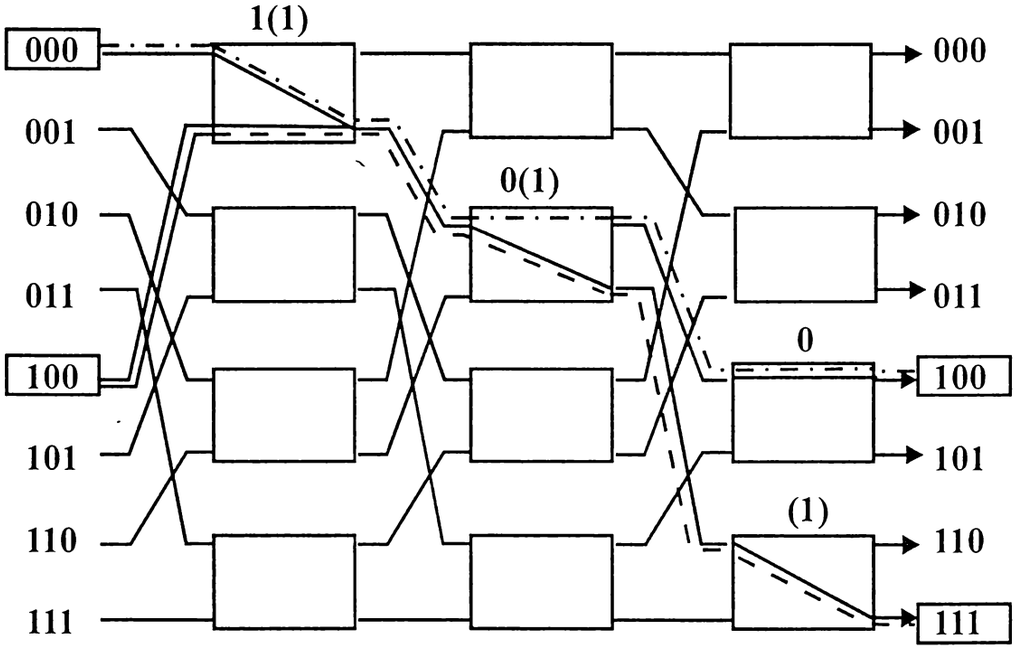


Fig. 2.36 Situația conflictuală la RIN *omega* pentru conexiunile $0 \rightarrow 4$ şi $4 \rightarrow 7$.

RIN *delta*

O RIN *delta* poate conecta $N=a^n$ intrări la $M=b^n$ ieşiri prin n niveluri de comutatoare *crossbar* $a \times b$. *Reţeaua cu amestec generalizat (generalized shuffle network)* (GSN) este capabilă să conecteze orice $M = m_1 * m_2 * \cdots * m_r$ intrări la $N = n_1 * n_2 * \cdots * n_r$ ieşiri prin intermediul a r niveluri de comutatoare. Al i-lea nivel utilizează $m_i \times n_i$ comutatoare crossbar şi este precedat de o GSN care în esenţă este un supraset a interconectărilor *omega* şi *delta*.

Aceasta este cea mai generalizată versiune a unei RIN–MN ce permite talii diferite în ceea ce priveşte numărul de intrări şi ieşiri. Toate celelalte RIN dinamice pot fi obţinute din această configuraţie alegând în mod corespunzător m_i şi n_i. De exemplu când $m_i = a$ şi $n_i = b$ pentru toţi i. rezultă o RIN *delta*. Când $m_i = n_i = 2$ pentru toţi i, rezultă o RIN *omega*. Dacă r = 1 se obţine RIN *crossbar*.

Când $M = M * 1$ şi $N = 1 * N$ se obţine o *magistrală partajată*.

În fig. 2.37 se prezintă structura unei RIN *delta* particularizată pentru n = 2.

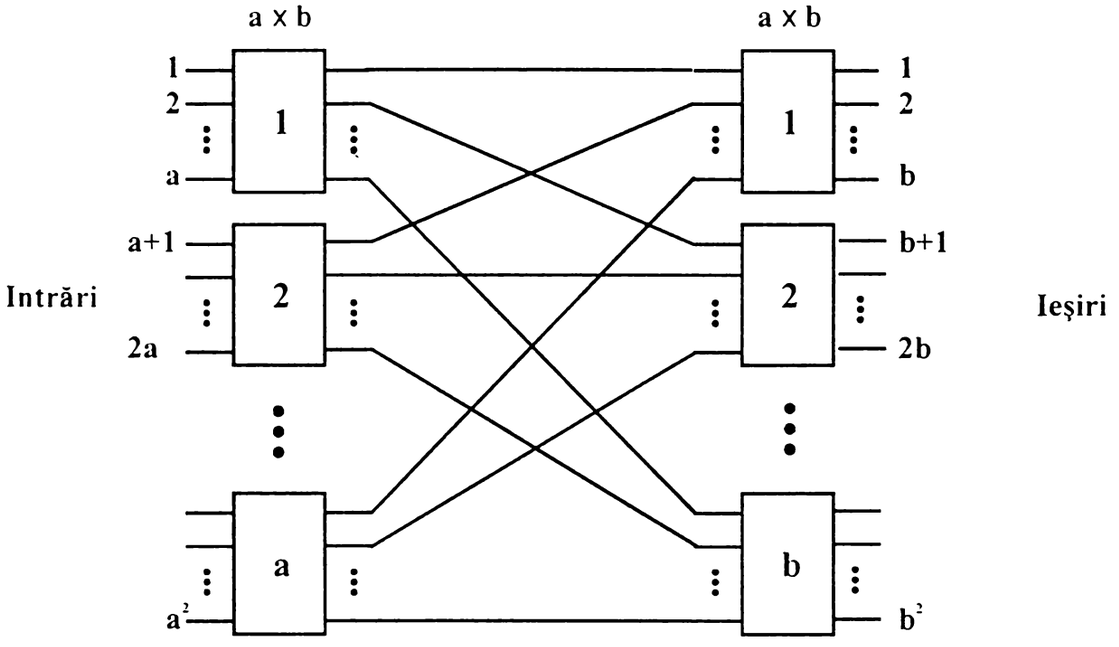


Fig. 2.37 RIN *delta* pentru n = 2

Un dezavantaj serios al RIN – MN îl constituie existenţa unei singure căi de date între o intrare şi o ieşire. A fost necesar să se incorporeze resurse ce să facă RIN–MN tolerantă la defecte. Ele vor fi analizate ulterior.

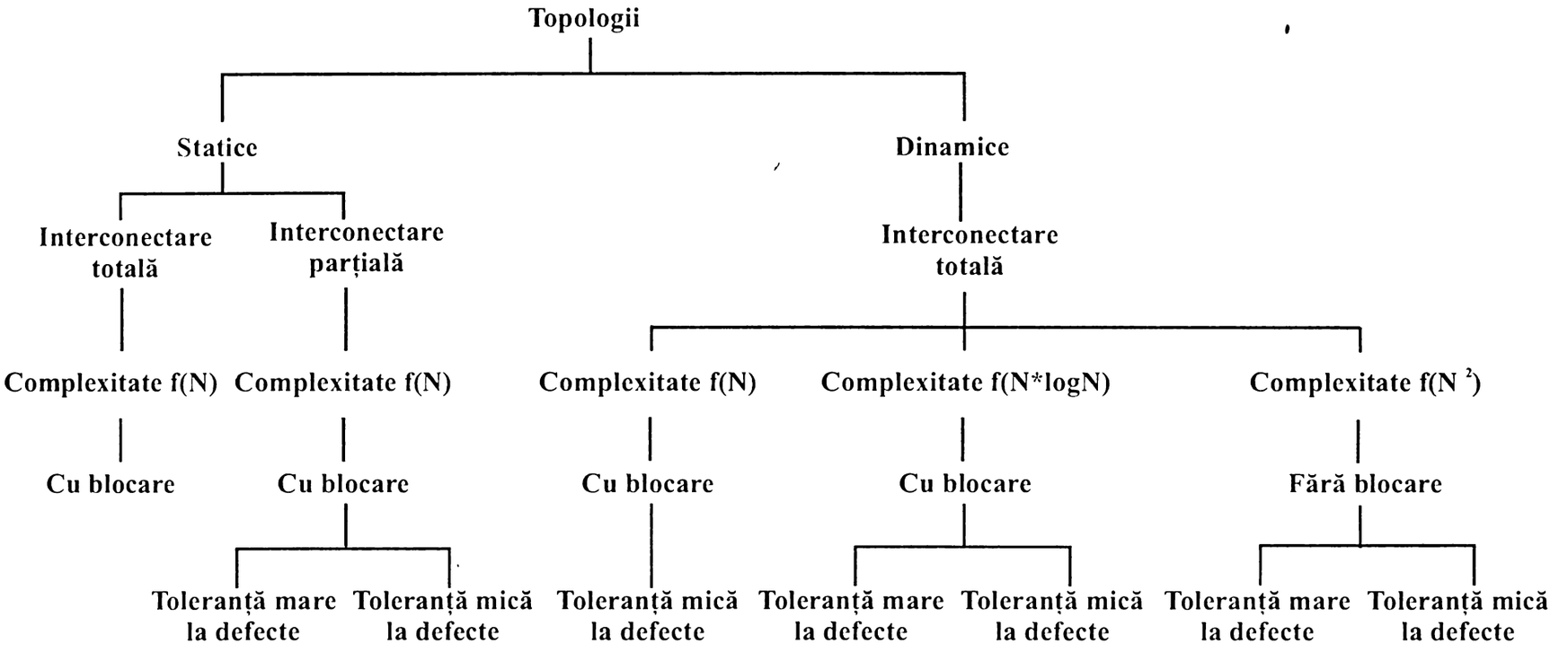


Fig. 2.38 Clasificarea topologiilor RIN

2.5 MECANISME DE SINCRONIZARE A PROCESELOR LA ARHITECTURI DIN CLASA MIMD

Arhitecturile MIMD trebuie să conțină capabilități pentru facilitarea comunicării între procese. O axiomă fundamentală ce guvernează problematica comunicării între procese este următoarea:

- *Nu se poate face nici o presupunere referitoare la vitezele relative ale proceselor*

Prin urmare mecanismele de sincronizare sunt programate în interiorul proceselor şi nu sunt impuse de procesoare.

Problematica comunicării între procese se reduce astfel la problematica sincronizării proceselor.

Interacțiunea dintre procese paralele se manifestă în sincronizarea lor în scopul de-a asigura:

1. *cooperarea* proceselor pentru a facilita comanda secvenței ce asigură rezultate corecte
2. *competiția* proceselor pentru accesul la resursele partajate prin furnizarea comenzii accesului pentru acestea

În literatură [Dasg 89], [Mold 93] se analizează diferite mecanisme de sincronizare pentru arhitecturi MIMD-T şi MIMD-L, *în extenso* pentru SMP - MP şi respectiv SMP - TM - multicalculatoare.

Tradițional aceste mecanisme se bazează pe utilizarea:

- *variabilelor partajate* - utilizate la SMP - MP;
- *transferurilor de mesaje* - utilizate la SMP - TM.

2.5.1 Moduri de sincronizare

În fig.2.39 se prezintă situația în care două procesoare P_i şi P_j execută procesele P_1 şi P_2. P_1 şi P_2 sunt două procese independente ce sunt procesate în paralel. În anumite puncte ale execuției lor, atât P_1 cât şi P_2 solicită modificarea structurii comune de date D rezidentă în memoria comună M.

Pentru a preveni apariția *nedeterminării funcționale* - situație în care D este modificată de o manieră incontrolabilă - este necesară tratarea segmentelor de cod CS_1 şi CS_2 ca fiind *regiuni critice*.

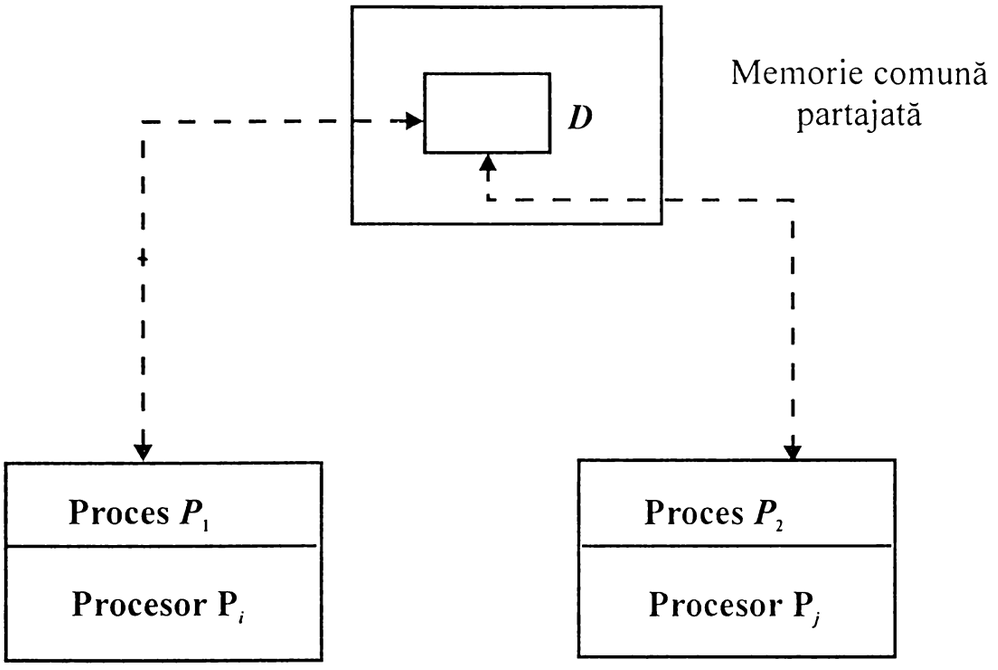


Fig. 2.39 Procese partajând o structură de date D

O *regiune critică* este o secțiune de cod în cadrul unui sistem concurent ce este astfel controlată încât la un moment dat doar un proces poate intra în ea.

Regiunile critice trebuie tratate ca fiind unități *indivizibile* iar modificarea structurilor de date se face într-o manieră de *excludere mutuală*.

În 1968 Dijkstra a tratat sistematic problemele sincronizării în regiunile critice devenind astfel un clasic.

Actualmente se preferă utilizarea termenului de *secțiune critică* în locul celui de *regiune critică*.

Implementarea celor două cerințe simultan obligatorii - *indivizibilitatea* şi *excluderea mutuală* - pentru o secțiune critică se face prin intermediul unui mecanism de *blocare* format din operații indivizibile (atomice) şi o "cheie" *k* ce poate fi *fanion*, *semafor* etc.

În cazul prezentat ca şi exemplu, structurii de date *D* i se asociază "cheia" *k* şi operațiile atomice LOCK (*k*) şi UNLOCK (*k*) ce poziționează *k* pe 1 respectiv pe 0.

Examinarea şi poziționarea lui *k* reprezintă o operație atomică în sensul că ea se efectuează pe durata unui singur impuls de orologiu şi nu poate fi întreruptă.

Structura unei astfel de operații atomice este:

Loop: [**if** $k = 0$ **then** $k = 1$ **else**] **go to** Loop;

unde [ ] indică indivizibilitatea.

Prin urmare un proces poate să execute LOCK (k) atunci când este în starea deblocată, adică când $k = 0$. În final, după efectuarea lui LOCK (k), $k = 1$ şi toate celelalte procese sunt excluse de la efectuarea acestei operaţii. Ele trebuie să aştepte într-o stare de *busy waiting*. Operaţia UNLOCK (k) efectuează $k = 0$.

Secţiunile critice în P_1 şi P_2 pot fi implementate astfel:

P_1:	P_2: . . . . .
LOCK (k)	LOCK (k)
secţiune critică CS_1	secţiune critică CS_2
UNLOCK (k)	UNLOCK (k)
. . . . .	. . . . .

P_1 şi P_2 au fost sincronizate în CS_1 şi CS_2 într-o manieră de excludere mutuală.

În situaţia *sincronizării condiţiei* se utilizează aceleaşi mecanisme de blocare ca şi la excluderea mutuală. Se asociază două chei, k_1 şi k_2, structurii de date D. P_1 deblochează k_2 pentru a semnala lui P_2 că a depus o nouă valoare în D în timp ce P_2 deblochează k_1 pentru a informa P_1 că a consumat ultima valoare plasată în D. Presupunând că k_1 şi k_2 sunt iniţial poziţionaţi pe 0 şi 1, segmentele relevante de cod în P_1 şi P_2 sunt:

P_1:	P_2: . . . . .
LOCK (k_1)	LOCK (k_2)
produce valoare şi pune în D	consumă valoare din D
UNLOCK (k_2)	UNLOCK (k_1)
. . . . .	. . . . .

Aceste mecanisme de blocare se realizează prin intermediul *primitivelor de sincronizare* ce utilizează variabile partajate.

În general se face distincţie între două mecanisme diferite de sincronizare:

- *sincronizare la nivel de comandă* - se utilizează întreruperile procesorului şi comenzi START, RESET;
- *sincronizări la nivel de date* - utilizată ori de câte ori una sau mai multe variabile dintr-un task sunt partajate;

2.5.2 Sincronizarea bazată pe variabile partajate

Implementarea operaţiilor LOCK/UNLOCK trebuie să îndeplinească două condiţii:

1. Operaţiile să fie suficient de primitive ca ele însuşi să fie atomice;
2. Să se cunoască condiţia de deblocare a procesului blocat.

Primitivele de sincronizare sunt:

- **TEST - AND - SET (TAS)**
- **FETCH - AND - ADD (FAA)**
- **COMPARE - AND - SWAP (CAS)**

***Primitiva* TEST - AND - SET (TAS)**

Este utilizată pentru coordonarea sau sincronizarea unor procese cât şi pentru crearea altora noi. TAS asigură excluderea mutuală prin executarea unei instrucţii ca şi o operaţie neîntreruptibilă, adică pe durata unui singur impuls de ceas.

Sintaxa este:

C: = TEST - AND - SET (*k*)

unde:

k - variabilă de memorie

C - variabilă locală de procesor

Semantica este:

TEST - AND - SET (*k*)

{*temp* ← *k*; *k* ←1; RETURN *k*; }

Acces la variabila partajată C

RESET (*k*)

{ *k* ←1; }

Operaţia LOCK poate fi acum implementată ca şi:

LOCK (*k*) : **repeat** C: = TEST - AND - SET (*k*) **until** C = 0

iar UNLOCK are forma:

UNLOCK (*k*) : *k* : = 0

La IBM 360/370 această operaţie se regăseşte sub forma instrucţiei TS iar la i386 ca şi TAS.

Inconvenientele majore ale primitivei TAS sunt:

- procesorul P_i în care P_1 aşteaptă un LOCK se găseşte în starea *busy-waiting* în care nu efectuează nimic;

- fiecare procesor aflat în starea *busy-waiting* consumă cicluri de memorie utilizabile mai eficient în alt mod.

***Primitiva de tipul* FETCH - AND - ADD (FAA)**

Scopul operațiilor de sincronizare este de a *serializa* două sau mai multe segmente de cod paralele implicând o structură de date partajate *D*.

În timp ce FAA se bazează pe ideea utilizată în TAS, modul ei de operare este diferit. Ca răspuns la *n* accese simultane, FAA acționează dând fiecărui proces un număr unic şi apoi le extrage serial.

Se obține astfel un anumit grad de simultaneitate a acceselor la o variabilă partajată în timp ce se păstrează cerința de serializare.

Principiul *serializării* este: dacă *x* reprezintă o variabilă partajată şi mai multe operații FAA adresează simultan pe *x*, efectul acestor operații este exact acela ce s-ar fi obținut dacă aceste operații ar fi apărut într-o ordine serială nespecificată.

Forma operației FAA este:

FETCH - AND - ADD (x,e)

unde:

x - variabila partajată

e - expresie întreagă

Semantica este:

FETCH - AND - ADD (x,e)

$$\{temp \leftarrow x,\ x \leftarrow temp + e;$$
$$\text{RETURN } temp\ \}$$

Execuția FAA provoacă ca valoarea lui x să fie returnată iar x să fie înlocuit cu $x + e$ ca şi o singură operație indivizibilă.

Cunoscând că x este variabila partajată şi că P_i şi P_j execută simultan P_1 şi P_2, atunci:

$$P_i : temp_i \leftarrow \text{FETCH - AND - ADD } (x,e_i)$$

$$P_j : temp_j \leftarrow \text{FETCH - AND - ADD } (x,e_j)$$

apoi fie:

$$temp_i \leftarrow x \quad \text{sau} \quad temp_i \leftarrow x + e_j$$

$$temp_j \leftarrow x + e_j \qquad temp_j \leftarrow x$$

În orice caz valoarea lui x la sfârşit devine $x + e_i + e_j$.

FETCH - AND - ADD (x, *true*) este echivalent cu TEST - AND - SET (x)

{*temp* ← x,
x ← *true*
RETURN *temp* }

În literatură [Dasg89], se prezintă o generalizare a primitivei FAA. Ea este:

FETCH - AND - OP (x,e)

unde:

x - variabila partajată

e - expresie compatibilă ca li tip cu x

Această operație returnează valoarea lui x şi în acelaşi timp o înlocuieşte cu valoarea OP (x,e).

Acum primitiva:

C: TEST – AND – SET (k)

poate fi realizată de instrucția:

C: FETCH – AND – OR (k, TRUE)

unde (k) este o variabilă boleană.

Pentru a reliefa avantajul primitivei FAO față de TAS considerăm două procese concurente P_1 şi P_2 ce operează excluderea mutuală asupra unei variabile utilizând secțiuni critice.

Utilizând TAS, avem:

versiunea 1 (inițial, k = 0)

P_1: . . . . . .

repeat C_1: TEST – AND – SET (k) **until** C_1 = 0 (“LOCK”)

secțiunea critică CS_1

k: = 0 (“UNLOCK”)

P_2: . . . . . .

repeat C_2: TEST – AND – SET (k) **until** C_2 = 0 (“LOCK”)

secțiunea critică CS_2

k: = 0 (“UNLOCK”)

Utilizând FAO, avem:

versiunea 2 (inițial, k = FALSE)

P_1: . . . . . .

repeat C_1: FETCH – AND – OR (k, TRUE) **until** C_1 = FALSE

secțiunea critică CS_1

k: = FALSE

P_2: . . . . . .

repeat C_2: FETCH – AND – OR (k, TRUE) **until** C_2 = FALSE

secțiunea critică CS_2

k: = FALSE

În versiunea 2, dacă se încearcă execuții simultane a două FAO, proprietatea de serializare va fi garantată fie pentru:

C_1 = FALSE , C_2 = TRUE

în care caz P_1 va intra în CS_1, sau:

C_1 = TRUE , C_2 = FALSE

în care caz P_2 va intra în CS_2

În oricare din cazuri k = TRUE.

Ceea ce este important, cele două actualizări ale lui k vor fi combinate într-un singur acces la memorie.

În contrast modificarea lui k în varianta 1 se face de către două operații TAS în mod serial.

***Primitiva* COMPARE – AND – SWAP (CAS)**

CAS este o primitivă de sincronizare ce blochează o variabilă partajată, o actualizează şi apoi o deblochează.

Semantica sa este:

COMPARE – AND- SWAP (r-old, r-new, addr)

{temp ← m (addr); if temp = r – old

then {m (addr) ← r – new; z = ← 1; }

else {r – old ← m (addr); z = 0; }

}

Variabila partajată *m (addr)* reprezintă conținutul unei locații de memorie denumite *addr*, şi *r* – old şi *r* – new sunt două registre utilizate pentru a înmagazina valoarea curentă şi noua valoare a variabilei partajate. Succesul operației CAS este indicat de fanionul Z. CAS este mult mai puternică decât TAS deoarece după ce noua valoare a variabilei partajate este disponibilă, într-o singură operație neîntreruptibilă această primitivă de sincronizare readuce variabila partajată, verifică dacă valoarea ei a fost schimbată, şi dacă nu, efectuează actualizarea.

Metoda numărătorului

Sincronizarea prin *metoda numărătorului (contorului)* este utilizată pentru un număr mai mare procese. De exemplu patru procese P_1, P_2, P_3 şi P_4 accesează o variabilă partajată. Un contor este asociat variabilei partajate. Fiecare proces P_n poate să acceseze locația variabilei partajate doar când contorul conține acelaşi număr ca şi cheia asigurată acelui proces.

Semafoare

Recunoscând dezavantajul că un procesor ce execută LOCK se găseşte în starea *busy-waiting*, Dijkstra (1968) a pròpus două noi operații indivizibile notate cu P şi V, definite asupra unei variabile speciale denumită *semafor*.

Un semafor este o variabilă întreagă ne–negativă. Dându-se un semafor *S*, operațiile P(*S*) şi V(*S*) sunt definite astfel:

P(*S*) : **if** $S > 0$

then $S := S - 1$

else blochează procesorul ce execută P(*S*) şi plasează-l într-un "şir de aşteptare" asociat cu *S*;

Alocă procesorul la alt proces din "şirul operațional"

V(*S*) : $S := S + 1$

if există oricare alte procese în "şirul de aşteptare" asociat cu *S*

then selectează un astfel de proces din "şirul de aşteptare" şi plasează-l în "şirul operațional"

Cu alte cuvinte, P(*S*) va bloca procesul apelant atâta timp cât *S* are o valoare pozitivă.

V(*S*) va face ca un proces ce a fost anterior blocat pe acest semafor să fie plasat în starea *ready* astfel încât să fie eventual alocat unui procesor disponibil. Astfel condiția *busy-wait* este evitată.

Utilizând un semafor, problema excluderii mutuale poate fi rezolvată astfel. Presupunem că S = 1.

Atunci:

P_1:	P_2: . . . . .
P(*S*)	P(*S*)
secțiunea critică CS_1	secțiunea critică CS_1
V(*S*)	V(*S*)
. . . . .	

Similar, problema sincronizării condiției descrisă anterior poate fi soluționată utilizând două semafoare S_1 și S_2. Presupunem că $S_1 = 1$ și $S_2 = 0$. Atunci:

P_1:	P_2: . . . . .
P(S_1)	P(S_2)
produce valoarea și depozitează în D	consumă valoare în D
V(S_2)	V(S_1)
. . . . .	. . . . .

Un semafor el însuși este o variabilă partajată, și P și V ca operații asupra lui *S* trebuie să satisfacă proprietatea de excludere mutuală. Prin urmare P și V trebuie să fie programate ca și secțiuni critice utilizând primitiva TAS. Deoarece acestea vor fi secțiuni critice foarte mici, timpul de busy – wait va fi mic.

2.5.3 Sincronizarea bazată pe transfer de mesaje

Sistemele multiprocesor cu transfer de mesaje - SMP-TM - sunt arhitecturi aparținând taxonului MIMD-L.

Reamintim că din punct de vedere structural un SMP -TM este format din perechi procesor/memorie - PM - interconectate prin RIN. Comunicarea datelor se face prin *transfer de mesaje* și nu prin variabile partajate, lungimea mesajelor diferă, dar în mod uzual fiecare mesaj constă dintr-un număr de pachete de dimensiune fixă. Comunicarea între calculatoare urmează un protocol de comunicare predeterminat. Conexiunile între

calculatoare pot fi seriale de mare viteză sau paralele. Prin urmare nodul activ într-un SMP - TM este un calculator.

În fig. 2.40 se prezintă o structură simplă formate din 8 PM conectate sub forma unui *cub boolean*, o structură particular a *hipercubului*.

Procesele sunt distribuite printre PM -uri. Fiecare proces are acces direct la memoria sa locală.

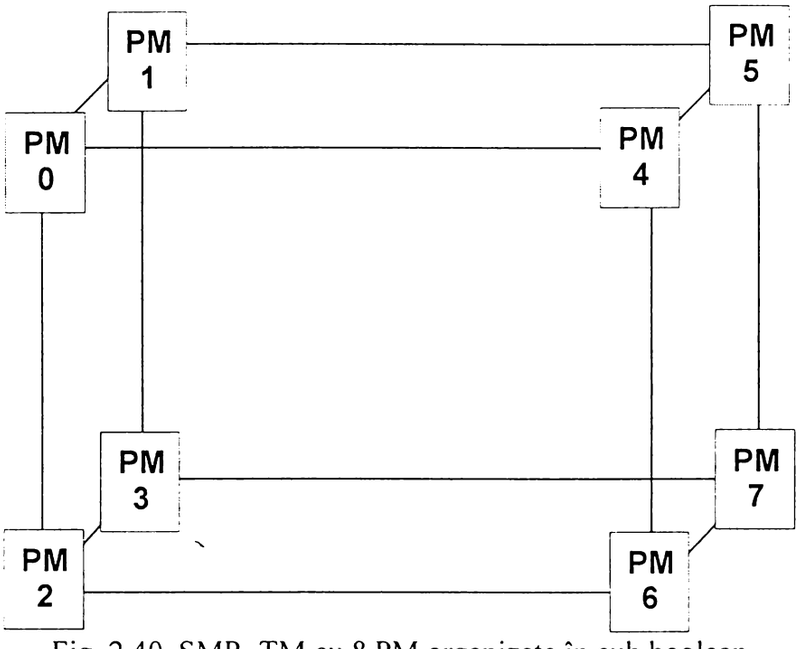


Fig. 2.40 SMP- TM cu 8 PM organizate în cub boolean

Sincronizarea proceselor se bazează pe principiul *transferului de mesaje*.

Operațiile primitive pentru transferul mesajelor sunt în mod uzual de tip SEND/RECEIVE.

Un proces trimite un mesaj executând o operație de forma:

SEND expresie TO procesul de destinație

ce face ca mesajul - valoarea expresiei - să fie transmis spre procesul de destinație.

Un mesaj este recepționat de un proces când se execută:

RECEIVE variabilă FROM procesul sursă

ce face ca mesajul din procesul sursă să fie recepționat în variabilă.

Când un proces *PS* emite şi altul *PR* recepționează se stabileşte un *canal de comunicație*.

Cele mai restrictive canale de comunicație sunt acelea în care procesele au o nominalizare explicită de destinație / sursă. Astfel de protocoale se numesc cu *nominalizare directă*.

În SMP -TM bazate pe protocoale de tip *nominalizare directă*, mijloacele pentru sincronizarea proceselor emitente şi a celor de receptare pot fi afectate în mai multe moduri.

În sistemele *sincrone* nu există buffere asociate cu canalul de comunicaţie. Prin urmare executarea unui SEND de către procesorul *PS* este întotdeauna întârziată până când *PR* execută RECEIVE. *PS* este sursa iar *PR* destinaţia mesajului.

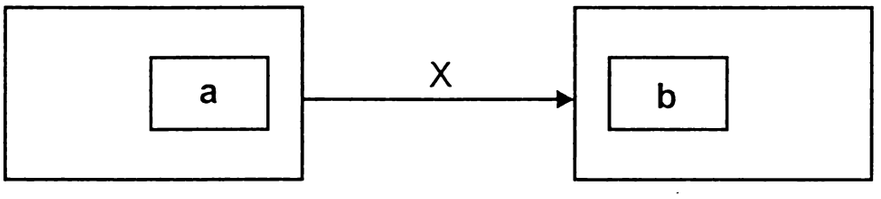
În sistemele *asincrone* se asociază buffere canalelor de comunicaţie. Operaţia SEND este întârziată doar când bufferul este plin şi operaţia RECEIVE va fi întârziată doar când bufferul este gol.

Un model de transfer de mesaj este cel utilizat în *transputer* utilizând limbajul de programare Occam. Acest limbaj permite programarea concurentă.

În exemplul anterior prezentat în fig. 2.39 problema producător - consumator consta în faptul că P_1 produce un flux de valori ce sunt consumate de P_2 în ordinea în care valorile sunt produse.

În modelul cu variabilă partajată, valorile produse au fost depozitate în structura partajată *D* care de asemenea servea ca sursă de valori pentru P_2. Accesul corect la *D* presupune utilizarea *sincronizării de condiţie*.

Utilizând Occam, aceeaşi problemă poate fi rezolvată astfel. Două *procese* evoluează în paralel şi comunică prin *canalul X*. (fig. 2.41).



procesul producător **procesul consumator**

Fig. 2.41 Structura proceselor Occam şi canalul lor de comunicare

Compoziţia întregului proces paralel este indicată sub forma:

par

[componentă

[componentă

unde componentele sunt indicate în raport cu cuvântul cheie *par*. Fiecare din cele două componente ale procesului paralel este un proces secvenşial iterativ, indicat de forma textuală:

```
chan x
  par
    while true
      var a
      seq
        calculează valoarea şi plasează în a
        x ! a
    while true
      var b
      seq
        x ? b
        consumă valoarea în b
```

Procesul producător execută o instrucție de ieşire:

x ! a

care transmite variabilă locală îm x_j în timp ce procesul consumator execută:

x ? b

ce cauzează ca valoarea din calcul x să fie citită în b, variabila locală.

Instrucțiile de intrare/ieşire sunt două din cele trei primitive din Occam.

Sincronizarea este obținute prin faptul că o instrucție de intrare în care x este implicat, nu se termină decât atunci când o instrucție de ieşire, ce implică acelaşi canal, nu este execțutată.

2.6 CONCLUZII

Din analiza arhitecturilor sistemelor multiprocesor se pot desprinde câteva concluzii referitoare atât la ariile de aplicabilitate cât şi la tendințele de evoluție ulterioare.

SMP-MP *simetrice* reprezintă o extensie naturală a stațiilor de lucru şi a calculatoarelor personale. O aplicație secvențială poate fi rulată în totalitate nemodificată şi un beneficiu în performanță se obține prin utilizarea avantajului de a avea o mare cantitate de memorie principală partajată şi capacități I/E tipic disponibile în astfel de maşini. Aplicațiile paralele sunt de asemenea relativ uşor de implementat deoarece toate datele partajate sunt direct accesibile de la toate procesoarele utilizând instrucțiuni obişnuite de tip LOAD şi

STORE. O paralelizare graduală este posibilă prin paralelizarea selectivă în mod intensiv a unor aplicații secvențiale. Pentru aplicații multiprogramate un avantaj substanțial îl reprezintă granularitatea fină în care resursele pot fi partajate printre aplicațiile de proces şi de către sistemul de operare.

Deoarece procesorul, sistemul de memorie, circuitele integrate şi tehnologiile de asamblare continuă să se amelioreze rapid, sarcina proiectantului se focalizează pe găsirea unor structuri performante. SMP cu număr redus de procesoare vor continua să fie importante din cel puțin trei aspecte. Primul se referă la faptul că ele oferă o combinație atractivă cost-performanță. Al doilea se referă la faptul că astăzi microprocesoarele sunt proiectate să fie pregătite pentru includerea în structuri multiprocesor, dispunând de tot setul de semnale necesare pentru interfațare. A treia rațiune se referă la faptul că tehnologia software esențială pentru maşinile paralele – compilatoare, sisteme de operare, limbaje de programare – se dezvoltă rapid pentru astfel de SMP-MP de scală redusă.

Aspectele referitoare la proiectare sunt importante la nivelele moderate de paralelism. Desigur, alternativele de proiectare optimală se pot schimba. De exemplu, deşi nu sunt foarte populare, este posibil ca memoriile cache partajate la un anumit nivel de ierarhie să devină foarte atractive atunci când tehnologia de încapsulare a modulelor multichip va deveni ieftină sau când se vor putea încapsula procesoare multiple într-o singură capsulă.

Deşi este un mecanism puternic, o interconectare de tip magistrală partajată are în mod clar limitări în ceea ce priveşte lățimea de bandă pe măsură ce numărul de procesoare sau viteza procesorului cresc. Oricum, soluția generală o reprezintă construcția maşinilor coerente din punct de vedere cache, scalabile, în care memoria să fie în mod fizic distribuită printre noduri şi în care să se utilizeze o interconectare scalabilă.

Aceste abordări trebuie corelate cu protocoalele de menținere a coerenței ce nu se bazează pe monitorizare. În proiectarea arhitecturilor cu memorie partajată coerente din punct de vedere cache, indiferent de interconectarea utilizată, plasarea unor capacități de memorare în astfel de RIN reprezintă o opțiune majoră ce ameliorează performanța. Indiferent însă de opțiunea de proiectare, aceste proiecte bazate pe magistrală oferă blocul constructiv de bază pentru arhitecturi paralele pe scară largă.

O evoluție firească o reprezintă arhitecturile paralele pe scară largă ce sunt construite cu noduri de utilizare generală ce au în componență o ierarhie de memorie locală completă

augmentată de o interfață de comunicare pretabilă la rețele scalabile. Oricum, o gamă largă de opțiuni de proiectare este disponibilă pentru interfața de asistare a comunicării. Proiectarea este puternic influențată de locul în care interfețele de asigurare a comunicării sunt plasate în arhitectura nodului: la nivel de procesor, la nivel de controller al memoriei cache, la nivel de magistrală a memoriei sau al magistralei de I/E. De asemenea o puternică influență o reprezintă *arhitectura de comunicare* şi *modelul de programare*. Modelele de programare sunt implementate în maşinile paralele de talie mare utilizând protocoale construite pe baza tranzacțiilor primitive de la nivel de rețea. Riscurile în implementarea unui astfel de protocol se referă la numărul mare de tranzacții ce trebuie să fie gestionate simultan şi la arbitrarea globală care în general nu este disponibilă.

Ca şi un rezultat se constată evoluția a cel puțin două direcții în care arhitecturile paralele pe scală largă evoluează. Maşini orientate în general pe conceptele de transfer de mesaj şi apariția unor grupuri de procesoare sau de memorii reprezintă o direcție. Cealaltă direcție este reprezentată de maşini care integrează puternic rețeaua de comunicare în sistemul de memorie pentru a furniza accese coerente din punct de vedere cache la un spațiu fizic de adresă partajată. Desigur, riscurile în promovarea unor mecanisme sofisticate de menținere a coerenței se referă la limitările în ceea ce priveşte lățimea de bandă scalabilă şi latența redusă.

Sistemele scalabile ce suportă un spațiu de adresă partajată coerentă reprezintă o parte importantă a mediului de multiprocesare deoarece ele combină uşurința programării unui model de programare cu spațiu de adresă partajată cu avantajele în ceea ce priveşte scalabilitatea unei memorii distribuite şi a interconectării aferente. Suportul hardware pentru asigurarea coerenței cache asigură evoluția unor protocoale bazate pe director atât în situația când sistmele se bazează pe memorie cât şi pe memoria cache.

În timp ce suportul hardware pentru asigurarea coerenței cache reprezintă o opțiune de proiectare costisitoare, apariția standardelor şi a faptului că microprocesoarele însăşi oferă suport pentru coerența cache, ameliorează această opțiune. Odată ce protocolul de menținere a coerenței este disponibil la nivel de microprocesor, proiectanții pot dezvolta protocoale la nivel de multiprocesor şi arhitecturi de comunicație încă de la început.

3. ANALIZA DE PERFORMANȚĂ A SISTEMELOR DE CALCUL DIN CLASA MIMD

3.1 ANALIZA DE PERFORMANȚĂ LA RIN

3.1.1 Terminologie de bază utilizată în evaluarea performanțelor RIN

În evaluarea performanțelor RIN se utilizează mai mulți parametrii derivați din metrica general aplicabilă în determinarea dependabilității sistemelor de calcul.

În analiza sistemelor *sincrone* se utilizează următorii parametri:

- *Lățimea de bandă a memoriei (BW)* - numărul mediu de module active de memorie într-un ciclu de transfer al unei RIN. Termenul "activ" înseamnă că un procesor a efectuat cu succes o operație referitoare la memorie (citire sau scriere) în modulul definit ca fiind activ. *BW* se utilizează în analizarea RIN *sincrone*.

În analiza sistemelor *asincrone*, se utilizează următorul parametru:

- *Capacitatea de trecere (Thr)* - numărul mediu de pachete furnizate de către RIN în unitatea de timp.

  În RIN utilizate în SMP - MP, *Thr* reprezintă numărul mediu de accese la memorie efectuate în unitatea de timp.

În continuare definim:

- *Probabilitatea de acceptanță* (P_A) - raportul dintre lățimea de bandă prevăzută şi numărul aşteptat de cereri de acces generate per ciclu.

$$P_A = \frac{BW}{Np}$$

- *Utilizarea procesorului* (P_u) - valoarea prevăzută a procentajului din timpul în care un procesor este activ. Un procesor este "activ" atunci când îşi efectuează calculul intern fără accesarea memoriei principale.
- *Puterea de procesare* (P_p) - extensie a P_u, reprezentând suma utilizărilor procesoarelor împărțită de numărul de procesoare din sistem.

Rezumând:

$$P_u = \frac{BW}{N\lambda T} \qquad \text{- sisteme sincrone}$$

$$P_u = \frac{Thr}{\lambda}$$ - sisteme asincrone

unde:

N - nr. de procesoare

T - timpul afectat unei operații de citire/scriere

λ - rata de cerere de acces la memorie

Studiul performanței unui sistem de calcul se bazează pe diverse tehnici.

Modelul analitic reprezintă o tehnică des utilizată sub raportul costului.

Pentru a face un astfel de model practicabil se fac următoarele ipoteze de aproximare.

- *modelul de referință uniformă - MRU.* Se presupune că toate procesoarele sunt identice şi atunci când un procesor efectuează o solicitare de acces la memoria principală, cererea va fi direcționată la oricare din cele M module de memorie cu aceeaşi probabilitate $1/M$. Aceasta înseamnă că adresa de destinație a unei cereri de acces la memorie este uniform distribuită printre cele M module de memorie.

  Dacă sistemul de memorie este întrețesut această presupunere este aproximativ corectă.

  Dacă sistemul de memorie nu este întrețesut, atunci există o localizare de referință şi o abordare în care se defineşte o memorie ca fiind "favorită", este mai corectă.

- *Rata de cerere de acces la memorie* λ a unui procesor identifică cât de des un procesor accesează memoria principală. În mod indirect reflectă timpul mediu de execuție a unei instrucții.

  În sistemele sincrone, acest parametru poate fi specificat ca şi o probabilitate în care un procesor generează o cerere de acces la memorie la începutul unui ciclu.

  În sistemele asincrone, o cerere de acces la memorie ar putea fi generată în orice moment deoarece nu există un ceas central. Se poate imagina o evoluție exponențială a timpului în sensul că durata dintre execuția unei cereri şi următoarea cerere de acces la memoria principală reprezintă o variabilă aleatoare distribuită exponențial.

- *Ipoteza independenței cererii de acces* - denumită aproximarea lui Strecker - este utilizată în analiza sistemelor sincrone.

  Se presupune că o cerere de acces la memorie generată într-un ciclu este independentă de cererile din ciclurile anterioare.

  În realitate însă o cerere de acces rejectată în ciclul anterior va fi relansată în ciclul curent.

TABEL I. Rezumatul caracteristicilor hardware ale celor trei RIN

	crossbar	**RIN - MN**	**Magistrală multiplă**
Nr. de comutatoare	$N \times M$	$N \log N$	$B \times (N+M)$
Încărcarea magistralelor	N	1	B
Nr. de fire	M	N	B
Nr.de arbitrii	M "1 din N"	$N \log N$ "1 din 2"	1 "B din M" M "1 din N"
Toleranța la defecte	Acceptabilă	Slabă	Bună

3.1.2 Analiza performanțelor magistralei unice

SMP - MP *cu magistrală unică* reprezintă cele mai simple arhitecturi de sisteme multiprocesor.

La aceste sisteme, RIN este constituită dintr-o magistrală unică, partajată în timp.

Procesoarele - module *master* - solicită accesul la memoria comună - resursă *slave* - şi apoi efectuează transferul de date.

Există două situații distincte în care se satisface o cerere de acces la memorie.

În primul caz, un singur procesor, la un moment dat, solicită şi obține accesul la un modul de memorie, după care se efectuează operația de citire sau scriere din /în memorie. Este cazul banal.

În al doilea caz, două sau mai multe procesoare solicită simultan acces la memorie. Este o situație conflictuală ce se rezolvă prin arbitrare, după un anumit algoritm de alocare a resursei solicitate procesorului declarat câştigător.

Reamintim că la SMP - MP cu *magistrală unică*, la un moment dat, un singur procesor operează pe magistrală cu modulul de memorie solicitat.

Evident că magistrala unică reprezintă resursa ce limitează drastic performanțele acestor sisteme.

Un procesor se poate afla în una din următoarele stări :

- *activ (active)* - procesorul lucrează cu memoria sa locală - ML;
- *în accesare (accesing)* - procesorul schimbă date cu alte procesoare prin operații de scriere în sau de citire din zona de memorie comună;
- *în coadă de aşteptare (queued)* - procesorul aşteaptă obținerea unui acces la MC.

Pentru a determina *încărcarea sistemului* (*workload*) în literatură [HP96], [Kak96] etc. se prezintă două abordări posibile : *deterministă* şi *probabilistică*.

În abordarea *deterministă*, încărcarea fiecărui procesor se consideră cunoscută şi pe această bază se poate dezvolta o secvență precisă a operațiilor efectuate de fiecare procesor pentru a caracteriza comportarea întregului sistem. Parametrii de performanță se pot evalua prin simulare, ceea ce limitează această tehnică la cazul concret aflat sub observare.

În abordarea *probabilistică* a sistemului, se dezvoltă un model statistic care încearcă să capteze esențialul în funcționarea SMP, fără a fi cunoscute detaliile individuale ale încărcării lui. Parametrii de încărcare se definesc ca şi variabile aleatoare cu o anumită distribuție.

Rezultatele furnizate au avantajul de-a reprezenta relații funcționale între parametrii de sistem şi performanță. De asemenea, un astfel de model stohastic permite studiul comparativ al diferitelor soluții arhitecturale cu referire la obiectivele de performanță impuse.

În cadrul modelelor de performanță se pot admite două moduri de funcționare ale procesoarelor : *asincronă* sau *sincronă*.

Uzual se utilizează abordarea de tip *sincron* care consideră sistemul la nivelul instrucțiilor, reprezentate ca operații sincrone pentru toate procesoarele. Axa timpului este divizată în intervale elementare - *sloturi*. (fig. 3.1).

La începutul unui slot procesoarele pot emite cereri de acces la memoria comună. Mecanismul de arbitrare selectează cererile ce pot fi onorate şi procesorul câştigător se conectează la modulele de memorie pe toată durata slotului.

Timpul de procesare al fiecărui procesor P este împărțit în :

- *timp activ* (t_a) - procesorul lucrează singur sau cu memoria sa locală;
- *timp de lucru cu memoria* (t_m) - procesorul, în urma unui acces reuşit, lucrează cu MC.

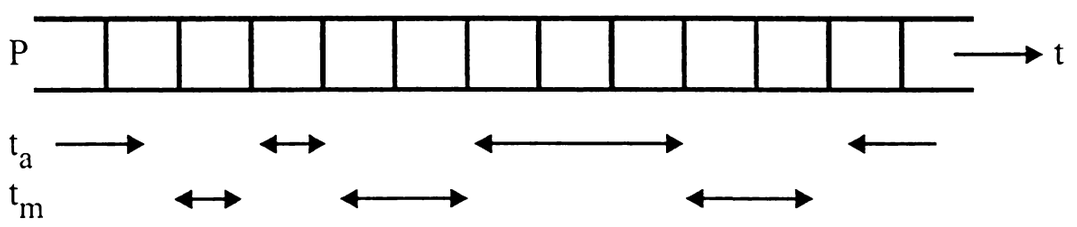


Fig. 3.1 Structura timpului de procesare

Ciclul de execuție al procesorului se studiază sub aspectul raportului dintre *timpul activ al procesorului* şi *timpul de citire/scriere* în MC.

Fie un SMP - MP *cu magistrală unică* ce conține N procesoare, P_1, P_2, ..., P_N, fiecare având propria memorie cache - MCH - şi toate conectate la o memorie comună MC formată din M blocuri intrețesute, M_1, M_2, ..., M_M.

Fie r probabilitatea ca un procesor oarecare P_i să solicite acces la MC la începutul unui slot.

Atunci Nr reprezintă numărul aşteptat de solicitări de acces la MC.

Dacă $r = 1$ atunci $t_a = 1$ iar pentru $r < 1$ acest timp va avea o distribuție geometrică.

BW reprezintă lățimea de bandă a subsistemului compus din o magistrală unică ce conectează M blocuri de memorie, adică numărul aşteptat de blocuri de memorie ocupate. Acesta reprezintă gradul de ocupare al memoriei.

Datorită conflictelor de acces există $BW < Nr$.

Probabilitatea ca să existe o solicitare de acces de la P_i către un anumit bloc M_j este r/M.

Probabilitatea ca să nu existe solicitări de la P_i către M_j este $1-r/M$ şi prin urmare probabilitatea ca nici unul dintre procesoare să solicite M_j la începutul unui slot este $(1 - r/M)^N$.

Fie E_j evenimentul ce constă în faptul că există cel puțin o solicitare pentru M_j.

Atunci :

$$\Pr[E_j] = 1 - \left(1 - \frac{r}{M}\right)^N = q \tag{3.1}$$

pentru orice i.

Lățimea de bandă va fi :

$$BW = 1 - \left(1 - \frac{r}{M}\right)^N = q \tag{3.2}$$

O rată ridicată a cererilor de acces duce la saturarea rapidă a magistralei.

Capacitatea de trecere Thr se poate exprima ca şi o funcție de numărul de instrucții executate pe secundă de către SMP.

Fie :

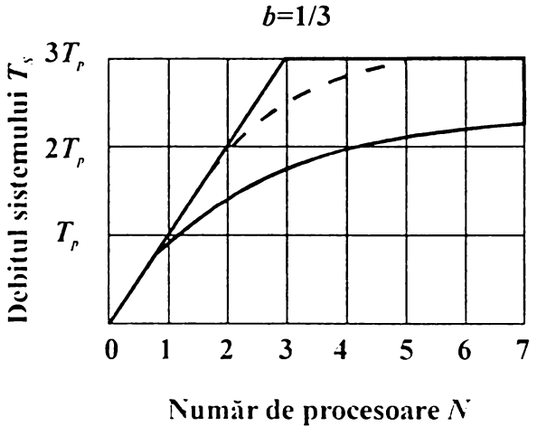
T_S - debitul SMP

T_p - debitul unui procesor P_i

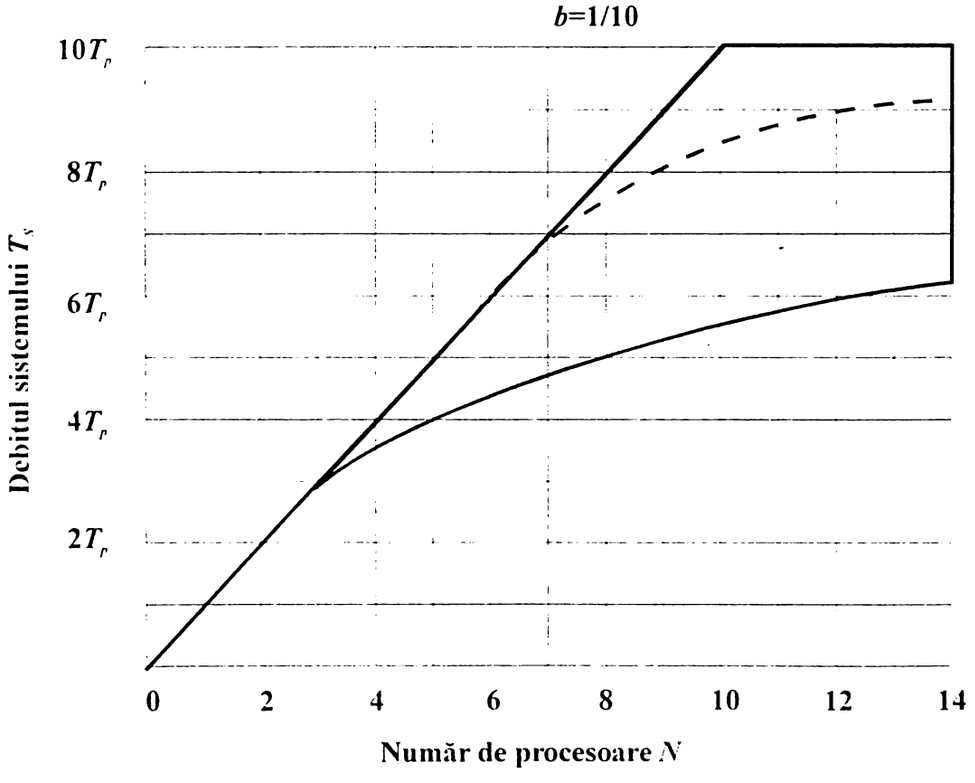
b - factorul de utilizare al magistralei ce exprimă numărul de cicluri de magistrală solicitate de fiecare procesor

În fig. 3.2 se prezintă variația funcției $T_S = f(N)$ pentru diferite valori ale lui b.

Se observă o ameliorare semnificativă a performanțelor atunci când b scade prin utilizarea memoriilor locale.



a)



b)

Fig. 3.2 Saturația magistralei

Dacă se consideră că W este numărul de procesoare aflate în aşteptare atunci se poate exprima :

$$W = \begin{cases} 0 & dacă \ N \le \frac{1}{b} \\ N - \frac{1}{b} & dacă \ N > \frac{1}{b} \end{cases} \qquad (3.3)$$

Atunci când la un SMP apare saturarea magistralei, traficul mare de pe magistrală va conduce la creşterea numărului de procesoare aflate în aşteptare şi prin urmare se pierde din eficienţa dobândită prin distribuirea sarcinilor la mai multe procesoare.
Curba caracteristică $W = f(N,b)$ este prezentată în fig. 3.3.

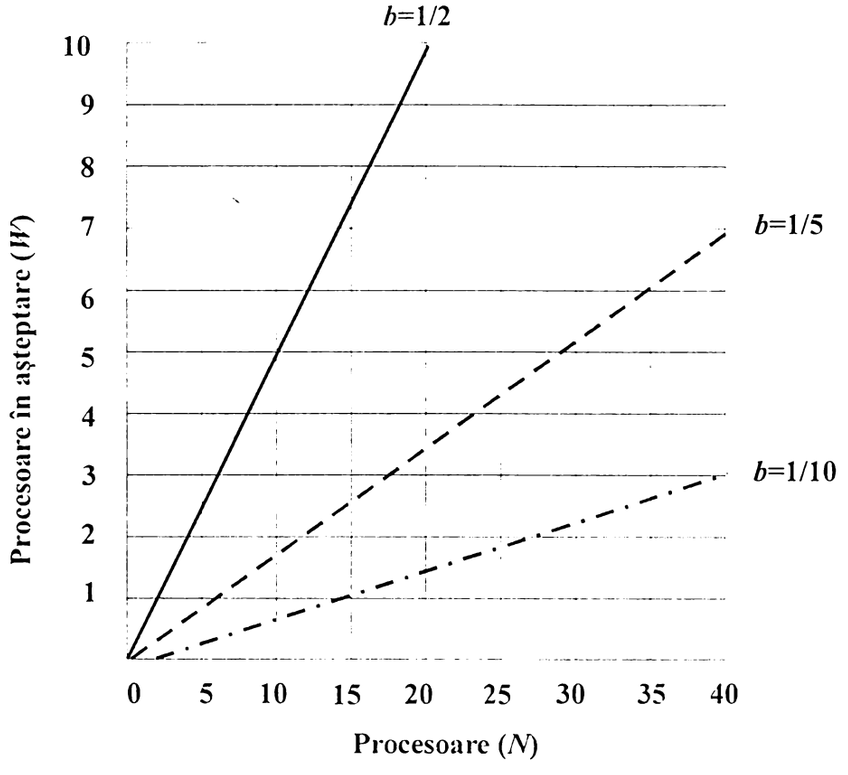


Fig. 3.3. Raportul între procesoarele în aşteptare şi cele active

3.1.3 Analiza performanţelor magistralei multiple.

În cazul SMP-MP o sursă majoră de limitare a performanţelor şi implicit a fiabilităţii şi a toleranţei de defecte o reprezintă magistrala unică.

Una dintre metodele de ameliorare a performanțelor o reprezintă creşterea capacității de transfer a magistralei.

O soluție luată în considerare [MHW87], [YB91] o reprezintă *multiplicarea numărului de magistrale* în scopul asigurării unei fiabilități superioare şi a unei *BW* mărite.

Arhitectura propusă reprezintă cazul conectării *complete* şi este prezentată în fig. 3.4.

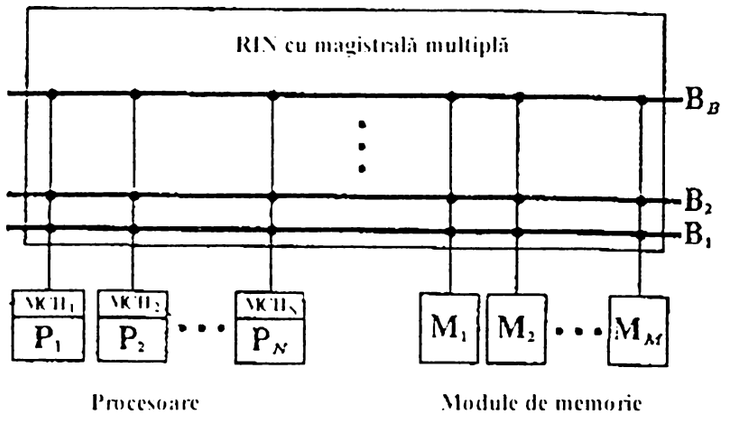
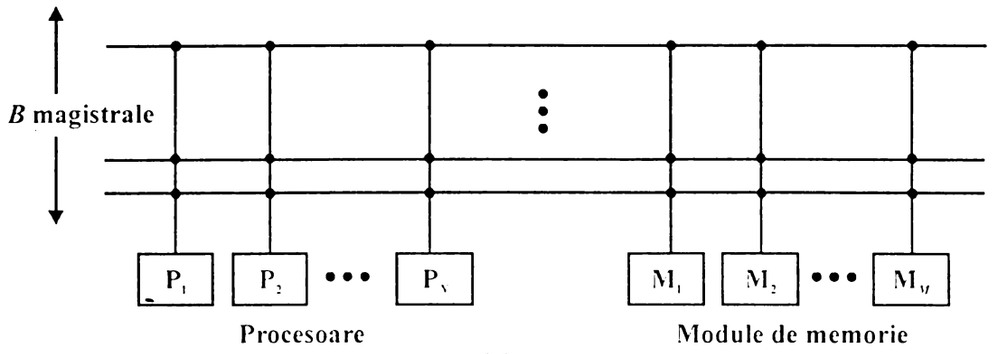


Fig. 3.4. Structura unui SMP-MP cu *magistrală multiplă*

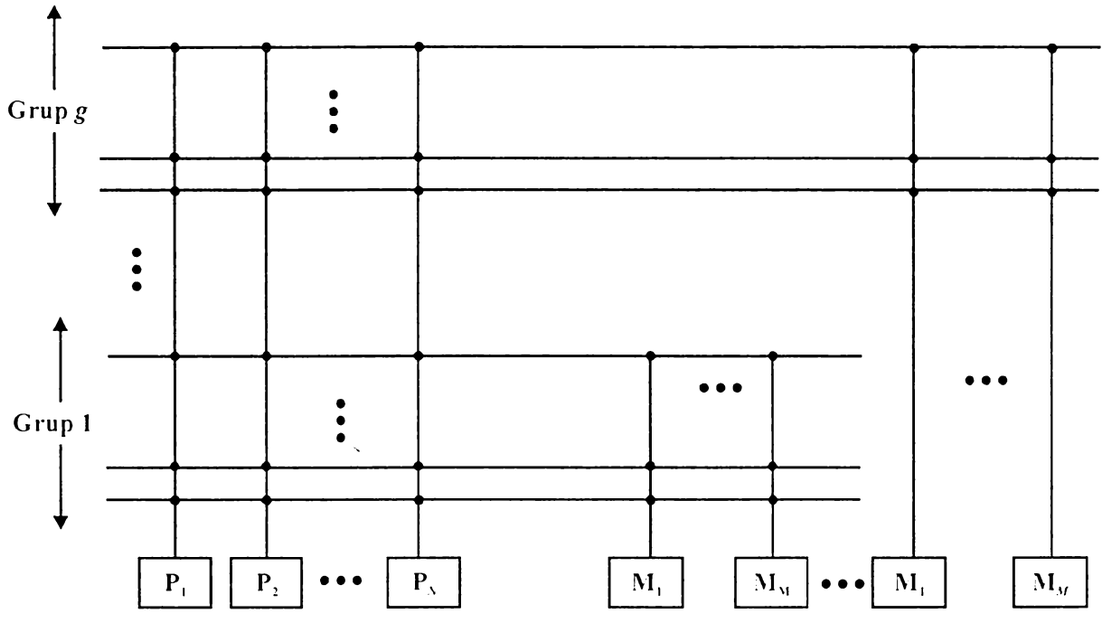
Ea conține N procesoare, P_1, P_2, ... P_N, fiecare având propria memorie cache - MCH - şi toate conectate la o memorie partajată - MP - prin intermediul a B magistrale, B_1, B_2, ..., B_B. MP constă din M blocuri întrețesute, M_1, M_2, ..., M_M pentru a permite solicitări concurente de acces la MP, memoria partajată. Această tehnică evită pierderea de performanță ce apare dacă accesele sunt serializate - cazul unui singur bloc de memorie.

Fiecare procesor este conectat la fiecare magistrală şi la fiecare bloc de memorie. Când un procesor dorește să aibă acces la un anumit modul, el are B magistrale din care poate alege una pentru a realiza conectarea. Astfel fiecare cuplu procesor - memorie este conectat prin câteva căi redundante ceea ce implică un comportament mai tolerant la defectarea uneia sau a mai multor magistrale, evident cu prețul degradării performanței sistemului în termeni de latență şi cost.

În fig.3.5.b se prezintă cazul unei conectări *paralele* printr-o magistrală multiplă. Fiecare memorie este conectată doar la un subset din magistrala multiplă. În particular, o configurație completă cu magistrală multiplă având $B \approx N/2$ are aproximativ aceeaşi lățime de bandă ca şi un *crossbar* $N \times M$.



(a)



(b)

Fig. 3.5 Structura unei magistrale multiple

În SMP cu *magistrală multiplă* trebuie adoptată o politică de alocare a magistralelor disponibile la procesoarele ce solicită acces la memorie. În particular acestă politică trebuie aplicată în cazul când $N > B$.

Procedura de alocare se implementează prin arbitri hardware ce conduce însă la creşterea complexităţii reţelei de interconectare.

Tehnicile de interconectare a procesoarelor multiple şi în particular, conectarea lor la o memorie partajată, au fost studiate în literatură [MHW87], [Pat81]. Cel mai studiat tip, probabil, este *reţeaua multinivel*. Acest tip de conectare are mai puţine componente decât o reţea *crossbar* dar reţine mai multe dintre proprietăţile de conectivitate specifice ei.

În contrast, SMP-MP utilizează arhitecturi de interconectare bazate pe magistrale convenționale.

Interconectarea prin magistrale multiple reține facilitățile unei singure magistrale dar permite construcția unor sisteme mai mari ce pot concura cu sistemele bazate pe rețele multinivel în ceea ce privește puterea de calcul.

A. Operarea sistemului cu magistrală multiplă

Importanța memoriilor cache rezidă din faptul că utilizarea lor reduce traficul pe magistrală.

Se pot construi astfel sisteme mai performante fără să crească numărul de magistrale.

Creşterea numărului şi a timpului de acces la memoria partajată MP depinde de algoritmul de utilizare a memoriei cache. Un aspect cheie al acestui algoritm este politica de actualizare a MP când o solicitare de acces este adresată memoriei cache - MCH.

Cele mai utilizate tehnici sunt *recopierea* şi *rescrierea*, unde un bloc solicitat din MCH este copiat în MP numai când blocul este dealocat din MCH.

Două surse de conflict legate de solicitările la memorie sunt prezente în sistemul din fig. 3.4.

Prima, mai multe solicitări pot fi adresate aceluiaşi modul de memorie. A doua sursă, capacitatea disponibilă a magistralei poate fi insuficientă pentru a se acomoda la toate solicitările.

În mod corespunzător, alocarea unei magistrale la un procesor necesită un proces în două faze după cum urmează :

(1). Conflictele de acces la memorie sunt rezolvate mai întâi de M-arbitrii "1 din N", câte unul pentru fiecare modul de memorie. Fiecare arbitru "1 din N" selectează o solicitare din până la maximum N posibile pentru a permite accesul la blocul de memorie;

(2). Solicitările de acces la memorie selectate de arbitrii de memorie sunt apoi alocate la o magistrală de câte un arbitru "B din M". Acesta poate selecta până la B solicitări din M arbitri de memorie.

Ipoteza operării asincrone a liniilor de adrese şi de date permite ca procesul de arbitrare să se suprapună peste transferul datelor.

Aparent s-ar părea că ar trebui să existe o magistrală pentru fiecare bloc de memorie.

Aceasta ar conduce la necesitatea unui al doilea nivel de arbitrare. Configurația optimă poate necesita mai puține magistrale decât numărul de blocuri de memorie.

B. Modelarea performanțelor

Cererile de acces ale unui procesor la memorie pot fi modelate ca şi o secvență de încercări Bernoulli independente. Se pot dezvolta expresii pentru un număr mediu de solicitări de acces produse de N procesoare la începutul fiecărui ciclu de memorie.

Fie p probabilitatea ca un procesor oarecare P_i să solicite acces la MP la începutul unui ciclu de memorie (aceasta este o încercare de tip Bernoulli), atunci Nr reprezintă numărul aşteptat de solicitări la MP.

Unele cereri vor fi întotdeauna blocate datorită celor două tipuri de conflicte menționate anterior, indiferent de dimensiunile lui B şi M.

BW reprezintă lățimea de bandă a subsistemului compus din B magistrale şi din M blocuri de memorie, adică numărul aşteptat de blocuri de memorie ocupate. De asemenea BW poate fi considerat ca reprezentând numărul aşteptat de accese reuşite la memorie.

Datorită conflictelor de acces $BW < Nr$.

Cînd două sau mai multe procesoare intenționează să acceseze memoria, se utilizează un arbitru pentru selectarea doar a unei cereri.

Dacă presupunem că blocurile de memorie sunt întrețesute pe spațiul biților mai puțin semnificativi ai adresei blocului cache şi că ciclurile de extragere a instrucțiilor şi accesul la date se întrepătrund, atunci empiric se poate sugera că solicitările la toate blocurile de memorie sunt egal probabile.

Probabilitatea ca o solicitare de acces de la P_i către un bloc anume M_j este r/M independent de i sau j.

În literatură [MHB86] dezvoltarea unei expresii pentru BW se face în doi paşi, corespunzător celor două niveluri de arbitrare.

(1) Arbitrarea memoriei

Probabilitatea să nu existe solicitări de la P_i către M_j este $1 - p/M$, şi prin urmare probabilitatea ca nici unul dintre procesoare să solicite M_j la începutul unui ciclu de memorie este $(1 - p/M)^N$.

Fie E_j evenimentul constând în faptul că există cel puțin o solicitare pentru M_j.

Cu alte cuvinte că arbitrul "1 din N", pentru M_j, extrage o solicitare de acces.

Atunci :

$$\Pr[E_j] = 1 - \left(1 - \frac{p}{M}\right)^N = q \tag{3.4}$$

pentru orice i.

Dacă evenimentele E_j , $j = 1, \ldots, M$, se presupun ca fiind independente şi dacă există întotdeauna un număr suficient de magistrale, adică $B \geq M$, atunci numărul aşteptat de blocuri de memorie ocupate este :

$$BW_S = \sum_{j=1}^{M} \Pr[E_j] = M\left[1 - (1 - \frac{p}{M})^N\right] \tag{3.5}$$

unde S înseamnă "suficiente magistrale".

În cazul unui N mai mare, expresia are limita inferioară aproximativă

$$BW_S \approx M\left(1 - e^{-\frac{Np}{M}}\right) \tag{3.6}$$

Ecuaţia (3.6) poate fi evaluată prin estimarea lui p, a capacităţii MC, a raportului a dintre timpul ciclului de memorie şi timpul ciclului de procesor.

În cel mai general caz, când $B < M$, se ajunge la arbitrarea magistralelor.

(2) Abitrarea magistralelor

Presupunerea că evenimentele E_j sunt independente nu permite să exprimăm *f(i)* ca exact i dintre arbitrii de memorie să producă o solicitare la începutul unui ciclu de memorie astfel:

$$f(i) = \binom{M}{i} \Pr[E_i]^i (1 - \Pr[E_i])^{M-i} = \binom{M}{i} q^i (1-q)^{M-i} \tag{3.7}$$

In cazul unde $i \leq B$, există suficiente magistrale care să manipuleze solicitările de acces şi arbitrul "B din M" nu blochează nici o solicitare. In cazul unde $i > B$, toate magistralele B sunt utilizate şi arbitrul "B din M" blochează $(i - B)$ din solicitări.

Tinând cont de aceste două cazuri se poate scrie expresia pentru numărul de blocuri de memorie utilizate astfel:

$$BW = \sum_{i=1}^{B} if(i) + \sum_{i=B+1}^{M} Bf(i) \tag{3.8}$$

Când $B = M$, $BW = BW_S$.

In general însă, $Nr > BW_S > BW$ pentru $r > 0$.

Aceste inegalități corespund conflictelor ce fac ca solicitările de acces la memorie să fie blocate în timpul arbitrării memoriei ($Nr > BW_S$) și în timpul arbitrării magistralei ($BW_S > BW$).

Mudge [MHB86] a observat că ecuația (3.6) se bazează pe două presupuneri: *independența temporală* și *independența spațială*.

Independența temporală pretinde ca solicitările succesive de acces la memorie să fie independente.

Independența spațială, corespunde evenimentelor E_i independente, o ipoteză cu validitate limitată.

Dacă notăm

$$\mathbf{F}(B) = \sum_{i=B}^{M} f(i) \tag{3.9}$$

atunci

$$BW = B\mathbf{F}(B) + \sum_{i=1}^{B-1} if(i) \tag{3.10}$$

Se poate generaliza (3.10) pentru cazul magistralelor partajate (fig. 3.5.b). Analiza interferenței de memorie este aceeași ca și în cazul anterior deoarece este independentă de configurația magistralei.

Analiza interferenței magistralelor necesită modificări. Fie cele B magistrale divizate în g grupe egale, presupunând că g este un factor depinzând de B și M. Cu $m = M/g$ și $b = B/g$ ecuațiile (3.9) și (3.10) devin

$$fg(i) = \binom{m}{i} q^i (1-q)^{m-i} \tag{3.11}$$

$$\mathbf{F}_g(B) = \sum_{i=b}^{m} fg(i) \tag{3.12}$$

În mod corespunzător

$$BW_g = g\left[b\mathbf{F}_g(B) + \sum_{i=1}^{b-1} if_g(i) \right] \tag{3.13}$$

care reprezintă de g ori lățimea de bandă a oricărui din cele g subsisteme formate din N procesoare, b magistrale şi m memorii.

În fig. 3.6 se prezintă comportamentul BW în funcție de numărul de magistrale.

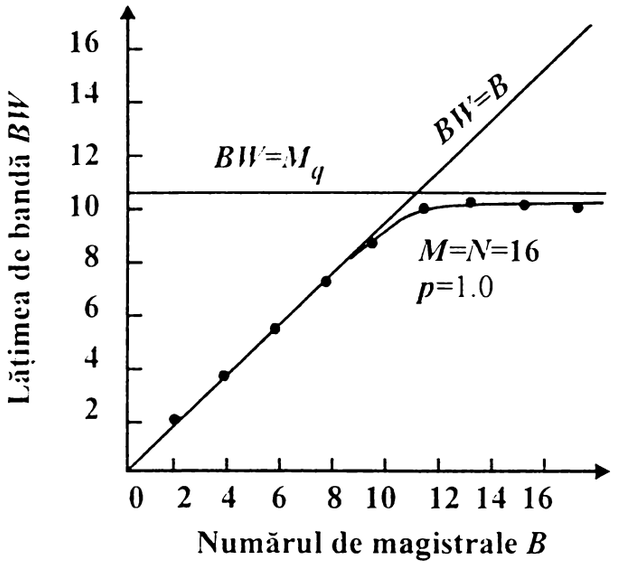


Fig. 3.6 Comportarea asimptotică a lui BW

C. Proiectarea circuitelor de arbitrare

Se utilizează două tipuri de arbitri:

"1 din N" – selectează procesorul din cele ce solicită accesul

"B din M" – alocă magistralele la acele procesoare care au reuşit să obțină accesul la memorie.

Arbitrii "1 din N"

Fiecare procesor P_i are o linie de solicitare R_i şi o linie de alocare G_i. Procesorul P_i solicită un acces la memorie prin activarea liniei R_i şi arbitrul indică alocarea blocului de memorie activând G_i.

Schemele de arbitrare se pot împărți în trei categorii: *cu prioritate fixă, în inel* şi *structură arborescentă.*

Arbitrii cu *prioritate fixă* sunt simpli şi rapizi, dar ei au dezavantajul că nu asigură şanse egale în procesul de arbitrare. Procesoarele cu prioritate scăzută pot fi forțate să aştepte nedefinit dacă procesoarele cu prioritate ridicată mențin memoria ocupată.

Arbitrii cu *structură inelară* oferă prioritate procesoarelor pe baza de rotație, cu prioritatea cea mai coborâtă data procesorului ce a utilizat cel mai recent modulul de memorie solicitat.

Se garantează astfel tuturor procesoarelor că într-un interval finit de timp vor accesa memoria, dar timpul de arbitrare crește linear cu numărul de procesoare.

Un arbitru "1 din N" cu structura arborescentă este în general un arbore binar de adâncime $\log_2 N$ construit din module de arbitrare "1 din 2". Schema unui astfel de arbitru este prezentată în fig. 3.7.

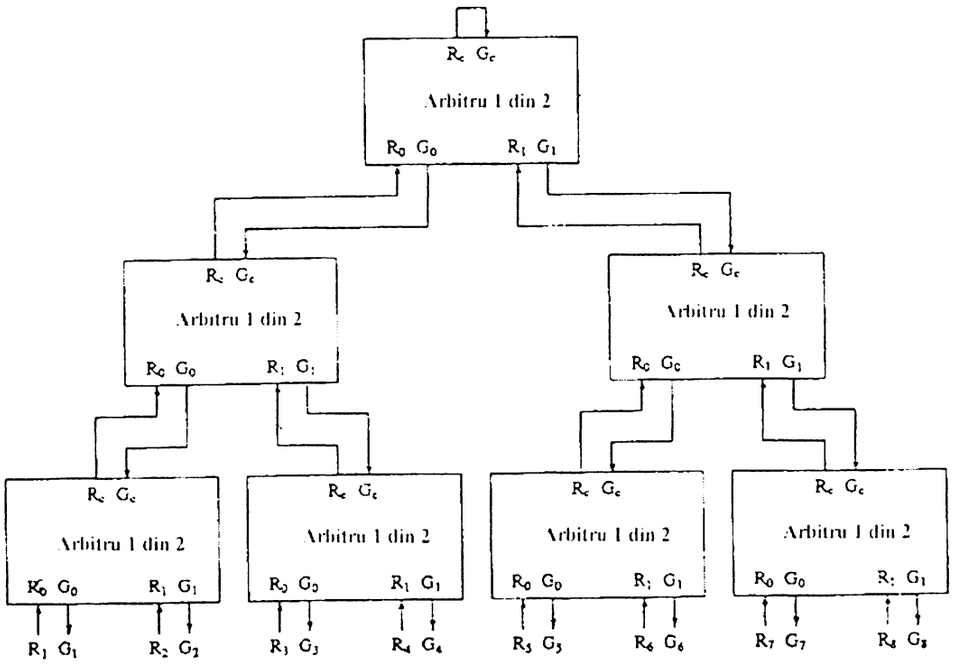


Fig. 3.7. Schema bloc a arbitrului pentru magistrala multiplă

Acest tip de arbitru este mai rapid decât arbitrii în inel deoarece timpul de arbitrare crește ca și $O(\log_2 N)$ în loc de $O(N)$.

Memorarea stării este necesară pentru o abordare corectă ținând cont de asignările anterioare ale magistralei. După fiecare ciclu de arbitrare, prioritatea cea mai mare este dată modulului ce urmează imediat celui ce a fost servit - de fapt se aplică o politică standard de tip *round - robin*.

Un ciclu de arbitrare începe având toate magistralele disponibile. Registrul de stare, prezentat în fig.3.8, identifică modulul de arbitrare A_j cu cea mai mare prioritate prin poziționarea e_i la intrarea acelui modul.

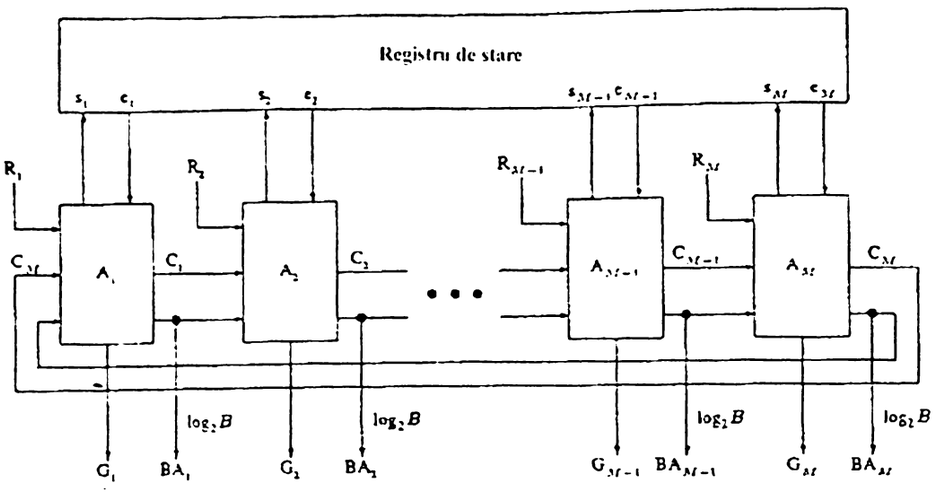


Fig. 3.8. Structura registrului de stare

Arbitrarea începe cu acest modul şi parcurge inelul de la stânga spre dreapta, la fiecare modul de arbitrare, intrarea R_i este examinată pentru a se observa dacă modulul de memorie M_i solicită magistrala. Dacă o solicitare este prezentă şi magistrala este disponibilă, atunci adresa primei magistrale disponibile este plasată pe ieşirea BA_i şi G_i este poziţionat.

BA_i este de asemenea transmis următorului modul pentru a indica magistralei cu indicele cel mai mare, că a fost asignată. Dacă unui modul nu i-a fost alocată o magistrală, atunci ieşirea corespunzătoare $BA_i = BA_{i-1}$, adică cu intrarea sa.

Dacă modulului i-a fost alocată o magistrală atunci ieşirea $BA_i = BA_{i-1}$.

Când $BA_i = B$ toate magistralele au fost utilizate şi procesul de asignare încetează.

Modulul cu cea mai mare prioritate A_i, indicat de semnalul e_i, ignoră intrarea sa BA_{i-1} se începe asignarea magistralei cu prima magistrală poziţionând $BA_i = 1$.

Intrarea C_{i-1} a fiecărui modul este un semnal provenind de la modulul anterior ce indică că acesta şi-a completat asignarea magistralei.

Arbitrarea continuă secvenţial prin toate modulele până când toate magistralele au fost asignate, sau până când toate solicitările au fost satisfăcute.

Ultimul modul îşi poziţionează propriul semnal s_i. Acesta este înregistrat în registrul de stare, care îl utilizează pentru a selecta următoarea ieşire e_i astfel încât următorul ciclu de arbitrare va începe imediat după acela în care s-a asignat ultima magistrală.

Referitor la performanţele arbitrilor "B din M", în literatura dedicată acestui subiect [LV 82] se evaluează întârzierea în arbitrare în raport cu M, numărul modulelor de arbitrare.

Prin combinarea a g din aceste module într-un singur modul, întârzierea este redusă cu un factor ce depinde de g.

Utilizând circuite PLA în implementarea schemelor din această categorie, circuite având o întârziere de 3Δ, întârzierea schemei de arbitrare devine $(3M/g)\ \Delta$.

În schema din fig. 3.7, unde $M = 16$ şi $g = 4$, întârzierea devine aproximativ 12Δ.

Întârzierea pe ansamblul de arbitrare format din arbitrul "1 din N" şi din "B din M" devine $18\Delta + 12\Delta = 30\Delta$.

Datorită caracterului asincron arbitrarea se poate suprapune peste accesul la magistrală ceea ce conduce la concluzia că durata ciclului memorie - magistrală poate fi în final de 30Δ.

3.1.4 Analiza performanţelor reţelelor de interconectare de tip *crossbar*

O reţea de interconectare de tip *crossbar* (RIN-CB) este constituită dintr-o matrice de perechi de contacte operate individual în care există câte o pereche pentru fiecare combinaţie de I/E. Schema de principiu este prezentată în fig. 2.32

O RIN-CB cu N intrări şi M ieşiri este denumită $N \times M$. Atât timp cât nu există interferenţă de memorie printr-un set de solicitări de acces la memorie generate de către procesoare, adică nu există două sau mai multe procesoare ce solicită accese la acelaşi modul de memorie, se pot stabili toate conexiunile în mod simultan.

Această disponibilitate este însă penalizată cu un cost ridicat al conectării care este $(O(NM))$. Deşi RIN-CB poate asigura simultan toate conexiunile, BW este cu mult mai scăzută decât capacitatea teoretic posibilă. Această scădere se datorează interferenţei de memorie cauzată de natura aleatorie a cererilor de acces într-un mediu multiprocesor. Prin urmare analiza de performanţă a unui RIN-CB se reduce la analiza interferenţei de memorie.

În literatură [BYA 89] se prezintă mai multe modele pentru interferenţa de memorie. În general, însă operaţiile sistemului sunt aproximate de procese stochastice după cum urmează: La începutul unui ciclu de sistem, un P_i selectează un modul M_j în mod aleator şi produce o solicitare de acces la acest modul cu o probabilitate p. Dacă se produc mai mult de o solicitare la acelaşi modul M_j, controlerul memoriei va selecta aleator una şi procesoarele respinse vor reîncerca în ciclul următor. Comportamentul procesoarelor este considerat independent şi statistic identic, aşa cum este şi comportamentul modulelor de memorie.

Pentru studiul interferenței de memorie se aplică modele de lanțuri Markov discrete. Se admite că un modul M este caracterizat de *durata ciclului* t_c ,ce constă din *timpul de acces* t_a, urmat de un *timp de rescriere* t_w.

Adică:

$$t_c = t_a + t_w$$

Comportamentul procesorului P este modelat ca şi o secvență ordonată, constând dintr-o solicitare de acces la memorie urmată de un *timp de execuție* t_p.

Parametrul t_p este măsurat din momentul în care data solicitată a fost obținută în urma cererii anterioare până la momentul în care următoarea cerere a fost transmisă memoriei.

În sistemele reale, $t_w = t_p$, ceea ce este echivalent cu situația în care P_i generează o cerere de acces la M_j la începutul fiecărui ciclu de memorie.

Complexitatea modelului interferenței memoriei este ameliorată dacă se presupune că un procesor încetează cererea şi generează o nouă cerere independentă la începutul ciclului următor (ipoteza independenței cererilor de acces).

Pentru un sistem cu N procesoare şi M module de memorie, dacă un procesor generează o cerere de acces cu probabilitatea p într-un ciclu direcționat spre fiecare modul de memorie cu o probabilitate egală (modelul de referință uniformă), atunci lățimea de bandă a memoriei este:

$$BW = M\left(1-\left(1-\frac{p}{M}\right)^N\right) \tag{3.14}$$

Explicația este următoarea: Dacă p/M este probabilitatea ca un procesor să solicite un acces la un anume modul de memorie, $[1-(p/M)]^N$ reprezintă probabilitatea ca niciunul din cele N procesoare să solicite acel modul de memorie într-un anume ciclu.

Paranteza $\left(1-\left(1-\frac{p}{M}\right)^N\right)$ indică probabilitatea ca cel puțin o solicitare de acces să fie transmisă.

În fine, factorul de multiplicare M dă numărul aşteptat de module distincte de memorie ce au fost solicitate într-un ciclu, şi prin urmare lățimea de bandă BW.

Ecuația (3.14) reprezintă un indicator simplu de previzionare a performanței unui RIN-CB. O ipoteză mai rafinată prezentată în [BYA 89] introduce problema *referinței neuniforme* (*non uniform reference model*). În acest model –MRNU- se introduce

conceptul de *memorie favorită a unui procesor*. Modulul de memorie solicitat cel mai des de un procesor este denumit *memoria favorită a procesorului*.

Fie m probabilitatea cu care un procesor îşi adresează memoria sa favorită, presupunând că procesorul generează o solicitare de acces într-un ciclu.

Atunci lăţimea de bandă pentru un SMP-MP cu RIN-CB de dimensiuni $N \times M$ este:

$$BW = N\left[1-(1-pm)\left(1-p\frac{1-m}{N-1}\right)^{N-1}\right] \tag{3.15}$$

Se furnizează de asemenea soluţii pentru cazurile $M \le N$ şi $M \ge N$.

Dacă se substituie $m=1/M$, analiza se reduce la modelul MRU.

Ecuaţiile sunt valabile doar în cazul sistemelor *sincrone* şi cu comutare prin circuite.

În cazul sistemelor *asincrone* comutate prin circuite se utilizează modele în care procesoarele sunt modelate ca şi un set de servere şi memoriile ca şi un set de şiruri de aşteptare cu disciplină FIFO.

Modelul este prezentat în fig. 3.9. În Tabel II se prezintă sintetic diverse tehnici analitice de evaluare a RIN-CB, acurateţea modelării, metrica de evaluare a performanţei şi costurile de calcul implicate.

TABEL II. Rezumat al analizelor RIN - *crossbar*

	RIN- *crossbar* sincronă		**RIN-*crossbar* asincronă**
Tehnica de analiză	Şir Markov discret	Probabilistică cu ipoteză de independenţă	Reţea de şiruri de aşteptare
Reprezentarea parametrilor	Rată de cerere acces	Probabiltatea de cerere acces	Timp de aşteptare
Parametrii de performanţă	BW sau P_A	BW sau P_A	P_u sau P_W
Acurateţea	Exactă	Bună	Exactă
Costul de calcul	Foarte ridicat	Coborât	Moderat

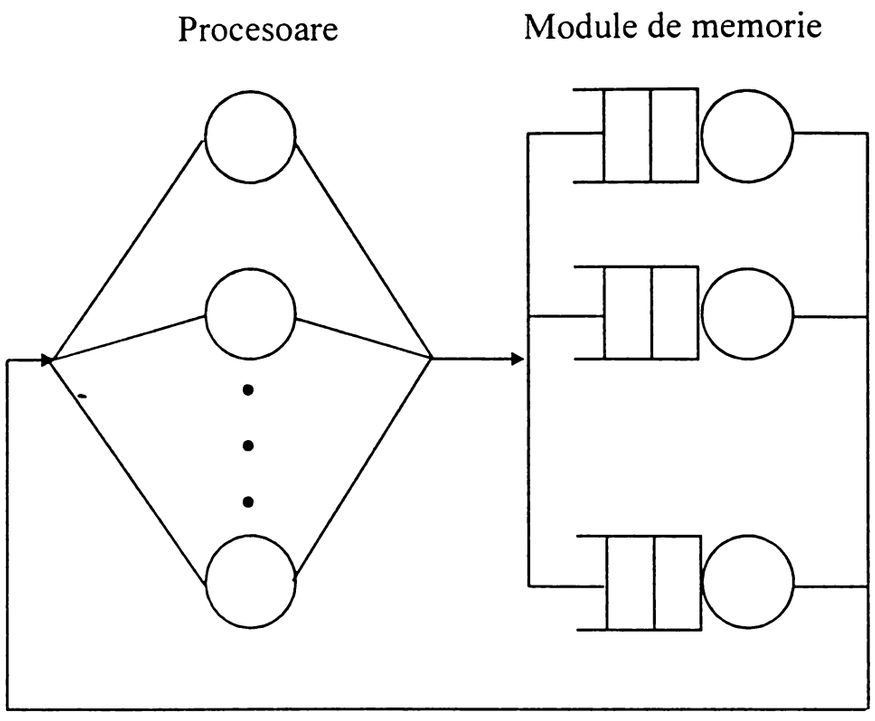


Fig. 3.9 Un model cu cozi de aşteptare pentru SMP - MP bazate pe RIN - CB

3.1.5 Analiza performanţelor reţelelor de interconectare *multinivel*

Reţelele de interconectare *multinivel* (RIN-MN) au apărut relativ recent şi au avantajul principal în raportul cost / performanţe acceptabil. Ele permit o multitudine de conexiuni unu la unu ale procesoarelor cu modulele de memorie, în timp ce costul hardware se reduce la $O(N \log N)$ în comparaţie cu $O(N^2)$ pentru o RIN - CB.

O RIN-MN de talie $N \times N$ conectează N procesoare la N memorii. Dacă N este o putere a lui 2, atunci se utilizează $\log_2 N$ etaje de comutatoare 2×2 cu $N/2$ comutatoare per nivel. Fiecare comutator are două intrări şi două ieşiri. Conexiunea dintre o intrare şi o ieşire este stabilită depinzând de un bit de control furnizat de intrare. Când $C=0$, intrarea este conectată la ieşirea superioară, iar când $C=1$, ea este conectată la ieşirea inferioară ca şi în fig. 2.22.

O RIN-MN *omega*, prezentată în fig. 2.35, este caracterizată de o interconectare de tip "*amestec perfect*" ce precede fiecare nivel de comutatoare.

Procesorul solicitat generează o etichetă ce este o reprezentare binară a destinaţiei. Conectarea unui comutator de pe nivelul i este apoi însoţită de către al i-lea bit a acestei etichete binare contorizat de la bitul cel mai semnificativ.

Conexiunea dintre intrarea 3 şi ieşirea 5 (101_2) este prezentată prin linie întărită în fig. 2.35. Proprietatea de autorutare a RIN-MN evită necesitatea existenţei unui dispozitiv central de comandă, ceea ce reprezintă un avantaj pentru SMP-MP. Prin urmare

discuțiile asupra performanțelor prezentate în această secțiune se vor concentra în mod unic asupra schemelor descentralizate de comandă.

Dimensiunea comutatorului în RIN-MN nu trebuie să fie restricționată la 2×2.

În literatură [BYA89], [MD96], se citează cazul mașinii Butterfly ce conectează N intrări la N ieșiri utilizând comutatoare *crossbar* 4×4 și un număr de $\log_4 N$ de niveluri cu $N/4$ comutatoare per nivel.

O RIN-MN *delta* poate conecta $N = a^n$ intrări la $M = b^n$ ieșiri.

În general se pot conecta orice $M = m_1 * m_2 * \ldots * m_r$ intrări la $N = n_1 * n_2 * \ldots * n_r$ ieșiri prin intermediul a r niveluri de comutatoare.

Toate proiectele de sisteme bazate pe RIN-MN suferă însă de un dezavantaj major. Există doar o singură cale de la o intrare la o ieșire. Din acest motiv este necesar să se încorporeze câteva facilități de toleranță la defecte în aceste rețele astfel că cel puțin o defecțiune să fie tolerată într-un comutator sau într-un traseu de conectare.

În literatură [BYA 89] se sugerează o abordare probabilistică în analiza RIN-MN de tip "*delta*". Metoda presupune un *model de referință uniform* - MRU - și ipoteza *independenței cererilor de acces la memorie*.

Se consideră o RIN-MN *delta* de dimensiuni $a^n \times b^n$ constituită din n module *crossbar* $a \times b$. Fiecare nivel a RIN-MN *delta* este comandat de un bit cu destinație distinctă (codificat în baza b) pentru poziționarea comutatoarelor $a \times b$ individuale. Deoarece destinațiile sunt independente și uniform distribuite, cererile de acces la oricare modul $a \times b$ sunt independente și uniform distribuite pentru cele b destinații diferite.

Ecuația (3.14) poate fi aplicată la orice element în RIN-MN *delta*.

Numărul așteptat de solicitări ce traversează spre cele b ieșiri, este obținut punând $N = a$ și $M = b$ în ecuația (3.14).

Împărțind acest număr prin b se obține probabilitatea de solicitare a accesului a oricărei dintre cele b linii de ieșire ale unui comutator $a\times b$, ca și o funcție a probabilității de intrare. Deoarece ieșirea unui nivel este intrarea următorului nivel, se poate evalua de o manieră recursivă probabilitatea de ieșire a oricărui nivel pornind cu nivelul 1. Dacă p_i este probabilitatea ca să existe o solicitare la ieșirea unui comutator de pe nivelul i, atunci:

$$p_i = 1 - \left(1 - \frac{P_i}{b}\right)^a \qquad (3.16)$$

pentru $1 \le i \le n$.

În particular, probabilitatea de ieşire a nivelului final determină lăţimea de bandă a RIN-MN *delta*, ce este $BW = p_n b^n$.

Această tehnică analitică a fost utilizată pentru evaluarea diferitelor tipuri de RIN-MN.

În literatură [BYA 89] se prezintă şi extinderea analizei pentru cazul *memoriei favorite*.

Pentru RIN-MN de dimensiuni $N \times N$, cu $N = a^n$, procesoarele sunt definite ca fiind conectate la memoriile lor favorite când toate comutatoarele sunt conectate direct, adică intrarea *i* a unui comutator este conectată la ieşirea *i* a circuitului.

Într-o RIN-MN *omega*, MM_j devine memoria favorită a procesorului P_j. Fie q_{i-1} probabilitatea că există o cerere favorită de acces la intrarea nivelului *i*. Într-o RIN-MN *delta* de dimensiuni $N \times N$, unde $N = a^n$, putem exprima:

$$p_i = 1 - (1 - p_{i-1} q_{i-1}) \left(1 - p_{i-1} \frac{1 - q_{i-1}}{a - 1}\right) \qquad (3.17)$$

Mai sunt necesare aproximativ şase alte ecuaţii pentru a determina q_{i-1} la nivelul i^3 şi, în final, $BW = N p_n$.

Analiza de mai sus este valabilă pentru RIN-MN sincrone cu comutare de pachete în următoarele condiţii:

- pachetele sunt generate doar la începutul ciclului RIN-MN;
- comutatoarele nu au circuite tampon, astfel încât pachetele sunt aleatoriu alese în caz de conflicte şi pachete ce nu au fost comutate sunt pierdute.

RIN-MN *delta* au mai fost studiate şi cu ajutorul reţelelor Petri ce oferă facilităţi grafice pentru o descriere precisă a operaţiilor de sistem. De asemenea performanţele pentru sisteme mici pot fi mai uşor evaluate.

Aceste modele bazate pe grafuri permit descrierea clară a concurenţei, conflictelor şi sincronizării task-urilor.

Oricum complexitatea reţelelor Petri creşte exponenţial cu creşterea în dimensiuni a sistemului.

3.1.6 Analiza comparativă a performanţelor RIN de tip *crossbar*, *magistrală multiplă* şi *multinivel*

În finalul acestui paragraf se impune analiza comparativ a performanţelor pentru trei tipuri de RIN specifice SMP-MP.

Nu se ia în considerare *magistrala unică* deoarece performanțele ei sunt mult mai slabe comparativ cu acelea ale RIN indicate mai sus.

Analiza se bazează pe compararea costului circuitelor şi a performanțelor sistemului pus în discuție.

Tabelul prezintă facilitățile hardware în mod selectiv pentru cele trei tipuri de RIN.

TABEL III. Sumarul facilităților hardware ale celor trei RIN

	crossbar	**MN**	**MM**
Nr. de comutatoare sau de conexiuni	$N \times M$	$N \log N$	$B \times (N+M)$
Încărcarea magistralelor	N	1	B
Nr. de fire	M	N	B
Arbitru	M "1 din N"	$N \log N$ "1 din 2"	1 "B din M" şi M "1 din N"
Toleranța la defect şi expandabilitate	Acceptabilă	Slabă	Bună

Numărul de elemente de comutare utilizate în RIN *crossbar* de dimensiuni $N \times M$ este $N \times M$ în comparație cu $N \log N$ ce sunt prezente în RIN – MN.

Numărul de conexiuni necesare într-o RIN – MM de dimensiuni $N \times M \times B$ este proporțional cu $B\ (N+M)$. Deoarece fiecare magistrală într-un SMP cu RIN – MM trebuie să conecteze $N+M$ module, încărcarea magistralei este proporțională cu $N+M$, în timp ce sarcina magistralei la o RIN – MN este 1 datorită conectării *unu la unu*.

Toate cele trei tipuri de RIN necesită arbitrii de complexități diferite pentru rezolvarea conflictelor de acces.

Expandabilitatea şi *siguranța în funcționare* sunt două facilități hardware foarte importante. În acest context o RIN – MN este foarte convenabilă, în raport cu celelalte două, datorită proprietății de *reconfigurabilitate* şi a *existenței căilor multiple de date* între fiecare procesor şi memorie. Se mai poate opera chiar şi în modul degradat după apariția unei defecțiuni a unui subset de magistrale.

În literatură [BYA 89] se prezintă rezultatele mai multor cercetări în domeniul RIN – MN tolerante la defect şi reconfigurabile. Toleranța la defect a RIN – MN poate fi realizată prin adăugarea de circuite suplimentare ca de exemplu niveluri suplimentare

sau căi dublate de date. Este mai simplu să se reconfigureze un *crossbar* dacât o RIN – MN.

În situația apariției unui defect, o anumită coloană sau o anumită linie pot fi îndepărtate şi RIN poate opera în modul degradat.

În ceea ce priveşte situația comparativă a performanțelor celor trei tipuri de RIN, figurile 3.10, 3.11 şi 3.12 sunt edificatoare.

În fig. 3.10, cele două curbe sunt trasate conform ecuațiilor (3.14) şi (3.15). Distanța dintre cele două curbe creşte pe măsură ce talia sistemului creşte. P_A rămâne constant la *crossbar* atunci când talia SMP – MP devine foarte mare. P_A descreşte la RIN – MN când talia sistemului creşte.

În fig. 3.11 se evidențiază saturarea rapidă a BW la magistrala unică şi creşterea lui în raport direct cu creşterea numărului de magistrale.

În fig. 3.12 prezintă comparativ comportarea RIN sincrone cu pachete comutate. Timpii ce caracterizează ciclul de procesor, întârzierea transferului de magistrală ca şi timpul total de transfer al unui pachet ideal între o intrare şi o ieşire al unui RIN – MN se presupun fixați în raport cu timpul ciclului de sistem. Timpul de acces la memorie se presupune egal cu patru cicluri de sistem. În timpul unui acces la memorie, procesorul ce trimite o solicitare de acces la memorie rămâne în aşteptare până când accesul la memorie este finalizat.

Rata de solicitare a memoriei ce caracterizează un procesor, analizată prin prisma unui RIN, se prezintă în fig. 3.12. De remarcat că un SMP – MP bazat pe RIN – MM cu doar patru magistrale poate să realizeze aproximativ aceleaşi performanțe ca şi un SMP – MP bazat pe RIN *crossbar* în timp ce costurile hardware sunt semnificativ diminuate.

Toate analizele anterior prezentate sunt pertinente în mediile sincrone cu comutare de circuite sau cu comutare de pachete. Pentru RIN - MN de talie mare, controlul centralizat, global al operațiilor rețelei este dificil. Prin urmare trebuie să se ia în considerare proiecte asincrone pentru SMP de talie mare.

Al doilea dezavantaj al acestei analize este ignorarea timpului de aşteptare al unui procesor pentru epuizarea accesului la memorie, dar se presupune sosiri continue la intrare după o distribuție Poisson.

Al treilea dezavantaj este presupunerea unui acces la o memorie favorită. Configurațiile de referire la memorie sunt puternic dependente de program şi pot fi arbitrare.

În literatură [BYA89], [YLL90], [YB91], se descrie metode bazate pe *cozi de aşteptare* şi pe *analiza a valorii medii* a RIN - MN operate asincron.

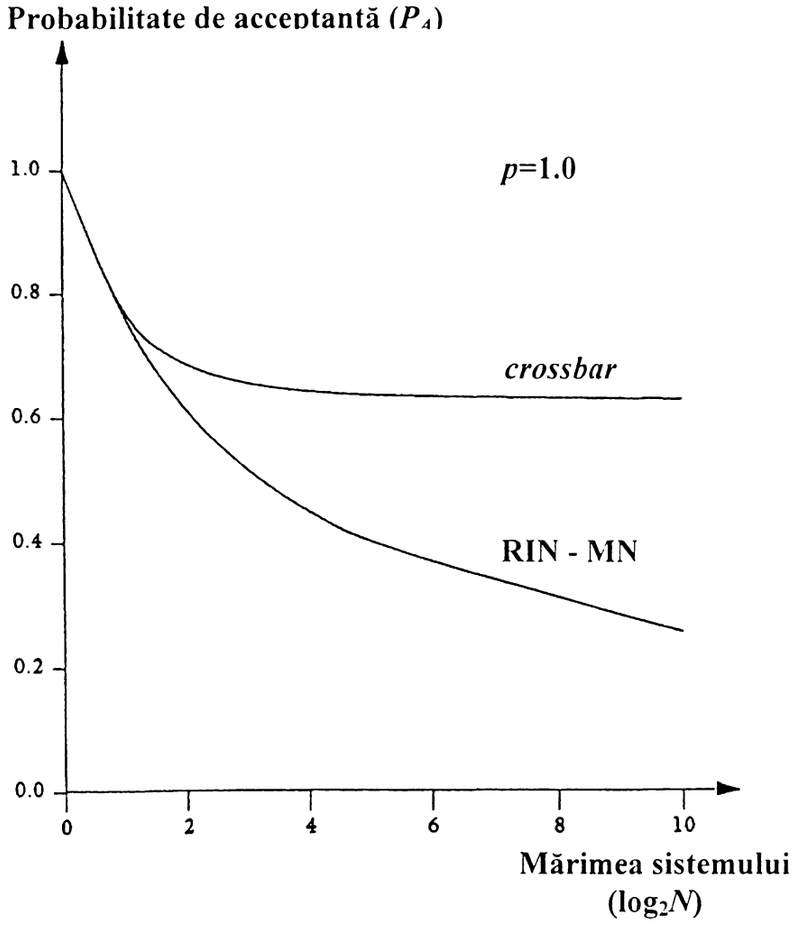


Fig. 3.10 Probabilitatea de acceptanță ca şi funcție de mărimea sistemului pentru sisteme sincrone comutate prin circuit

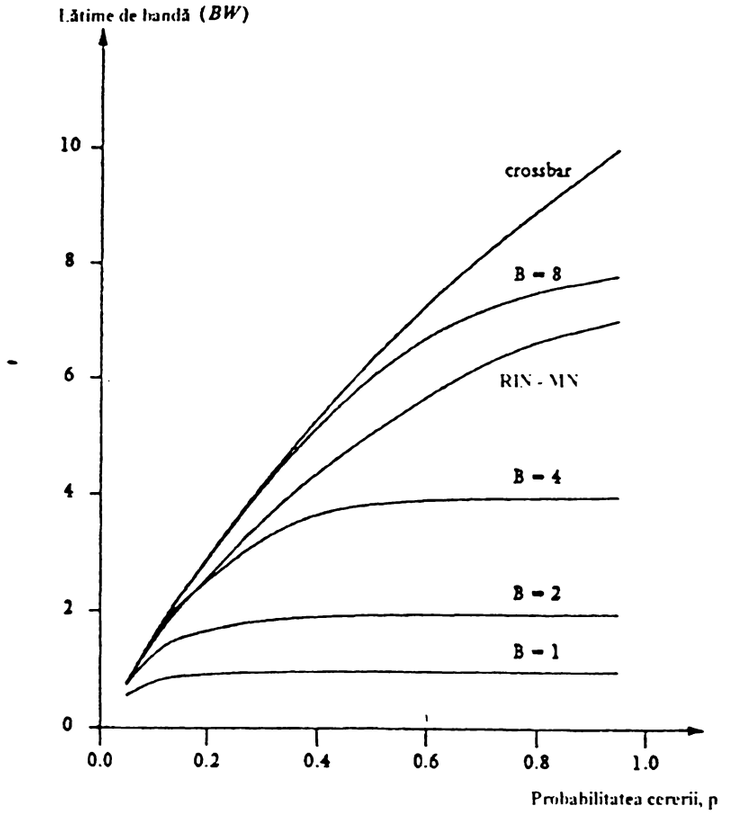


Fig. 3.11. Lățimea de bandă a memoriei ca şi o funcție de probabilitate de solicitare de acces pentru un sistem sincron 16 x 16 comutat prin circuit

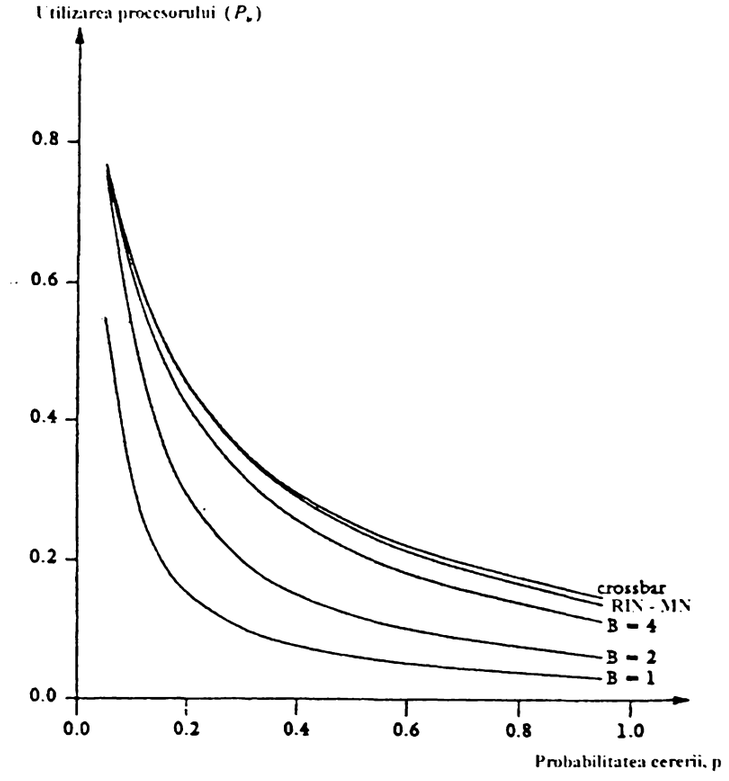


Fig 3.12 Utilizarea procesorului ca şi o funcție a probabilităților de solicitare de acces pentru un sistem sincron 16 x 16 comutat prin pachet

3.2 ANALIZA DE PERFORMANȚĂ A SISTEMELOR DE MEMORII ÎN ARHITECTURILE SMP DIN CLASA MIMD

În capitolul 2 s-au analizat caracteristicile structurale ale arhitecturilor SMP din clasa MIMD grupate în subtaxonii MIMD-T şi MIMD-L.

Reamintim că, în funcție de organizarea memoriei, sistemele din subtaxonul MIMD-T sunt cu *memorie partajată centralizată* (*centralized shared-memory*) în timp ce acelea din MIMD-L sunt cu *memorie partajată distribuită* (*distributed shared-memory*).

SMP-MP se caracterizează prin faptul că au o singură memorie principală ce are un timp de acces uniform pentru fiecare procesor. Din acest punct de vedere ele sunt maşini cu *accces la memorie uniform*. (*uniform memory access - UMA*).

Structura de bază a unui SMP-MP este prezentată în fig. 3.13 Se remarcă faptul că mai multe subsisteme procesor - cache împart aceeaşi memorie fizică, uzual conectate printr-o magistrală.

SMP-MP sunt maşini cu memorie fizic distribuită. Distribuirea memoriei este necesară pentru a se putea conecta un număr mare de procesoare ce reclamă o lățime de bandă mare.

Distribuirea memoriei printre noduri oferă beneficii majore. Primul, este o cale cu costuri reduse pentru a scala memoria, dacă cele mai multe dintre accese sunt la memoria locală din nod. Al doilea, se reduce latența pentru accese la memoria locală.

Dezavantajul major îl constituie faptul că procedura de comunicare între procesoare devine mai complexă.

Arhitectura de bază a unui sistem SMP-MP este prezentată în fig. 3.13

În aceste arhitecturi nodurile de comunicație pot conține unul sau mai multe procesoare împreună cu memoria şi circuitele de interfață cu RIN. Se pot crea *grupuri* (*cluster*) de procesoare conectate pe mai multe niveluri de RIN.

Diferențele arhitecturale majore ce se disting printre maşinile cu memorie distribuită constau în modul în care se produce comunicarea şi arhitectura logică a memoriei distribuite.

Există două abordări arhitecturale alternative în ceea ce priveşte comunicarea datelor între procesoare.

Prima constă în faptul că memoriile fizic separate pot fi adresate ca şi un singur spațiu de adresă logic partajat, ceea ce înseamnă că orice procesor poate accesa orice locație de memorie, dacă condițiile de acces sunt îndeplinite.

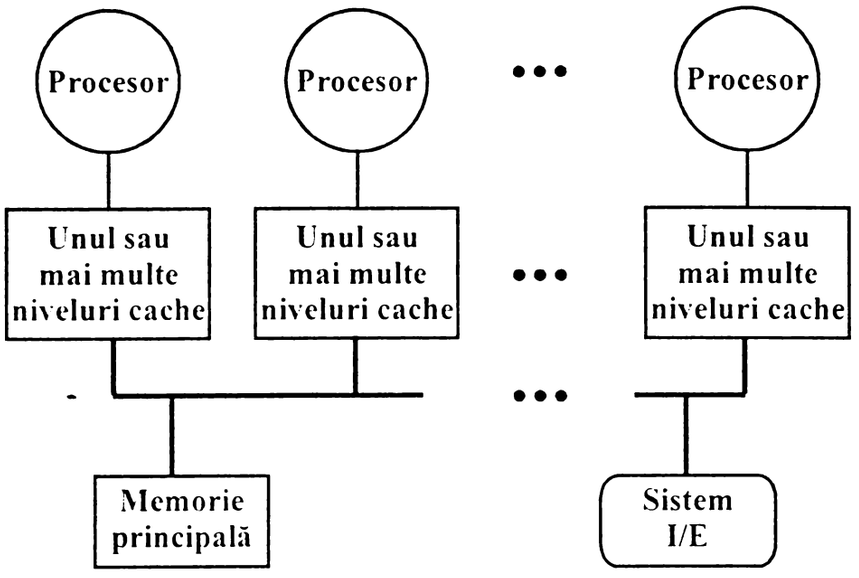


Fig. 3.13Structura de bază a unui SMP-MP

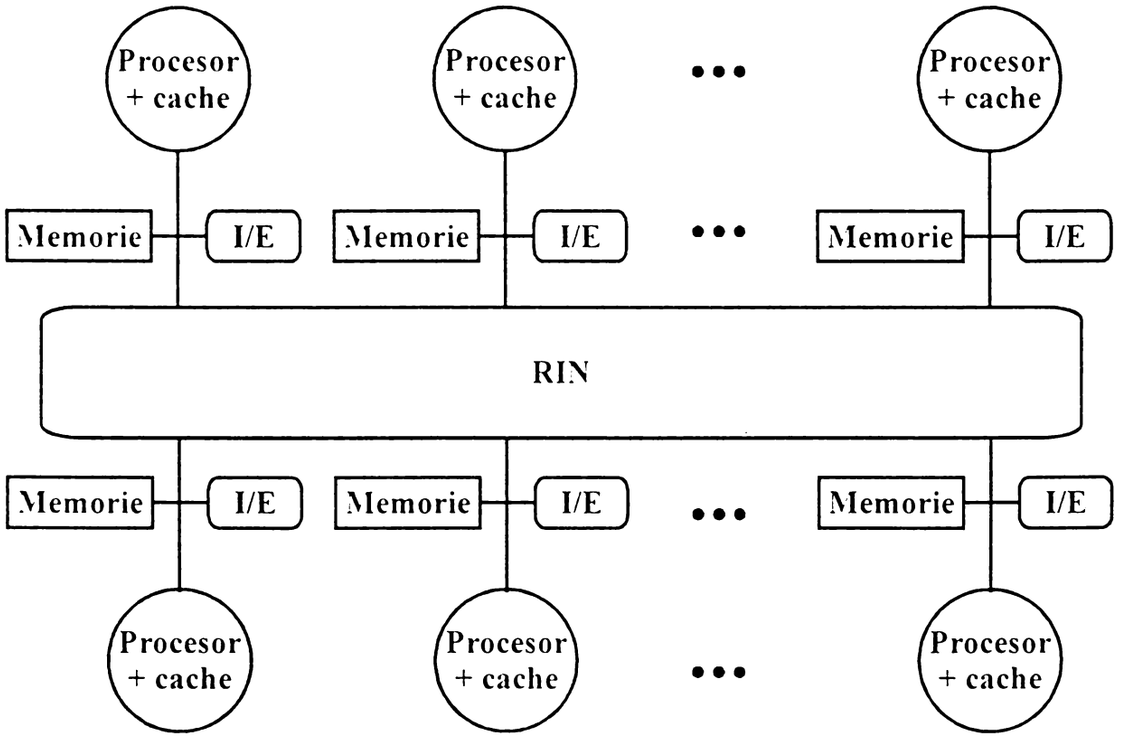


Fig. 3.14 Arhitectura de bază pentru un SMP-TM, cu memorie partajată distribuită

Denumirea de *memorie partajată* se referă la faptul că *spațiul de adresă* este partajat: adică aceeași adresă fizică solicitată de două procesoare se referă la aceeași locație în memorie. Memoria partajată nu înseamnă că există o singură memorie centrală. Prin

urmare sistemele cu memorie partajată pot fi cu *acces uniform la memorie* (*uniform memory access - UMA*) sau cu *acces neuniform la memorie* (*non - uniform memory access - NUMA*).

Soluția alternativă pentru comunicarea datelor o oferă spațiul de adresă ce constă din spații de adresă private multiple ce sunt logic disjuncte şi nu pot fi adresate de către un procesor de la distanță. La aceste maşini, aceeaşi adresă fizică în două procesoare diferite se referă la două locații diferite în două memorii diferite. Fiecare modul procesor - memorie este un calculator separat şi aceste sisteme se mai numesc *multicalculatoare*. Sunt arhitecturile clasice pentru SMP-TM !

În paragraful 3.1 s-au studiat performanțele RIN.

În acest paragraf ne vom focaliza eforturile pentru evidențierea performanțelor oferite de memoriile cache în diferite configurații.

În literatură [HP 96] se indică mai multe procedee de-a evalua performanțele diferitelor arhitecturi de SMP. De remarcat că există multe aplicații ale calculului paralel dintre care amintim : transformarea Fourier rapidă (FFT), descompunerea LU, aplicații ale algoritmului Barnes-Hut pentru rezolvarea problemelor legate de evoluția galaxiei, tehnici Gauss-Seidel pentru rezolvarea ecuațiilor diferențiale eliptice etc.

O caracteristică cheie în determinarea performanțelor programelor paralele îl constituie raportul dintre calcul şi comunicare.

Dacă raportul este mare înseamnă că aplicația are o mulțime de calcule pentru fiecare dată comunicată.

Este situația ideală. Într-un mediu de procesare paralelă se urmăreşte tot timpul cum se modifică raportul calcul/comunicare atunci cînd se adaugă noi procesoare, se măreşte dimensiunea problemei sau se manifestă ambele tendințe.

În TABEL IV se prezintă modul de evoluție al raportului calcul/comunicare în funcție de diferite tipuri de aplicații.

Se notează :

p - numărul de procesoare;

n - mărimea setului de date.

TABEL IV. Scalarea componentei de calcul, a celei de comunicare şi a raportului calcul/comunicare

Aplicaţie	Scalarea calculului	Scalarea comunicării	Scalarea calcul/comunicare
FFT	$\frac{n \log n}{p}$	$\frac{n}{p}$	$\log n$
LU	$\frac{n}{p}$	$\frac{\sqrt{n}}{\sqrt{p}}$	$\frac{\sqrt{n}}{\sqrt{p}}$
Barnes	$\frac{n \log n}{p}$	$\frac{\sqrt{n(\log n)}}{\sqrt{p}}$	$\frac{\sqrt{n}}{\sqrt{p}}$
Ocean	$\frac{n}{p}$	$\frac{\sqrt{n}}{\sqrt{p}}$	$\frac{\sqrt{n}}{\sqrt{p}}$

3.2.1 Arhitecturi cu memorie partajată centralizată

Aceste arhitecturi au dominat perioada anilor '80 datorită evoluţiei spectaculoase a circuitelor VLSI în general şi a microprocesoarelor în special.

Caracteristica principală - din punct de vedere al organizării ierarhiei memoriilor - constă în plasarea nivelului superior format din cache-uri pe aceleaşi module cu procesoarele şi a nivelului inferior - memorie RAM principală - modul (e) separat (e).

Arhitectura SMP-MP suportă plasarea în memorii cache atât a *datelor private* cât şi a celor *partajate*. Datele private sunt utilizate de un singur procesor în timp ce datele partajate sunt utilizate de mai multe procesoare, asigurându-se comunicarea între procesoare prin citiri şi scrieri ale datelor partajate. Când o dată privată este plasată în memoria cache, locaţia sa *migrează* în cache, reducându-se timpul de acces mediu atât cât o solicită lăţimea de bandă a memoriei.

Deoarece nici un alt procesor nu utilizează aceste date private, comportamentul programului este identic cu acela dintr-un uniprocesor.

Când datele partajate sunt plasate în cache, valoarea partajată trebuie să fie *replicată* în mai multe memorii cache. În plus faţă de reducerea latenţei accesului şi a lăţimii de bandă a memoriei, această replicare de asemenea furnizează o reducere a conflictelor ce pot apărea în procesul de citire a datelor partajate de către mai multe procesoare simultan.

Menționăm că plasarea în memorii cache a datelor partajate introduce problema *coerenței memoriei cache*.

Performanțelor arhitecturilor cu memorie partajată sunt în relație directă cu tehnicile de menținere a coerenței utilizate în acestea.

Introducerea memoriilor cache într-o ierarhie de memorii cauzează apariția problemei coerenței pentru operații de I/E deoarece viziunea asupra memorie prin cache poate fi diferită de viziunea asupra memoriei obținută prin subsisteme de I/E.

Figura 3.15 ilustrează un exemplu simplu de apariție a incoerenței.

Timp	**Eveniment**	**Conținutul cache pentru CPU A**	**Conținutul cache pentru CPU B**	**Conținutul memoriei la adresa X**
0				1
1	CPU A citeşte X	1		1
2	CPU B citeşte X	1	1	1
3	CPU A scrie 0 în X	0	1	0

Fig. 3.15 Problema coerenței cache pentru o locație de memorie, citită şi scrisă de două procesoare

Reamintim că toate operațiile de citire/scriere în memoria partajată situată pe nivelul inferior în cadrul unei ierarhii de memorii se fac prin intermediul cache-urilor plasate pe nivelul superior, logic şi fizic amplasat lângă procesoare.

Un sistem de memorie este coerent dacă :

1. O operație de citire efectuată de un procesor P_i, la o locație X ce urmează după o scriere de către P_i în X, fără să existe alte scrieri ale lui X de către alt procesor ce să apară între scrierea şi citirea efectuate de P_i întotdeauna returnează valoarea scrisă de P_i.
2. O citire efectuată de către un procesor P_i la o locație X ce urmează o scriere efectuată de alt procesor P_j în X, returnează valoarea scrisă dacă citirea şi scrierea sunt separate suficient şi nu apar alte scrieri în X între cele două accese.
3. Scrierile în aceeaşi locație de memorie sunt *serializate* : aceasta înseamnă că două scrieri în aceeaşi locație efectuate de către oricare alte două procesoare sunt privite în aceeaşi ordine de către toate procesoarele.

Modelul de consistență a memoriei oferă informații despre momentul şi condițiile în care o valoare scrisă trebuie să fie văzută de către utilizator.

Coerența şi *consistența* sunt complementare :

Coerența defineşte comportamentul scrierilor şi citirilor în aceeaşi locație de memorie, în timp ce *consistența* defineşte comportamentul citirilor şi scrierilor în raport cu accesele la alte locații de memorie.

În capitolul 2, paragraful 2.3, sunt descrise tehnici şi scheme de bază pentru întărirea coerenței.

În acest capitol vom dezvolta tehnici şi scheme de asigurare a coerenței la memoriile cache (SAC-MCH) specifice arhitecturilor cu memorie partajată.

A. Scheme de asigurare a coerenței memoriilor cache (SAC-MCH)

Într-un SMP-MP cu coerență asigurată, memoriile cache asigură atât *migrarea* cât şi *replicarea* datelor partajate. Memoriile cache coerente asigură migrarea, deoarece o dată poate fi deplasată într-un cache local şi utilizată apoi într-o manieră transparentă; această procedură reduce latența la accesul unei date partajate ce este alocată unui procesor fizic situat la distanță. De asemenea MCH asigură replicarea pentru datele partajate ce sunt simultan citite, deoarece MCH produc o copie a datei partajate în memoria cache locală (MCH – L). Replicarea reduce atât latența accesului cât şi conflictul la o dată partajată ce trebuie citită.

Migrarea şi replicarea sunt subiecte critice pentru stabilirea criteriilor de performanță în accesarea datelor partajate.

Prin urmare în SMP-MP de talie redusă se adoptă soluții hardware prin introducerea de protocoale de menținere a coerenței.

În literatură [HP 96] sunt descrise protocoale specifice pentru asigurarea coerenței memoriilor cache la SMP-MP.

Protocoalele de menținere a coerenței la memoriile cache (*cache - coherence protocols*) utilizează tehnica trasării stării pentru orice partajare a unui bloc de date.

În literatură [HP 96], [HP 94], [SLM 89] etc, se inventariază mai multe protocoale dar ele pot fi grupate în două clase ce utilizează diferite tehnici de trasare a stării de partajare.

Aceste sunt :

- *Bazate pe director* (*directory - based*) - starea de partajare a unui bloc de memorie este menținută într-o locație denumită *director*; este tipică pentru arhitecturile cu memorie partajată scalabile, adică SMP-MP.
- *Monitorizare* (*snooping*) - Fiecare cache ce are o copie a datelor provenind de la un bloc din memoria comună partajată are de asemenea o copie a stării de partajare a blocului. Nici o stare centralizată nu este păstrată. Toate controlerele atașate memoriilor cache *monitorizează* (*snoop*) magistrala partajată pentru a determina dacă au sau nu o copie a blocului ce este solicitat pe magistrală. Aceste protocoale bazate pe monitorizare sunt populare pentru SMP-MP deoarece aceste arhitecturi utilizează microprocesoare şi cache-uri atașate unei singure memorii partajate şi pot utiliza conexiuni fizice pre-existente de la magistrală la memorie.

Există două metode de a menține coerența.

Prima metodă constă în asigurarea că un procesor are un acces exclusiv la o dată înainte ca ea să fie scrisă.

Protocolul de invalidare a scrierii (*write invalidate protocol*) se bazează pe invalidarea altor copii la o operație de scriere. Este cel mai utilizat protocol atât pentru scheme bazate pe director cât şi pentru cele bazate pe monitorizare.

Accesul exclusiv asigură că nu există alte copii de citit sau de scris ale unei date atunci când apare scrierea : toate celelalte copii din cache-uri ale datei implicate sunt invalidate.

Acest protocol asigură coerența astfel. Se consideră o scriere urmată de-o citire de către alt procesor : deoarece scrierea solicită un acces exclusiv, orice copie deținută de procesorul efectuând citirea trebuie să fie *invalidată*. Astfel când apare citirea, copia lipseşte din cache şi se forțează căutarea unei noi copii a datei.

Pentru o scriere, este necesar ca procesorul ce scrie să aibă un acces exclusiv, prevenind orice alt procesor de-a ajunge în starea de-a scrie simultan. Dacă două procesoare intenționează să scrie aceeaşi dată simultan, unul dintre ele va câştiga cursa provocând invalidarea copiei celuilalt procesor. Pentru ca celălalt procesor să-şi completeze scrierea, este necesar să obțină o copie nouă a datei, care trebuie să conțină acum data actualizată.

Prin urmare acest protocol se bazează pe *serializarea scrierii*.

În fig. 3.16 se prezintă un exemplu a unui protocol de invalidare pentru o magistrală cu linie de monitorizare utilizând rescrierea în memoriile cache.

Activitate a procesorului	Activitate a magistralei	Conținut cache CPU A	Conținut cache CPU B	Conținut locație X din memorie
				0
CPU A citeşte X	Lipsă cache pentru X	0		0
CPU B citeşte X	Lipsă cache pentru X	0	0	0
CPU A scrie 1 în X	Invalidare pentru X	1		0
CPU B citeşte X	Lipsă cache pentru X	1	1	1

Fig. 3.16 Protocol de *invalidare* utilizat pe o magistrală
cu monitorizare pentru un singur bloc cache utilizând *rescrierea*

Protocolul de actualizare la scriere (*write update protocol, write broadcast protocol*) reprezintă o alternativă la protocolul de invalidare a scrierii şi constă în actualizarea tuturor copiilor deținute în memoriile cache ale datei ce este scrisă.

Pentru a se păstra sub control necesitățile dictate de lățimea de bandă, este util să se traseze dacă un cuvânt din cache este sau nu partajat, adică dacă este conținut în alte memorii cache. Dacă nu este conținut în alte MCH, atunci nu este necesar să se actualizeze oricare alte MCH.

În fig. 3.17 se prezintă un exemplu de *protocol de actualizare la scriere*. Acest protocol este mult mai puțin utilizat față de cel de invalidare a scrierii datorită diferențelor semnificative în performanțe.

Activitate a procesorului	Activitate a magistralei	Conținut cache CPU A	Conținut cache CPU B	Conținut locație X din memorie
				0
CPU A citeşte X	Lipsă cache pentru X	0		0
CPU B citeşte X	Lipsă cache pentru X	0	0	0
CPU A scrie 1 în X	Actualizare X	1	1	1
CPU B citeşte X		1	1	1

Fig. 3.17 Protocol de *actualizare la scriere* pe o magistrală
cu monitorizare pentru un singur bloc cache cu *rescriere*

Diferențele în performanțe dintre cele două protocoale apar datorită a trei caracteristici :

1. Scrieri multiple la acelaşi cuvânt, fără citiri ce să intervină, pretind transmisii multiple de scrieri într-un *protocol de actualizare*, dar doar o singură invalidare inițială într-un *protocol de invalidare de scriere*.
2. În cazul existenței de blocuri cache având cuvinte multiple, fiecare cuvânt scris într-un bloc cache pretinde o transmisie a semnalului de scriere într-un *protocol de actualizare*, în timp ce doar prima scriere la orice cuvânt din bloc necesită să se genereze o invalidare într-un *protocol de invalidare*. Un *protocol de invalidare* acționează asupra blocurilor cache, în timp ce un *protocol de actualizare* trebuie să acționeze asupra cuvintelor individuale.
3. Întârzierea dintre scrierea unui cuvânt într-un procesor şi citirea valorii scrise în alt procesor este uzual mai mică într-o schemă de actualizare la scriere, deoarece data scrisă este imediat actualizată în memoria cache a procesorului ce citeşte.

Cele mai utilizate protocoale sunt cele bazate pe *invalidare*.

B. Tehnici de implementare de bază

Cheia implementării unui protocol de invalidare într-un SMP-MP de talie redusă o constituie utilizarea magistralei pentru efectuarea operațiilor de invalidare.

Pentru a efectua o invalidare procesorul obține accesul la magistrală şi transmite adresa pentru a fi invalidată pe magistrală. Toate procesoarele monitorizează continuu magistrala urmărind adresele. Procesoarele verifică dacă adresa de pe magistrală este în cache-urile proprii. În caz afirmativ, data corespunzătoare din cache este invalidată. Semnalizarea accesului – asigurată de către magistrală – de asemenea forțează serializarea scrierilor, deoarece când două procesoare concurează pentru a scrie în aceeaşi locație, unul trebuie să obțină accesul la magistrală înaintea celuilalt. Primul procesor ce a câştigat competiția va provoca invalidarea copiei celuilalt procesor, obligând serializarea scrierilor. O consecință a acestei scheme este că o scriere la o dată partajată nu poate fi efectuată până când nu se obține accesul la magistrală.

Pentru un cache cu rescriere problema găsirii celei mai recente date este mai dificilă, deoarece cea mai recentă dată poate fi mai probabil în cache decât în memorie. Fiecare procesor monitorizează fiecare adresă plasată pe magistrală. Dacă un procesor

constată că are o copie viciată a blocului din cache solicitat, el furnizează acest bloc ca şi răspuns la o solicitare de citire şi cauzează abandonarea accesului la memorie.

Deoarece MCH cu rescriere nu au nevoie de lățime de bandă semnificativă pentru memorie sunt preferate în SMP.

În literatură [HP 96] sunt descrise tehnici de implementare a *MCH multinivel cu rescriere* (*write - back multilevel caches*).

Un protocol de menținere a coerenței bazat pe magistrală se implementează uzual incorporând un controler cu stări finite în fiecare nod. Acest controler răspunde la solicitările procesoarelor şi ale magistralei, schimbând starea blocului cache selectat, utilizând magistrala pentru a avea acces la magistrală sau pentru a o invalida.

În fig. 3.18 se prezintă cereri generate de către un modul procesor-cache într-un nod cât şi acelea generate de magistrală. În acest protocol nu se face distincție între un succes la scriere (write hit) şi un eşec la scriere (write miss) la un bloc cache partajat. În ambele cazuri un astfel de acces este tratat ca şi un eşec la scriere. Când o astfel de scriere este plasată pe magistrală, oricare dintre procesoare cu copii ale blocului cache le invalidează.

Solicitare	**Sursă**	**Funcție**
Succes la citire	Procesor	Citeşte data în cache
Succes la scriere	Procesor	Scrie data în cache
Eşec la citire	Magistrală	Solicită data din cache sau din memorie
Eşec la scriere	Magistrală	Solicită data din cache sau memorie. Invalidează

Fig. 3.18 Mecanism de asigurare a coerenței cu solicitări generate de modulul procesor-cache

În fig. 3.19 se prezintă o diagramă de tranziție a unui automat cu stări finite pentru un singur bloc cache ce utilizează un protocol de invalidare la scriere şi un cache cu rescriere. Din rațiuni de simplitate cele trei stări ale protocolului sunt dublate pentru a reprezenta tranziții bazate pe solicitări din CPU (în partea stângă) în opoziție cu tranziții bazate pe solicitări din magistrală (partea dreaptă). Starea în fiecare nod reprezintă starea blocului selectat specificat de către procesor sau de către cererea provenind din magistrală. Există doar un automat cu stări finite per cache, cu stimuli provenind fie din CPU ataşat, fie din magistrală.

În fig.3.20 se prezintă combinația dintre diagramele prezentate anterior, pentru a forma o singură diagramă de stare pentru fiecare cache.

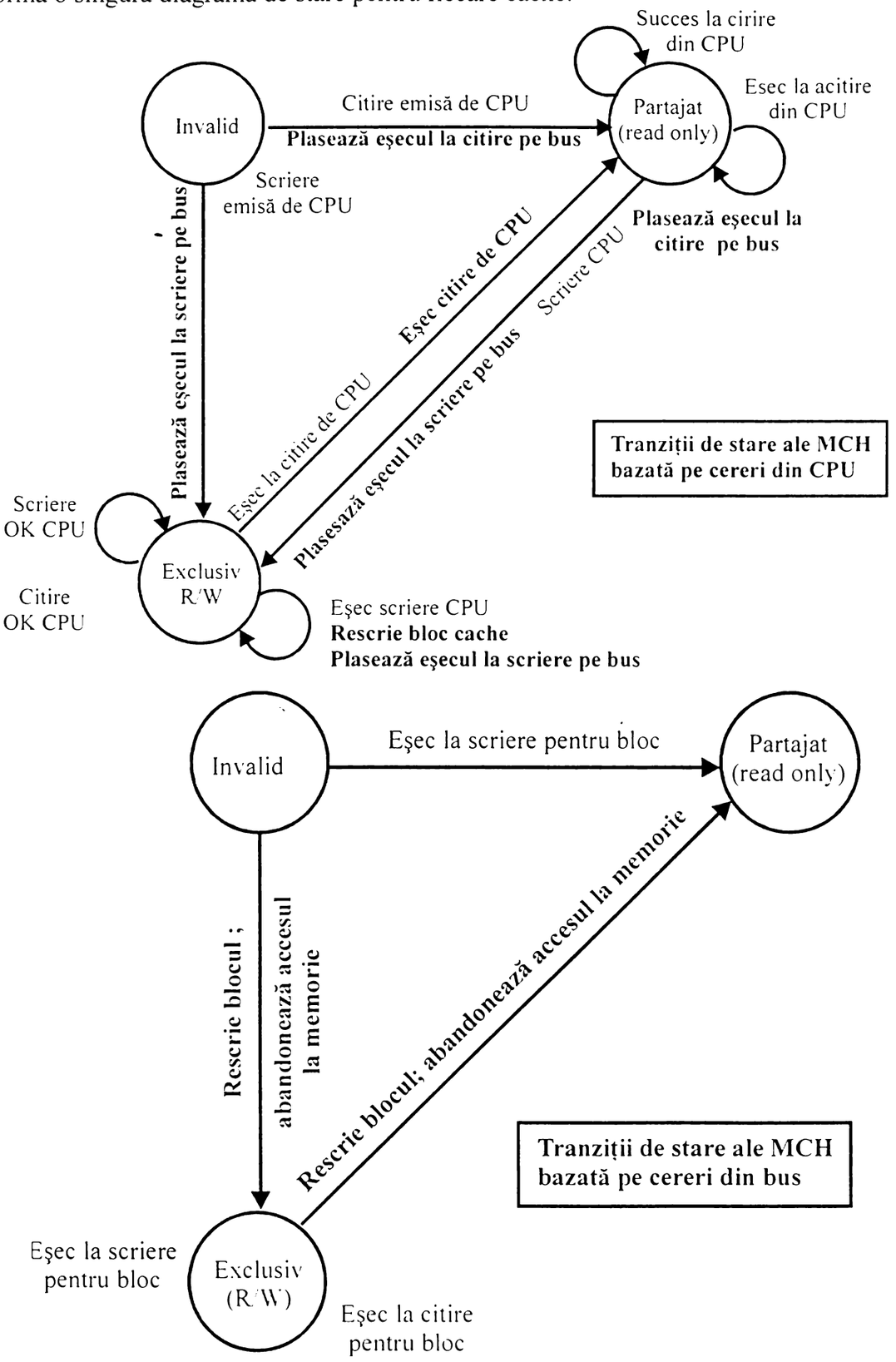


Fig.3.19 Un protocol cu invalidare la scriere pentru un cache cu rescriere prezentat ca şi un automat finit

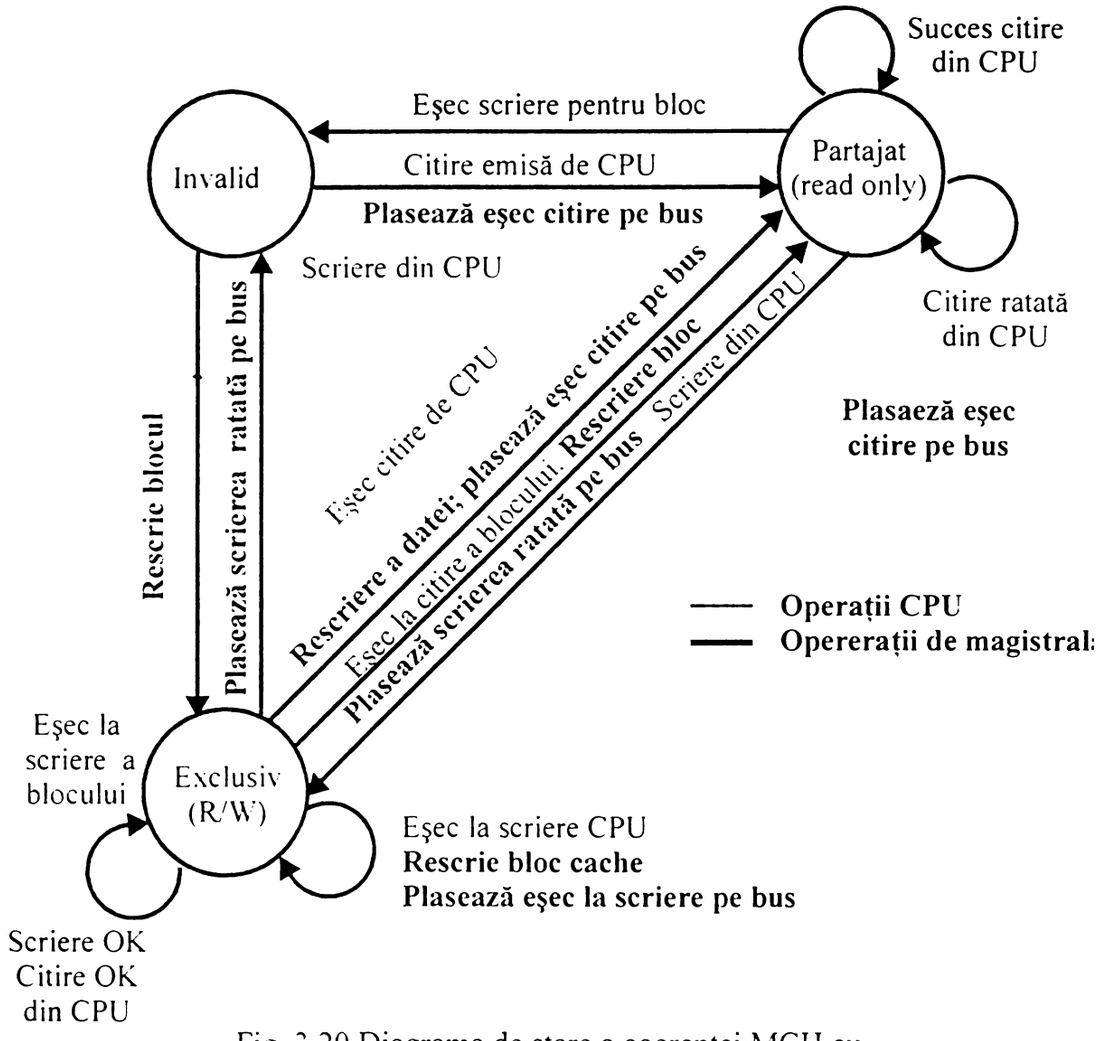


Fig. 3.20 Diagrama de stare a coerenței MCH cu

tranziții de stare induse de procesorul local şi activitățile magistralei

În implementarea acestor protocoale se presupune - ca ipoteză simplificatoare - că operațiile sunt *atomice*. De exemplu se consideră că ratările la scriere pot fi detectate, magistrala câştigată şi un răspuns obținut toate fiind operații indivizibile. În realitate nu este aşa. Modificând protocolul pentru a prezerva corectitudinea, acțiunile non atomice introduc posibilitatea ca protocolul să se poată *bloca* (*deadlock*), adică automatul finit ce-l implementează să atingă o stare de unde să nu mai poată continua.

În literatură [HP 96] se studiază performanțele MCH ce contribuie masiv la performanțele globale ale SMP. Se utilizează simulatoare pe care rulează *programe de etalonare* (*benchmark programms*).

Se evaluează *ratele de eşecuri* (*miss rates*) ca şi funcții de *numărul de procesoare* (*processor count*), *talia MCH* (*cache size*) şi *talia blocului de date* (*block size*).

Rata totală de eşec se descompune în *eşecuri datorate coerenței* (*coherence misses*) şi *eşecuri de uniprocesor normale* (*normal uniprocesor misses*).

Eşecurile de uniprocesor normale constau din *eşecuri datorită capacităţii* (*capacity misses*), *eşecuri datorită conflictului* (*conflict misses*) şi *eşecuri obligatorii* (*compulsory misses*).

De obicei aceste eşecuri se etichetează ca fiind *eşecuri datorate capacităţii* deoarece aceasta este dominantă în programele de etalonare.

În literatură [HP 96] se prezintă comparativ fiecare din aceşti parametrii luând în considerare mai multe tipuri de aplicaţii pentru un parametru.

Se va prezenta doar o aplicaţie - analiză Fourier rapidă (FFT) - ce constă în prelucrarea a 1 milion de puncte de date reprezentate în complex.

Se observă că :

- rata de eşec datorită coerenţei creşte cu numărul de procesoare; (fig. 3.21);
- ratele de eşec datorită coerenţei şi capacităţii scad atunci când talia memoriei cache creşte; (fig.3.22);
- ratele de eşec scad semnificativ atunci când talia blocului de memorie creşte. (fig.3.23).

În fig. 3.24 se prezintă variaţia traficului pe magistrală în funcţie de talia blocului. El creşte odată cu talia blocului.

De asemenea rata de eşec scade cu creşterea dimensiunii memoriei cache.

În fig. 3.25 se prezintă evoluţia componentelor ratei de eşec în funcţie de talia memoriei cache şi, respectiv, în funcţie de talia blocului.

C. Concluzii referitoare la performanţele schemelor de asigurare a coerenţei MCH utilizând monitorizarea

Traficul datorat coerenţei poate introduce comportamente noi în sistemul de memorie ce nu răspund uşor la modificările taliei memoriei cache sau a taliei blocului ce sunt normal utilizate pentru a ameliora performanţele MCH la uniprocesoare.

În cadrul procesării paralele cerinţele de menţinere a coerenţei sunt semnificative dar nu în mod exagerat.

În cadrul multiprogramării, utilizatorul şi sistemul de operare (SO) evoluează foarte diferit.

În porţiunea de SO, combinaţiile componentelor obligatorii şi respectiv datorate capacităţii în ceea ce priveşte rata de eşec sunt mult mai semnificative. Aceste rate de eşec afectează performanţele CPU depinzând de restul sistemului de memorie, incluzând latenţa şi lăţimea de bandă a magistralei şi a memoriei.

Graficele sunt prezentate în fig.3.28, 3.29, 3.30.

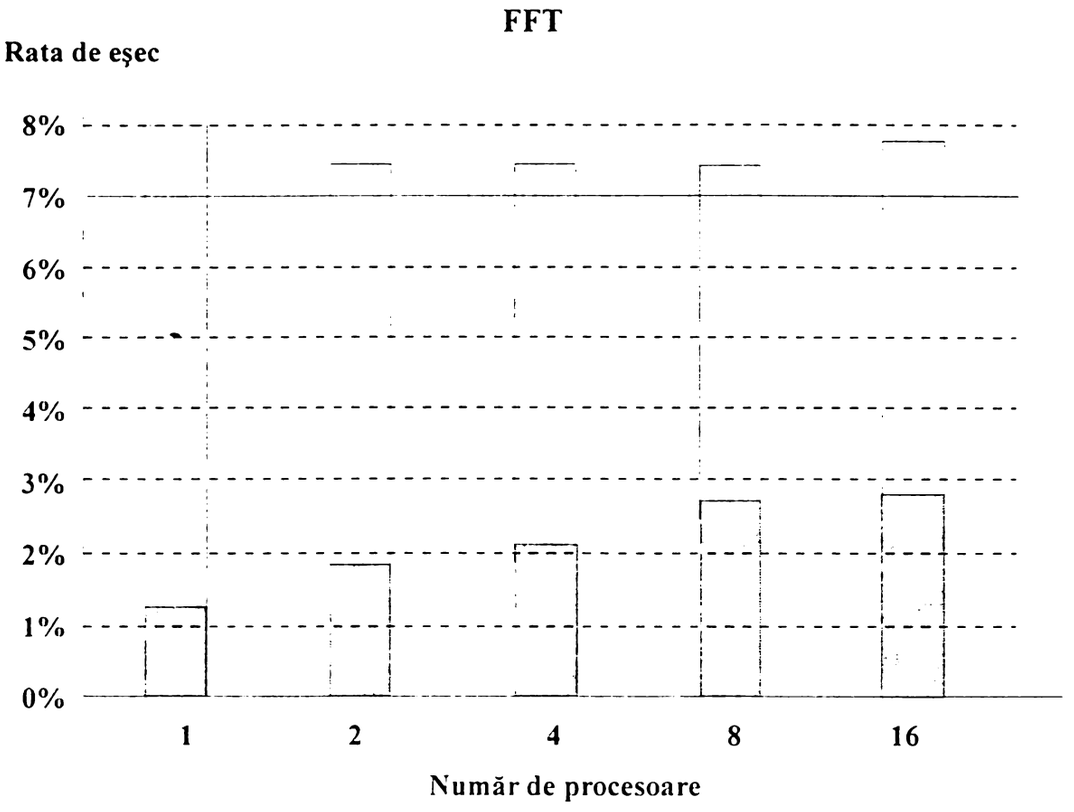


Fig. 3.21 Variația ratelor de eşec în funcție de numărul de procesoare

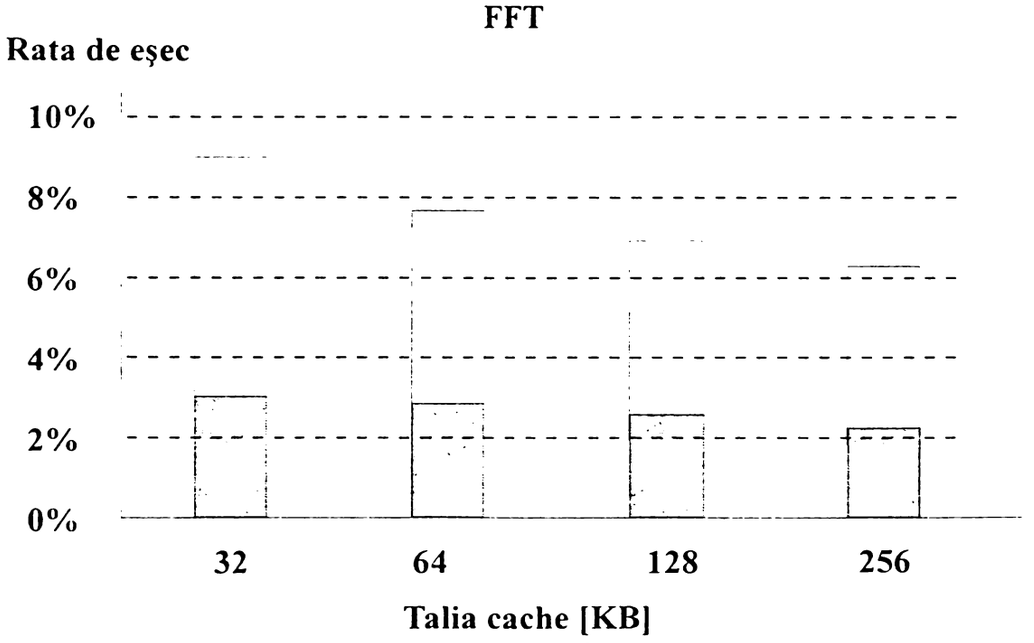


Fig. 3.22 Variația ratei de eşec funcție de talia MCH

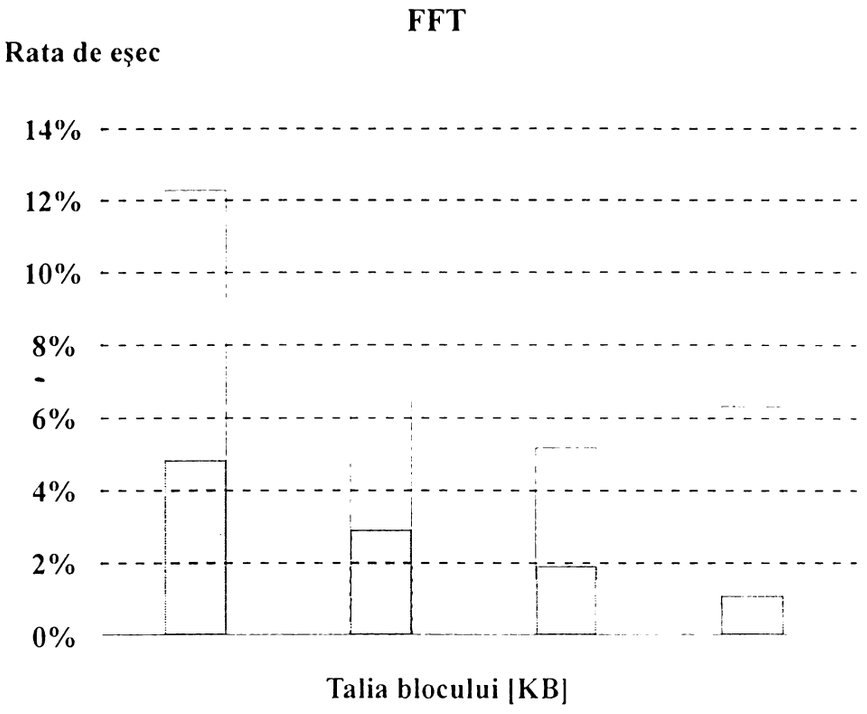


Fig. 3.23 Rata de eşec în funcție de talia blocului

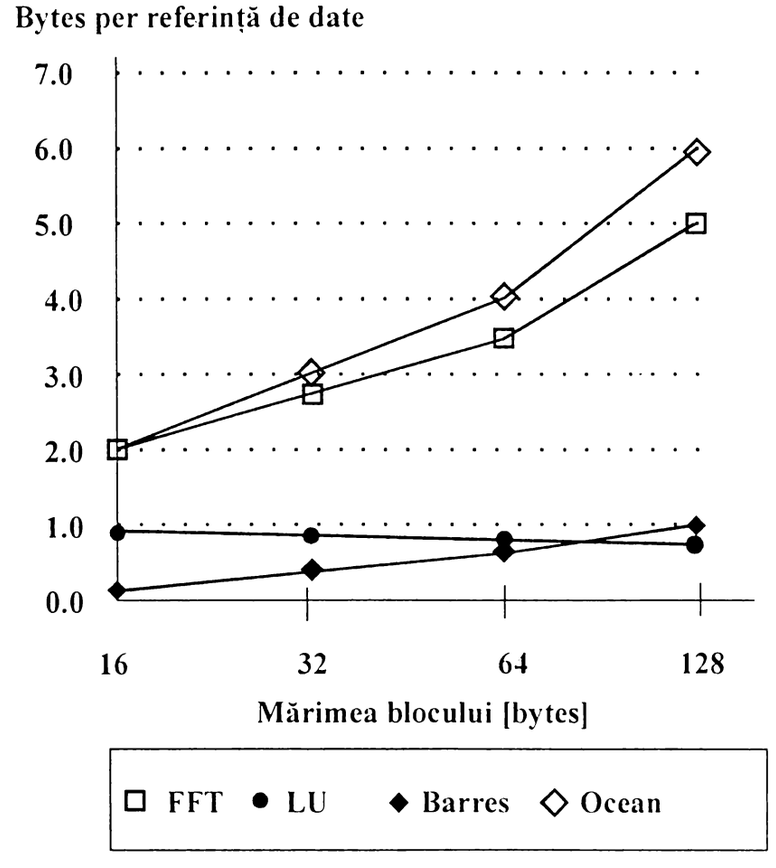


Fig. 3.24 Traficul de magistrală în raport cu talia blocului

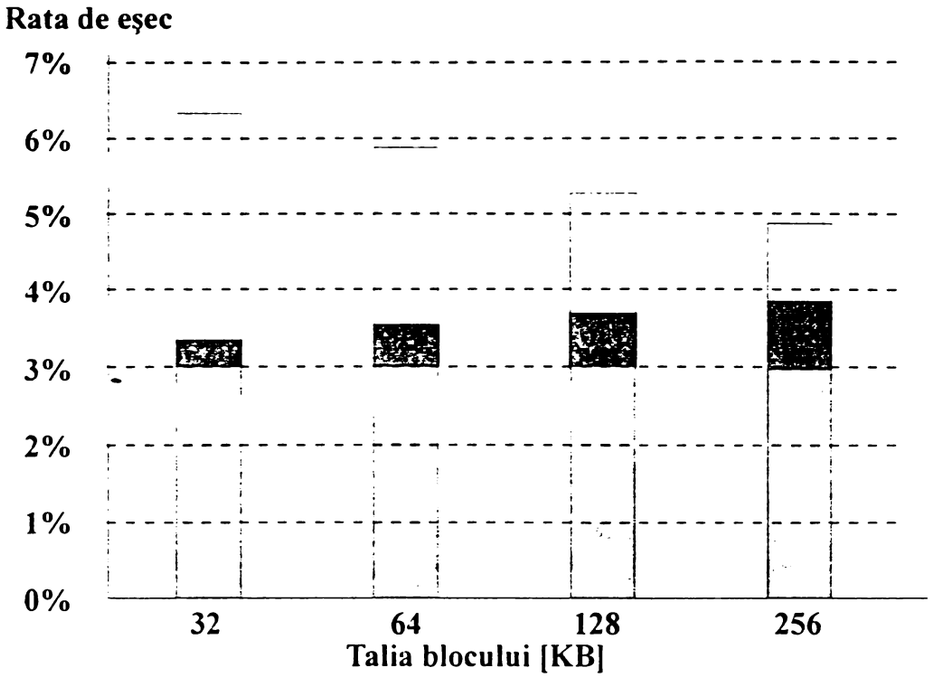


Fig. 3.25 Componentele ratei de eşec a datelor funcție de talia MCH

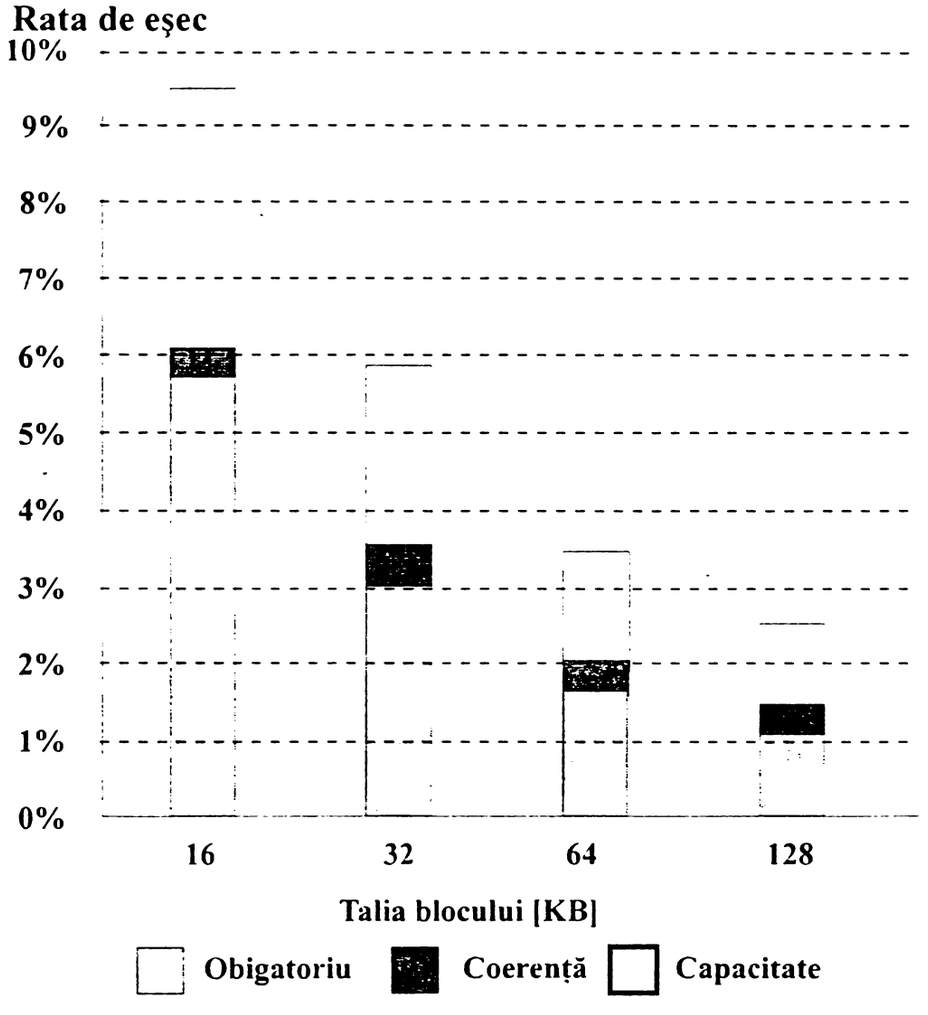


Fig. 3.26 Componentele ratei de eşec funcție de talia cache-ului şi a blocului

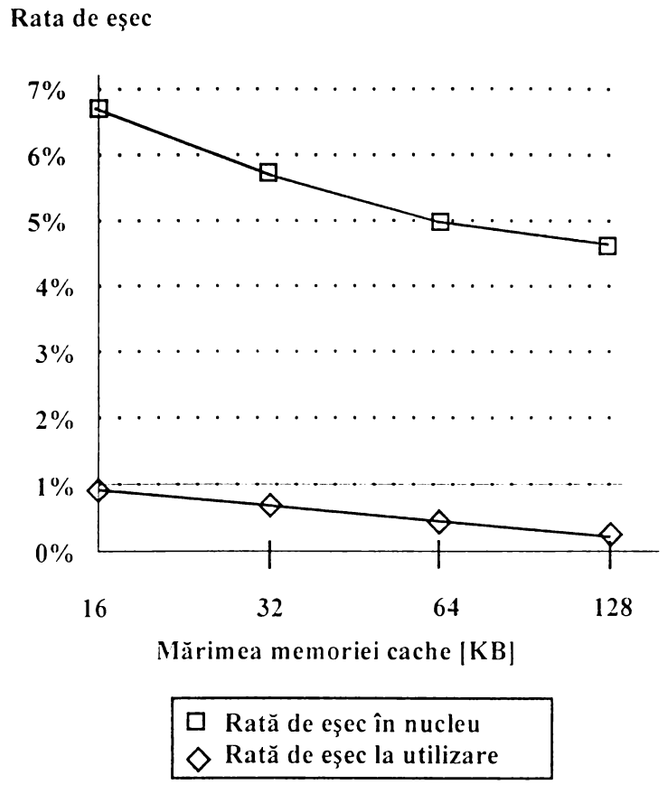


Fig. 3.27 Rata de eşec funcție de talia cache

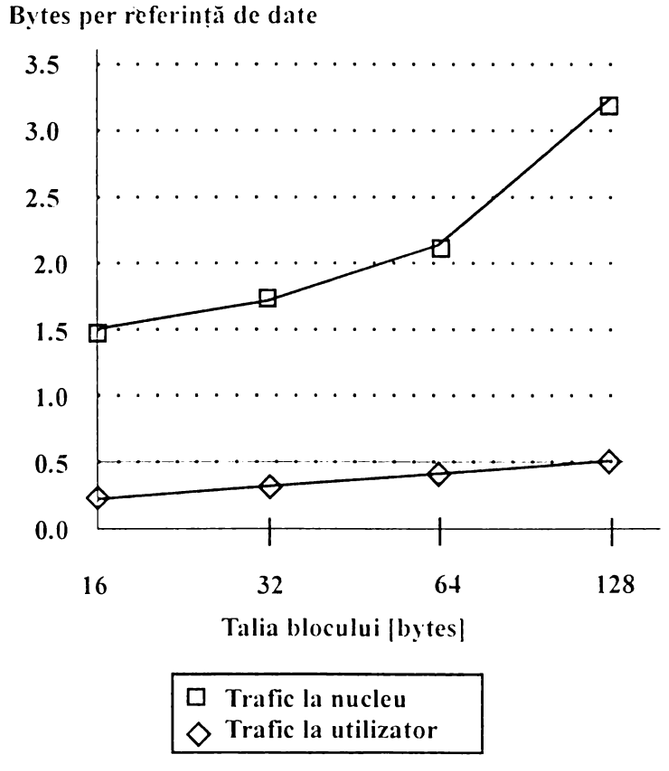


Fig. 3.28 Evoluția traficului în porțiunea de SO şi utilizator în cazul programelor paralele

3.2.2 Arhitecturi cu memorie partajată distribuită

O maşină scalabilă ce suportă o memorie partajată ar putea să aibă exclusiv sau inclusiv coerența memoriei cache. Din punct de vedere hardware, cea mai simplă situație este aceea în care coerența memoriei cache este exclusivă. În astfel de sisteme memoria este distribuită printre noduri şi toate nodurile sunt interconenctate printr-o RIN. Accesul poate fi fie local fie la distanță, strategia fiind decisă de un controler dispus în fiecare nod, pe baza adresei, şi care se referă la faptul că data este rezidentă în memoria locală sau în memoria de la distanță.

În ultimul caz, se trimite un mesaj controlerului dispus în memoria de la distanță, pentru accesarea datei.

Aceste sisteme sunt dotate cu MCH, dar pentru a preveni probleme legate de coerență, data partajată este marcată ca fiind indispensabilă pentru plasarea în MCH şi dacă data privată este menținută în MCH.

Evident, sistemul de operare poate explicit să cache-eze valoare datei partajate copiind data din porțiunea partajată a spațiului de adresă în porțiunea privată locală a spațiului de adresă cache-at. Coerența este apoi controlată prin tehnici software.

Avantajul unui astfel de mecanism este că necesită un suport hardware redus.

Există câteva dezavantaje în această abordare.

În primul rând, mecanismele de compilare pentru asigurarea coerenței cache prin mijloace software, de-o manieră transparentă, sunt foarte limitate.

Dificultatea majoră o constituie faptul că algoritmii de menținere a coerenței, bazați pe tehnici software, trebuie să fie conservativi : fiecare bloc ce *poate* fi partajat trebuie să fie tratat ca şi când *este* partajat. Această abordare conduce la cheltuieli excesive de asigurare a coerenței deoarece compilatorul nu poate prezice partajarea actuală cu acuratețe.

În al doilea rând, fără asigurarea coerenței MCH, sistemele pierd avantajul de-a fi capabile să găsească şi să utilizeze cuvinte multiple într-un singur bloc cache.

În al treilea rând, mecanismele pentru tolerarea latenței ca şi readucerea din memorie, sunt mult mai utile atunci când pot aduce cuvinte multiple, ca şi un bloc cache, şi unde data adusă rămâne coerentă.

Datorită motivelor prezentate mai sus, coerența MCH este o cerință acceptată în SMP de talie mică. Pentru arhitecturi de-o talie superioară, există diferite tehnici de extindere a modelului de memorie cache partajată şi coerentă.

Prin urmare magistrala poate fi înlocuită cu o RIN mai scalabilă şi memoria poate fi distribuită astfel încât lățimea de bandă să fie şi ea scalată.

Schemele de asigurare a coerenței prin monitorizare (*snooping coherence scheme*) se caracterizează prin absența scalabilității. Un protocol de monitorizare necesită comunicarea cu toate cache-urile la fiecare eşec cache, incluzând scrieri a datelor partajate potențial. Absența oricărei structuri de date centralizate referitoare la starea cache-urilor este avantajul fundamental al schemei bazate pe monitorizare, deoarece îi permite să fie ieftină.

Se pot construi arhitecturi scalabile cu memorii partajate ce includ coerența cache. Un protocol alternativ protocolului bazat pe monitorizare îl reprezintă cel *bazat pe director* (*directory - based*). Un *director* păstrează starea fiecărui bloc ce poate fi cachat. Informația conținută în director include marcarea situațiilor de tipul : există copii ale blocului în cache, sunt viciate, etc.

Implementările cu directori existente asociază o intrare în director cu fiecare bloc de memorie, în protocoale tipice, cantitatea de informație este proporțională cu produsul dintre numărul de blocuri de memorie şi numărul procesoarelor.

Pentru evitarea situației când directorul devine un factor de strangulare a traficului, intrările în director pot fi distribuite în modulele de memorie astfel încât diferite accese la director pot fi distribuite la diferite locații, la fel ca şi în situația în care diferite solicitări de acces la memorie sunt direcționate la diferite memorii.

Un director distribuit se caracterizează prin faptul că starea partajată a unui bloc este întotdeauna într-o singură locație cunoscută. Tocmai această proprietate este aceea care permite protocolului de menținere a coerenței să evite *difuzarea* (*broadcast*).

În fig.3.29 se prezintă arhitectura unui SMP cu memorie distribuită cu directori ataşați fiecărui nod. Este schema din fig. 3.14 detailată!

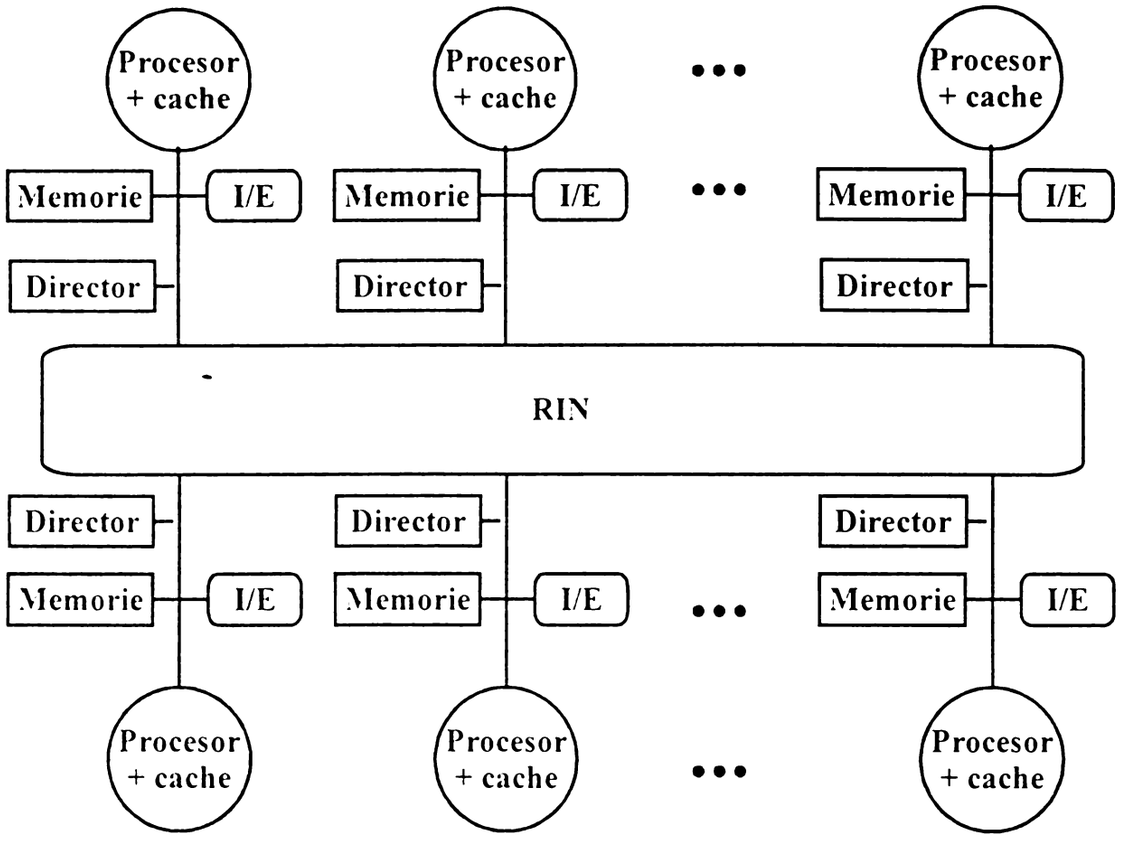


Fig. 3.29 Arhitectura SMP - TM cu directori ataşaţi fiecărui nod
pentru a implementa coerenţa cache

Din fig. 3.29 se observă că fiecare director este responsabil cu marcarea memoriilor cache ce partajează adresele de memorie ale porţiunii de memorie din nod. Directorul poate comunica cu procesorul şi cu memoria prin intermediul unei magistrale comune, ca şi în figură, sau poate avea un port separat la memorie, sau poate fi o parte a unui controler central de nod prin care trec toate comunicaţiile intranod şi internod.

A. Protocoale de asigurare a coerenţei bazate pe director. Fundamente

Există două operaţii primare pe care un protocol bazat pe director trebuie să le implementeze:

- manipularea unui eşec la citire;
- manipularea unei scrieri la un bloc cache partajat.

Manipularea unui eşec la scriere într-un bloc partajat este o combinaţie dintre cele două prezentate anterior. Pentru a implementa aceste operaţii, un director trebuie să

traseze starea fiecărui bloc cache. Într-un protocol simplu, aceste stări ar putea fi următoarele:

- *Partajată* (*shared*) - Unul sau mai multe procesoare au blocul partajat şi valoarea în memorie trebuie să fie actualizată. La fel în toate memoriile cache.
- *Necache-ată* (*unchached*) - Nici un procesor nu are o copie a blocului cache.
- *Exclusivă* (*exclusive*) - Doar un procesor - identificat exact - are o copie a blocului cache şi blocul a fost scris, astfel că copie memoriei este perimată. Procesorul este denumit *proprietarul* blocului.

Suplimentar cu trasarea stării fiecărui bloc cache, trebuie marcate procesoarele ce au copii ale blocului când acesta este partajat, deoarece acestea trebuie invalidate la o scriere. Cea mai simplă cale pentru efectuarea acestei operații este de-a menține un vector binar (de unul sau mai mulți biți) pentru fiecare bloc de memorie. Când blocul este partajat, fiecare bit al vectorului indică dacă procesorul corespunzător are o copie a acestui bloc. Se poate de asemenea utiliza vectorul pentru menținerea traseului proprietarului blocului când blocul este într-o stare exclusivă. Din rațiuni de eficiență, se trasează starea fiecărui bloc cache la memoriile cache individuale.

Stările şi tranzițiile pentru un SMP sunt identice la fiecare cache cu cele utilizate pentru monitorizarea cache-urilor. Acțiunile la o tranziție sunt uşor diferite.

Se fac următoarle ipoteze simplificatoare :

- încercările de scriere a unor date non exclusive întotdeauna generează eşecuri la scriere şi procesoarele se blochează până când un acces se finalizează;
- interconectarea nu se poate utiliza ca şi un singur punct de arbitrare, o funcție realizată de magistrală în cazul monitorizării.
- toate mesajele trebuie să aibă răspunsuri explicite, deoarece RIN este orientată pe mesaje (magistrala este orientată pe tranzacții).

În TABEL V se prezintă tipurile de mesaje trimise între noduri. Nodul *local* este nodul ce generează o solicitare. Nodul *rezident* (*home*) este nodul unde sunt rezidente locația de memorie şi intrarea în director a unei adrese. Spațiul de adresă fizică este static distribuit astfel încât nodul ce conține atât memoria cât şi directorul pentru o adresă fizică dată este identificat.

Nodul *îndepărtat* (*remote*) este nodul ce are o copie a unui bloc cache, fie exclusiv sau partajat. Nodul *local* poate fi nod *rezident* şi vice versa. În ambele cazuri protocolul este acelaşi, prin urmare mesajele între noduri pot fi înlocuite prin tranzacţii în interiorul nodului ce sunt mai rapide.

În acest paragraf se presupune un model simplu de consistenţă a memoriei. Pentru a minimiza tipul mesajelor şi complexitatea protocolului se presupune că mesajele vor fi recepţionate şi procesate în aceeaşi ordine în care ele au venit. Această presupunere, greu de îndeplinit în practică, ne asigură că invalidările transmise de un procesor sunt imediat onorate.

TABEL V. Mesajele posibile transmise între noduri pentru a menţine coerenţa.

Tip	Sursă	Destinaţie	Conţinut	Funcţie
EC	CH-L	P A	P, A	P are un eşec la citire în A; Solicită data şi îl face pe P partener la partajarea la citire.
ES	CH-L	P A	P, A	P are un eşec la scriere în A; Solicită data şi îl face pe P proprietar exclusiv.
I	DR	CH-D	A	Invalidează o copiere partajată a datei la adresa A.
A	DR	CH-D	A	Aduce blocul la adresa A şi-l trimite la directorul lui de acasă; invalidează blocul în cache.
A/I	DR	CH-D	A	Aduce blocul la adresa A şi-l trimite la directorul lui de acasă; invalidează blocul în cache.
RVD	DR	CH-L	Data	Returnează o valoare a datei din memoria de acasă.
RD	CH-D	DR	A, Data	Rescrie o valoare de dată pentru adresa A.

Abrevierile sunt următoarele :

EC - eşec la citire (read miss);

ES - eşec la scriere (write miss);

I - invalidare;

A - aducere;

A/I - aducere/invalidare;

RVD - returnează validarea datei;

RD - rescrie data;

CH-L - cache local;

CH-D - cache la distanță;

DR - director "rezident";

P - procesor.

Un exemplu de protocol bazat pe director.

În literatură [HP 96] se prezintă mai multe exemple de protocoale bazate pe director. Stările de bază ale unui bloc cache într-un protocol bazat pe director sunt aceleaşi ca şi într-un protocol de monitorizare şi stările într-un director sunt similare cu acelea prezentate anterior.

În continuare se analizează diagrame simple de stare ce prezintă tranziții de stare pentru un bloc cache individual şi apoi se examinează diagrama de stare pentru intrarea din director corespunzătoare fiecărui bloc din memorie. Aceste diagrame de tranziție de stare nu reprezintă toate detaliile unui protocol de asigurare a coerenței.

Fig. 3.30 prezintă acțiunile protocolului la care răspunde un cache individual.

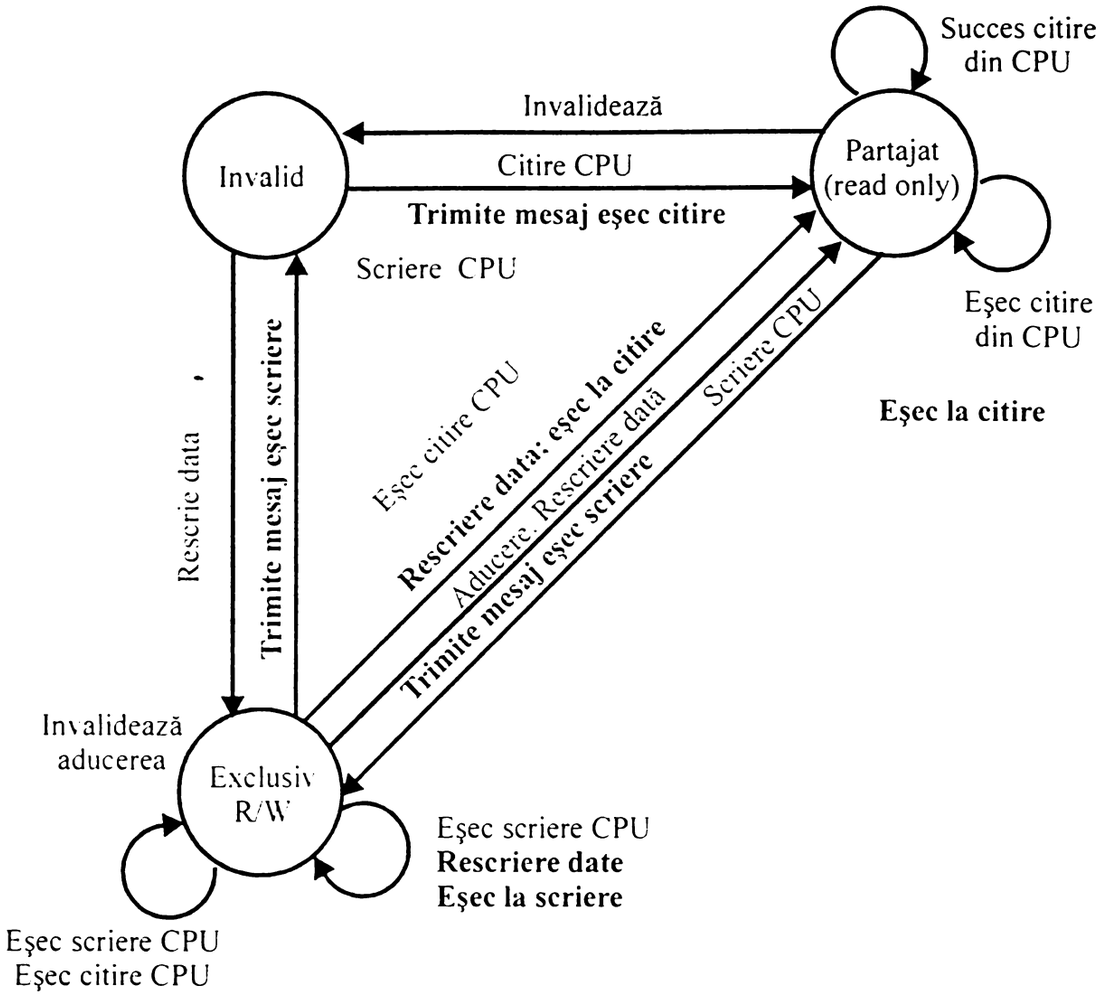


Fig. 3.30 Diagrama tranziţiilor de stare pentru un bloc cache individual
într-un sistem bazat pe director

Câteva remarci sunt necesare. Stările sunt identice cu acelea din protocolul de monitorizare şi tranzacţiile sunt foarte asemănătoare cu solicitări de invalidare şi de rescriere ce înlocuiesc eşecurile la scriere ce au fost difuzate pe magistrală. Se presupune că o încercare de a scrie un bloc cache partajat este tratată ca şi un eşec; în practică o astfel de tranzacţie poate fi tratată ca şi o solicitare a proprietarului şi poate fi livrată proprietarului fără solicitarea ca blocul cache să fie adus. Prin urmare, operaţia de eşec la scriere, ce a fost difuzată pe magistrală în schema cu monitorizare, este realizată cu operaţii de căutare a datei şi de invalidare ce sunt trimise selectiv de către controlerul directorului. Orice bloc cache trebuie să fie în starea exclusivă când este scris şi orice bloc partajat trebuie actualizat în memorie.

În protocolul bazat pe director, directorul implementează cealaltă jumătate a protocolului de coerență. Un mesaj transmis spre un director provoacă două tipuri diferite de acțiuni :

- actualizează starea directorului;
- trimite mesaje adiționale pentru satisfacerea solicitării.

Stările într-un director reprezintă cele trei stări standard pentru un bloc, dar pentru toate copiile cache ale unui bloc de memorie nu numai pentru un singur bloc cache. Blocul de memorie poate fi necache-at de către orice nod, cache-at în noduri multiple şi partajat sau cache-at în mod exclusiv cu posibilitatea de a fi scris într-un nod anume.

Figura 3.31 prezintă acțiunile luate la nivel de director în răspuns la mesajele recepționate. Directorul recepționează trei solicitări diferite : *eşec la citire*, *eşec la scriere* şi *rescriere a datei*.

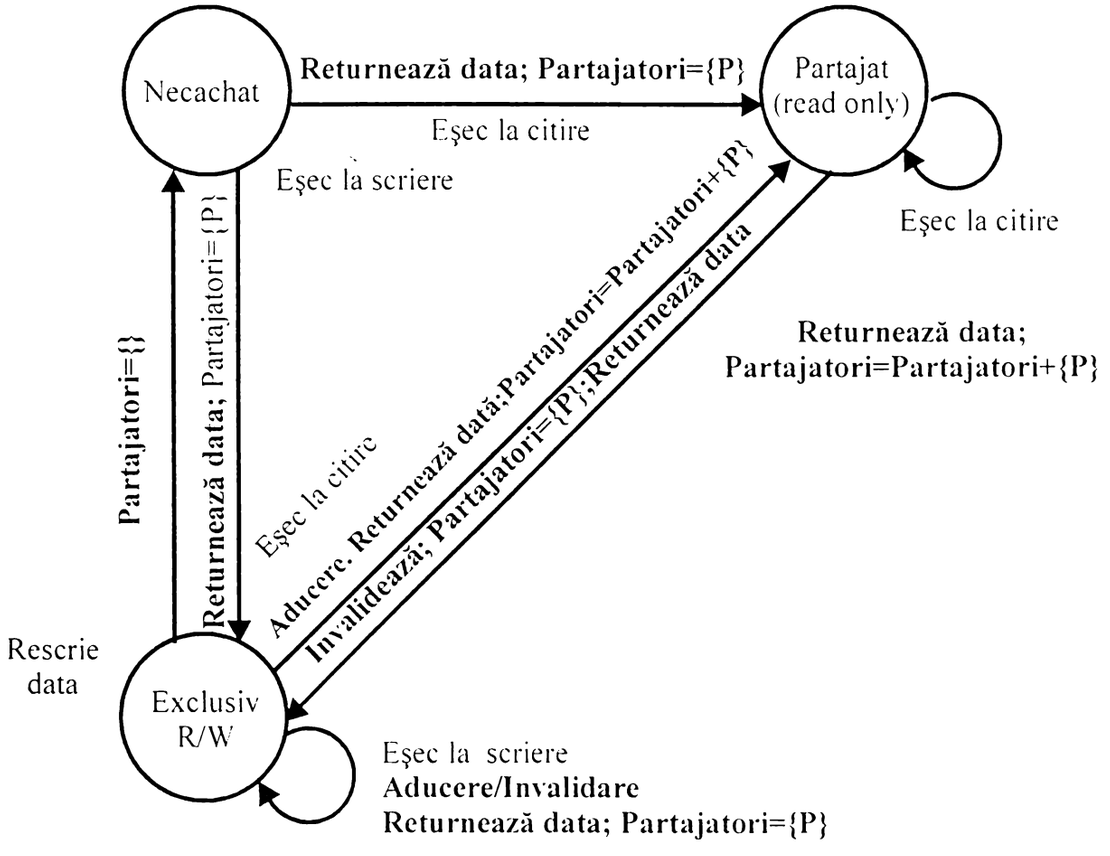


Fig. 3.31 Diagrama tranzițiilor de stare pentru director are aceleaşi stări şi structură ca şi diagrama de tranziție pentru o memorie cache individuală

Se remarcă existența așa numiților *partajatori* (*sharers*).

Aceştia sunt vectori binari ce efectuează trasarea seturilor de procesare ce au o copie a unui bloc.

Protocolul simplificat prezentat presupune că unele acțiuni sunt atomice, ca de exemplu solicitarea unei valori şi transmiterea ei altui nod.

Analizând solicitările primite şi acțiunile luate stare cu stare se constată că atunci când un bloc este într-o stare necachată următoarele solicitări sunt posibile :

- *Eşec la citire* - Procesorul solicitant a transmis data solicitată din memorie şi solicitatorul este declarat singurul nod ce partajează. Starea blocului este declarată *partajată*.
- *Eşec la scriere* - Procesorul solicitat a transmis valoarea şi devine *nod partajator*. Blocul este făcut *exclusiv* pentru a indica că doar copia validă este cachată. Partajatorii indică identitatea deținătorului. Când un bloc este în starea "*partajată*" valoarea din memorie este de actualizat, așa pot apare două cereri :
- *Eşec la citire* - Procesorul solicitant a transmis data solicitată din memorie şi procesorul solicitant este adăugat setului partajator.
- *Eşec la scriere* - Procesorul solicitant a transmis valoarea. Toate procesoarele din setul Partajatorilor au transmis mesaje de invalidare, şi setul Partajatorilor conține identitatea procesorului solicitant. Starea blocului este făcută "*exclusiv*".

Când blocul este în starea "*exclusiv*", valoarea curentă a blocului este menținută în memoria cache a procesorului identificat de către deținător astfel încât există trei posibilități de solicitări de director :

- *Eşec la citire* - Procesorul deținător a transmis un mesaj de aducere a datei ce provoacă tranziția stării blocului din cache-ul deținătorului în "*partajat*" şi determină deținătorul să transmită data către director, unde este scrisă în memorie şi retransmisă procesorului solicitant. Identitatea procesorului solicitant este adăugată la partajatori care încă mai conțin identitatea procesorului ce a fost deținător, el mai posedând o copie ce poate fi citită.

- *Rescrierea datei* - Procesorul solicitant înlocuieşte blocul şi prin urmare trebuie să-l rescrie. Copia memoriei se poate actualiza, blocul devine necach-at, şi setul de partajatori se videază.
- *Eşec la scriere* - blocul are un nou deținător. A fost transmis un mesaj către vechiul deținător, ce provoacă cache-ul să trimită valoarea blocului către director, de la care este transmis procesorul solicitant, ce devine noul deținător. Partajatorii sunt poziționați să identifice noul deținător şi starea blocului rămâne "*exclusiv*".

Diagrama de tranziție a stărilor din fig. 3.34 este simplificată.

În sistemele reale cele mai multe protocoale transmit data de la nodul deținător către nodul solicitant în mod direct. Astfel de optimizări nu duc la creşterea complexității dar repercursiunile se manifestă în alte părți ale sistemului.

B. Performanțe ale protocoalelor de asigurare a coerenței bazate pe director

Performanța unui sistem bazat pe director depinde de mulți dintre factorii ce influențează perfomanța maşinilor orientate pe magistrală (de ex. talia cache-ului, numărul de procesoare, talia blocului) cât şi de distribuția eşecurilor la diferite locații în ierarhia de memorie. Locația unei date solicitate depinde atât de alocarea inițială cât şi configurațiile partajate.

Analiza performanțelor se va efectua pe o memorie cache de 128 KB şi asupra unui bloc de 64 bytes.

În arhitecturile cu memorie distribuită, distribuția solicitărilor de acces la memorie între memoria locală şi cea de la distanță, reprezintă cheia performanței. Prin urmare eşecurile la memoria cache se vor separa în solicitări locale şi la distanță.

În literatură [HP 96] se analizează performanțele SMP-TM, bazate pe director, observând evoluția mai multor programe paralele specifice SMP. De exemplu programele FFT, LU, Barnes şi Ocean.

În acest paragraf vom analiza doar comportametul schemelor bazate pe director în cazul evoluției unui program FFT.

În fig. 3.32 şi 3.33 se prezintă dependența ratei de eşec de numărul de procesoare, respectiv de talia memoriei cache.

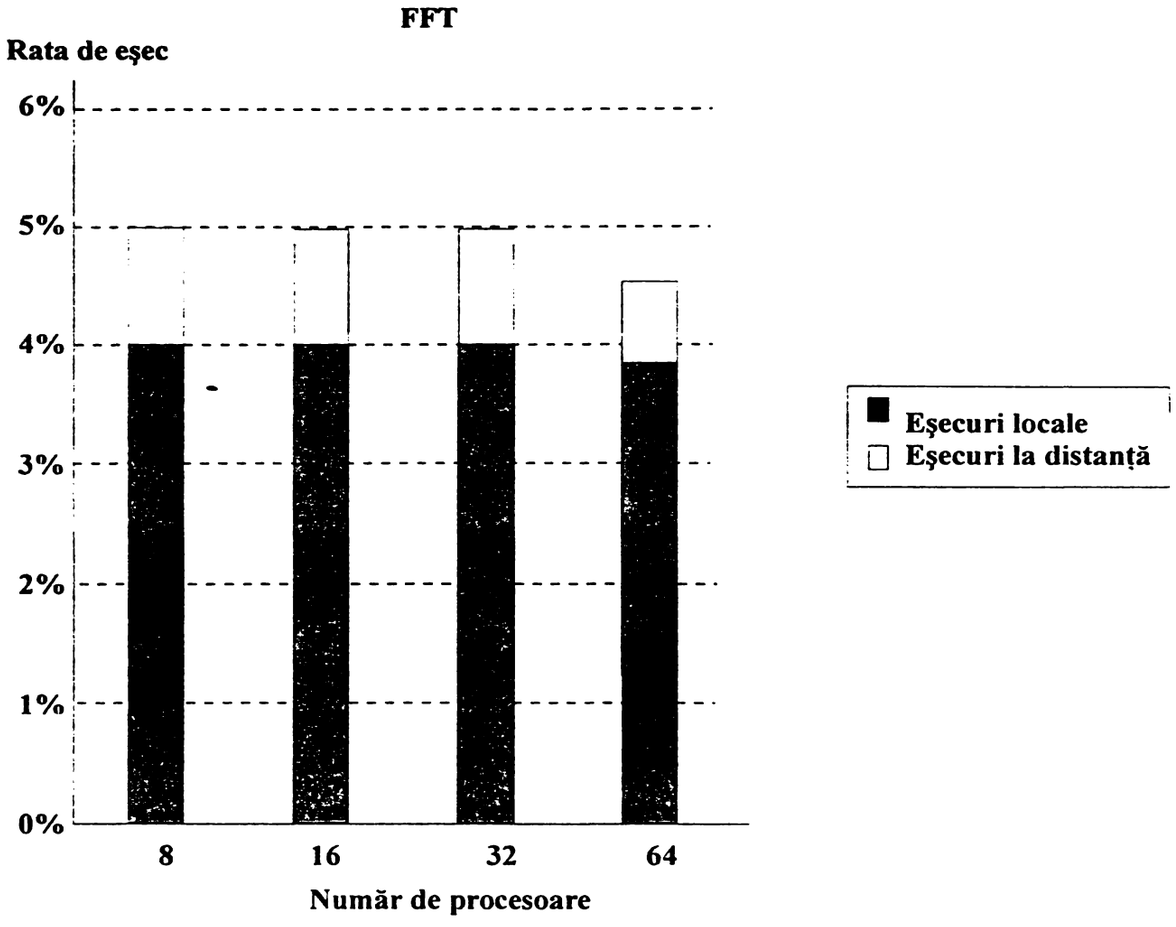


Fig. 3.32 Rata de eşec în funcţie de numărul de procesoare

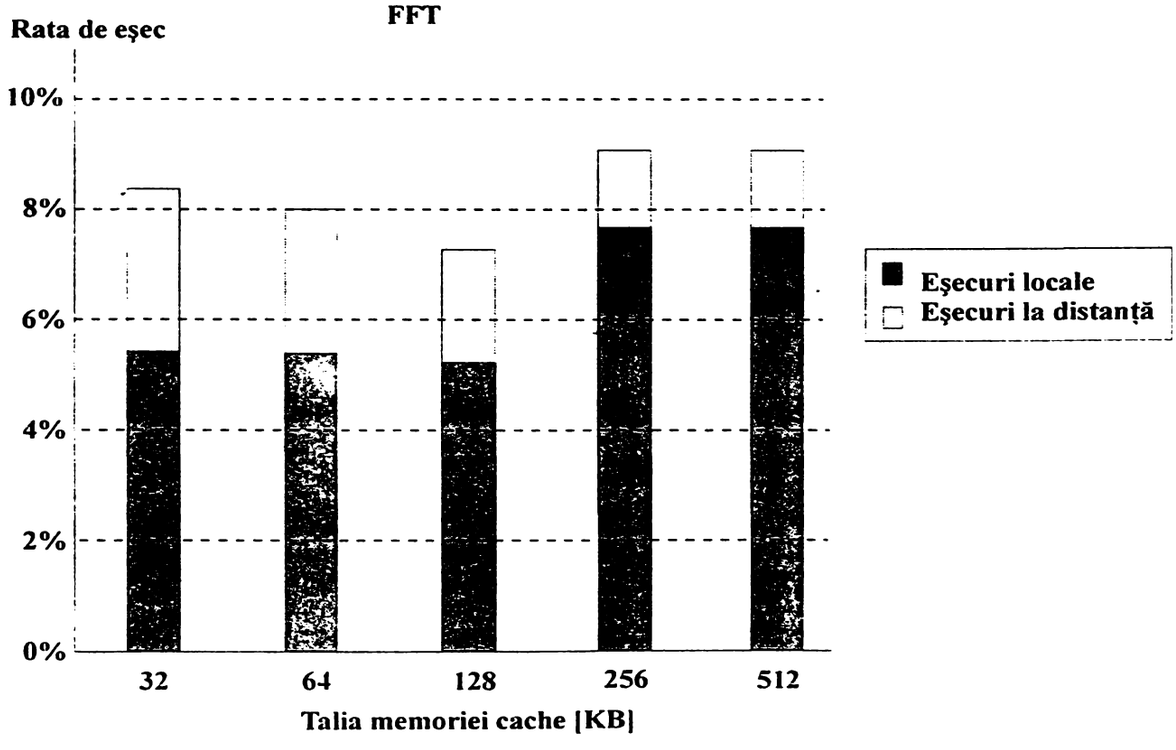


Fig. 3.33 Ratele de eşec scad pe măsură ce creşte talia cache

În fig.3.34 se prezintă efectul schimbării taliei blocului din memoria cache. Deoarece aplicația FFT are o bună localitate spațială, creşterea taliei blocului reduce rata de eşec chiar şi pentru blocuri mari şi prin urmare nu este rentabilă utilizarea acestora.

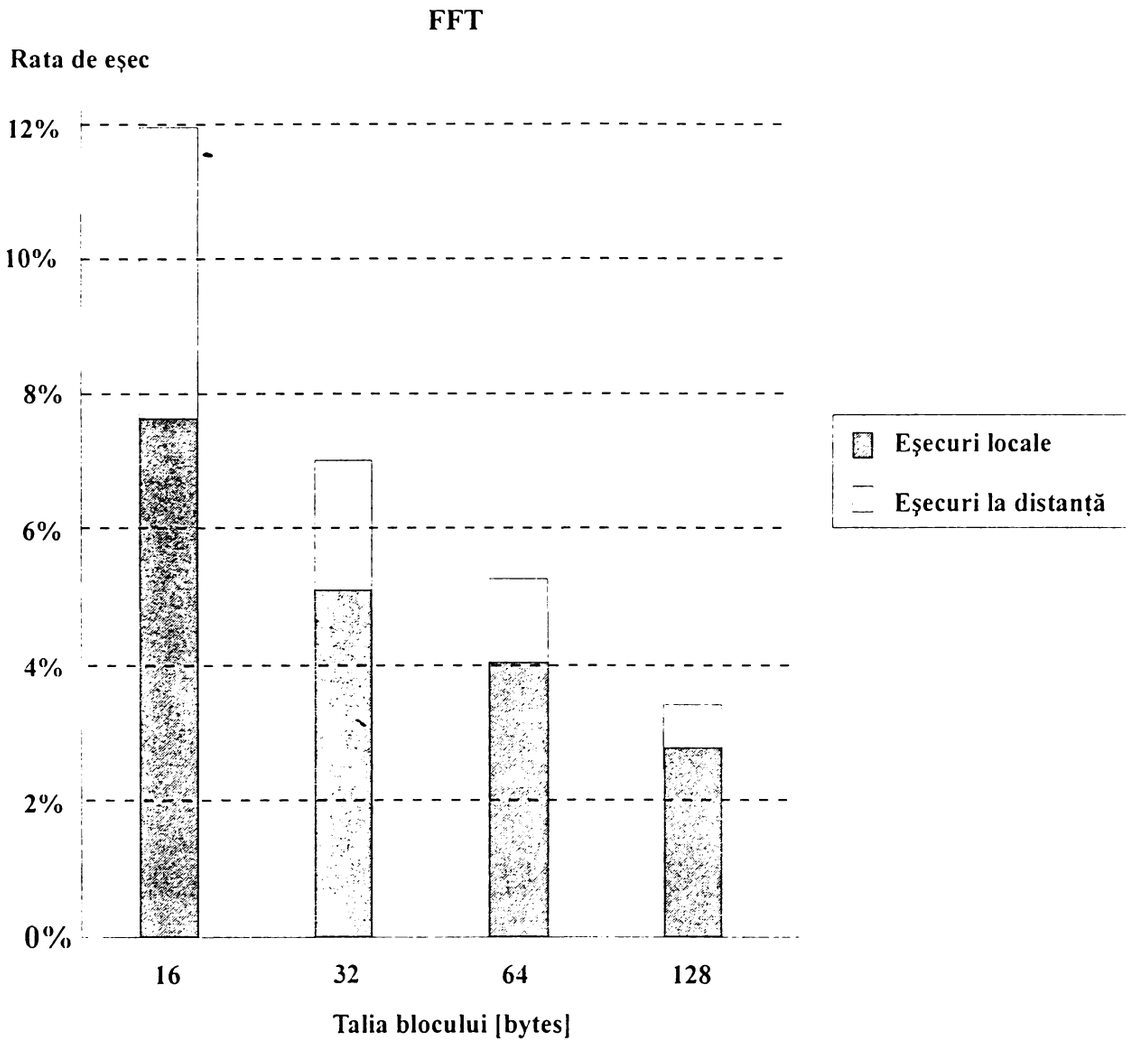


Fig. 3.34 Rata de eşec în funcție de talia blocului într-un sistem cu 64 de procesoare şi 128 KB memorie cache

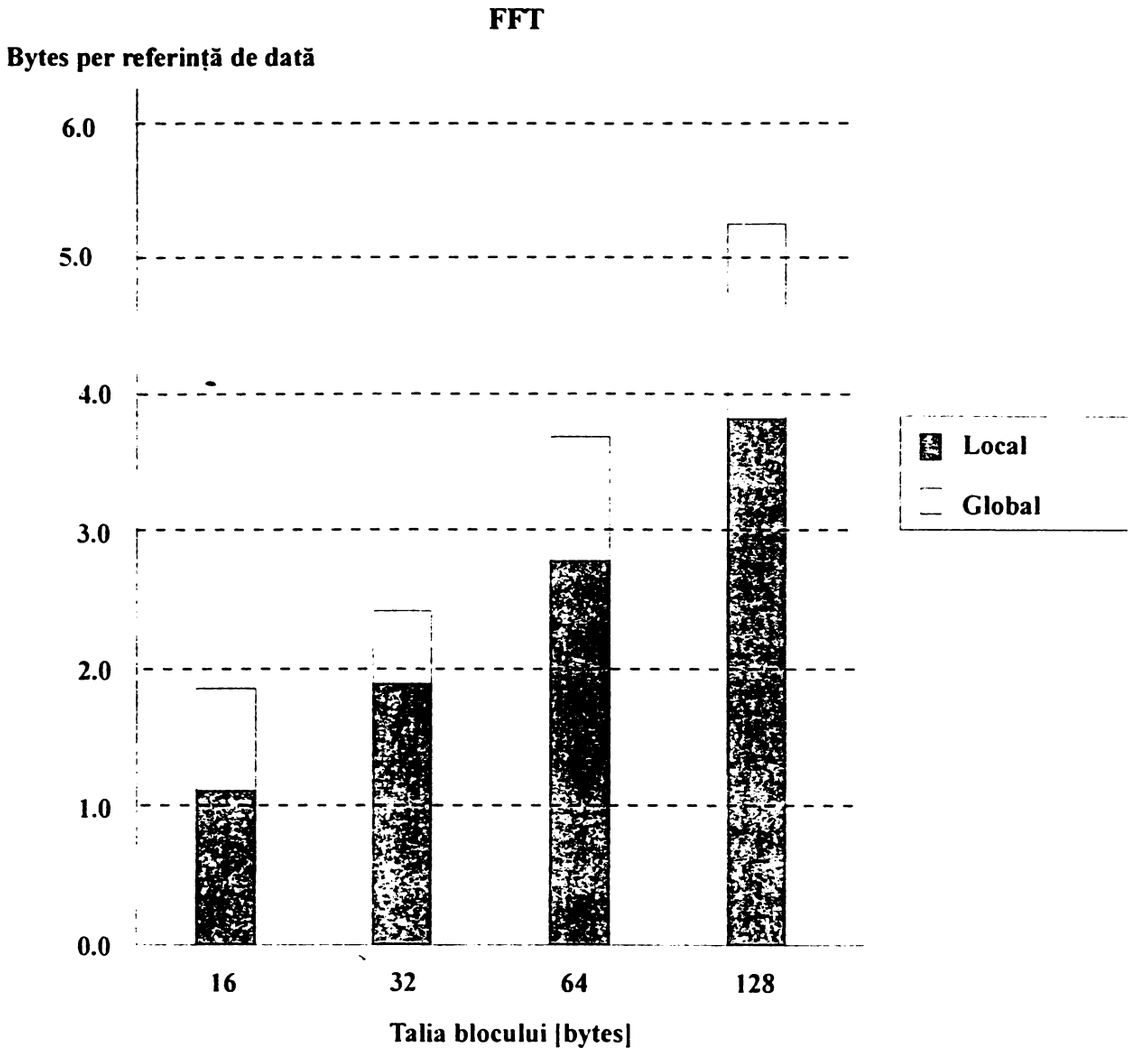


Fig. 3.35 Numărul de bytes per referință de dată crește cu talia blocului

3.2.3 Modele de consistență a memoriei

Coerența memoriei cache asigură procesoarelor multiple o viziune consistentă asupra memoriei.

Se pune problema gradului de consistență ce trebuie asigurat. De exemplu : când un procesor trebuie să "vadă" ce a fost actualizat de alt procesor, în ce ordine trebuie un procesor să observe scrierile de date ale altui procesor ?

Un exemplu simplu poate fi edificator asupra complexității procedurii de stabilire a gradului de consistență.

Fie două segmente de cod aparținând proceselor $P1$ şi $P2$.

```
P1 :    A = 0 ;                 P2 :    B = 0 ;

        A = 1 ;                         B = 1 ;
L1 :    if (B = = 0) . . .      L2 :    if (A = = 0) . . .
```

Se presupune că procesele rulează pe procesoare diferite şi că locaţiile A şi B sunt cachate iniţial de ambele procesoare cu valoarea iniţială 0.

Dacă scrierile au efect imediat şi sunt văzute imediat de alte procesoare, va fi imposibil pentru *ambele* instrucţii **if** (etichetate L1 şi L2) să evalueze condiţiile ca adevărate, deoarece ajungând la instrucţia **if** înseamnă că fie A fie B trebuie să fi avut asigurată valoarea 1.

Dar presupunând că invalidarea la scriere este întârziată şi că procesorului îi este permis să continue, atunci este posibil ca atât P1 cât şi P2 să nu fi văzut invalidările pentru B şi A înainte de-a încerca să citească valorile. Se ridică problema dacă un astfel de comportament este permis şi dacă da, în ce condiţii.

Cel mai direct model pentru consistenţa memoriei este denumit *consistenţă secvenţială*.

Consistenţa secvenţială pretinde ca rezultatul oricărei execuţii să fie acelaşi ca şi în cazul când accesele executate de fiecare procesor au fost ţinute în ordine şi accesele printre procesoare diferite au fost întreţesute.

Aceasta elimină posibilitatea unor execuţii ambigue ca şi în exemplu anterior, deoarece asingnările trebuie să fie completate înaintea efectuării instrucţiilor **if**.

În fig. 3.36 se prezintă procedura de interzicere de către consistenţa secvenţială a unei execuţii când ambele instrucţii **if** întâlnesc condiţia "true".

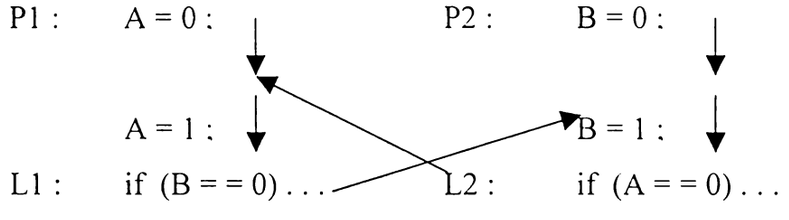


Fig. 3. 36 Un exemplu de consistenţă secvenţială

Cea mai simpla metodă de implementare a consistenței secvențiale constă în solicitarea adresată unui procesor să-şi întârzie efectuarea oricărui acces la memorie până când toate invalidările cauzate de acest acces sunt efectuate.

De reținut că consistența memoriei implică operații asupra diferitelor variabile. Cele două accese ce trebuie să fie ordonate se referă la locații diferite de memorie.

În exemplul anterior trebuie întârziată citirea lui A sau B (A = = 0 sau B = = 0) până când scrierea anterioară a fost efectuată (B = 1 sau A = 1). În consistența secvențială nu se poate plasa, de-o manieră simplistă, operația de scriere într-un tampon de scriere şi să se continue cu citirea.

Consistența secvențială oferă o paradigmă simplă în programare şi reduce performanța potențială îndeosebi la SMP cu un număr mare de procesoare sau cu întârzieri datorate interconectărilor la distanță.

Din punct de vedere al programatorului, acest model de consistență este cel mai agreat datorită simplității sale.

Programele trebuie să fie însă *sincronizate*, adică toate accesele la data partajată sunt ordonate de către operații de sincronizare. O referință la o dată este ordonată de către o operație de sincronizare dacă o scriere a unei variabile de către un procesor şi un acces (scriere sau citire) a acelei variabile de către alt procesor sunt separate de către o pereche de operații de sincronizare, una executată *după* scriere de către procesorul ce scrie şi alta executată *înaintea* accesului efectuat de cel de al doilea procesor.

Cursele de date apar când variabilele pot fi actualizate fără ordonarea prin sincronizare.

În literatură [HP 96] se prezintă mai multe tehnici de sincronizare.

În afară de operații de sincronizare, este necesară definirea ordonării operațiilor de memorie. Există două tipuri de restricții în ordonare : *bariere la scriere* (*write fences*) şi *bariere la citire* (*read fences*).

Barierele sunt puncte fixe în cadrul unui calcul ce asigură că nici o citire sau scriere nu pot trece peste ele.

De exemplu, o barieră la scriere executată de către un procesor P asigură că :

- toate scrierile efectuate de către P, ce apar înainte ca P să fi executat operația de barieră la scriere, sunt efectuate şi
- nu se inițiază nici o scriere ce apare după bariera din P, înainte de barieră.

În cadrul consistenței secvențiale, toate citirile sunt citiri cu barieră şi toate scrierile sunt scrieri cu barieră. Această proprietate limitează abilitatea structurii hardware de-a optimiza accesele, deoarece ordinea trebuie strict menținută.

Din punct de vedere al performanței, procesorul ar trebui să execute citirile cât se poate de devreme şi să efectueze scrierile cât de târziu posibil. Barierele acționează ca şi frontiere, forțând procesorul să ordoneze citirile şi scrierile în raport cu bariera.

O *barieră de memorie* (*memory fence*) reprezintă o operație ce acționează atât ca şi o barieră de citire cât şi ca una de scriere.

Barierele de memorie accentuează ordonarea printre accesele diferitelor procese. În cadrul unui singur proces este necesar ca ordinea din program să fie întotdeauna păstrată, astfel că citirile şi scrierile la aceeaşi locație nu pot fi interschimbate.

A. Modele relaxate pentru consistența memoriei

Aceste modele permit implementări cu performanță mai ridicată şi în acelaşi timp mai păstrează un model simplu de programare pentru programe sincronizate.

Modelele relaxate variază în ceea ce priveşte gradul de constrângere impus la implementare [CBZ95], [GLL+90], [GW88], [HP96].

Modelele pot fi definite în funcție de tipul de ordonări efectuate de un singur procesor, asupra citirilor şi scrierilor.

Există patru astfel de ordonări :

1. $R \rightarrow R$ o citire urmată de o citire;
2. $R \rightarrow W$ o citire urmată de o scriere, care este întotdeauna păstrată dacă operațiile sunt la aceeaşi adresă, aceasta fiind o antidependență;
3. $W \rightarrow W$ o scriere urmată de o scriere, care este întotdeauna prezervată dacă ele sunt la aceeaşi adresă, aceasta fiind o dependență de ieşire;
4. $W \rightarrow R$ o scriere urmată de o citire, care este întotdeauna păstrată dacă ele sunt la aceeaşi adresă, deoarece aceasta reprezintă o adevărată dependență.

Dacă există o dependență între citire şi scriere, atunci semanticile programului uniprocesor pretind ca operațiile să fie ordonate. Dacă nu există dependență, modelul de consistență, a memoriei determină ce ordonări trebuie păstrate. Un model de consistență secvențială pretinde ca toate cele patru ordonări să fie păstrate şi aceasta

este echivalent cu asumarea unui singur modul centralizat de memorie ce serializează toate operațiile procesorului sau cu presupunerea că toate citirile şi scrierile sunt bariere de memorie.

O ordonare este *relaxată*, atunci când se permite ca o operație ce trebuie executată mai târziu de către un procesor să fie efectuată prima.

În această situație, de exemplu, relaxarea ordonării W → R înseamnă că citirea care este în urma scrierii să se termine înainte ca scrierea să fie efectuată.

În realitate un model de consistență nu restricționează ordonarea evenimentelor. Ne interesează gradul de observabilitate a unor posibile ordonări. cu titlu de exemplu, citățm cazul consistenței secvențiale în care sistemul trebuie să păstreze cele patru ordonări, descrise anterior. În practică este permisă reordonarea fără ca ea să fie observabilă.

Modelul de consistență trebuie de asemenea să definească ordonările impuse între accesele la variabilele de sincronizare, ce acționează ca şi bariere, şi toate celelalte accese.

Când un sistem implementează consistența secvențială, toate citirile şi scrierile - inclusiv accesele de sincronizare - sunt bariere şi prin urmare ținute în ordine.

În cazul modelelor mai slabe este necesară specificarea restricțiilor de ordonare impuse de accesele de sincronizare la fel ca şi restricțiile de ordonare implicând variabile ordinare. Cea mai simplă restricție de ordonare este aceea că fiecare acces de sincronizare este o barieră de memorie.

Fie S un acces la o variabilă de sincronizare.

Utilizând notația de ordonare putem scrie :

S → W, S → R, W → S, şi R → S.

De remarcat că un acces de sincronizare este fie R fie W şi ordonarea sa este afectată de alte accese de sincronizare, ceea ce implică existența unei ordonări S → S.

În literatură [HP 96] se citează mai multe exemple de modele relaxate, caracterizate prin eliminarea unui ordin.

De exemplu dacă o operație de sincronizare este executată înaintea unei citiri, atunci ordonările W → S şi S → R asigură că scrierea se termină înaintea citirii.

Pentru acest model se utilizează nume ca şi *consistență a procesorului* (*processor consistency*) şi *ordonare de înmagazinare totală (total store ordering - TSO).*

Acest model este echivalent cu a face scrierile prin bariere de scriere. Prezentarea acestor modele este rezumată în TABEL VI iar un exemplu este prezentat în fig. 3.37.

TABEL VI. Ordonările impuse de diferite modele de consistență pentru accese ordinare și accese de sincronizare

Model	**Utilizat în**	**Ordonări ordinare**	**Ordonări de sincronizare**
Consistență secvențială	Cele mai multe sisteme	$R \rightarrow R$, $R \rightarrow W$, $W \rightarrow R$ $W \rightarrow W$	$S \rightarrow W$, $S \rightarrow R$, $R \rightarrow S$, $W \rightarrow S$, $S \rightarrow S$
TSO sau consistență de procesor	IBM 370, DEC VAX, SPARC	$R \rightarrow R$, $R \rightarrow W$, $W \rightarrow W$	$S \rightarrow W$, $S \rightarrow R$, $R \rightarrow S$, $W \rightarrow S$, $S \rightarrow S$
PSO	SPARC	$R \rightarrow R$, $R \rightarrow W$	$S \rightarrow W$, $S \rightarrow R$, $R \rightarrow S$, $W \rightarrow S$, $S \rightarrow S$
Ordonare slabă	Power PC		$S \rightarrow W$, $S \rightarrow R$, $R \rightarrow S$, $W \rightarrow S$, $S \rightarrow S$
Consistență eliberată	Alpha, MIPS		$S_A \rightarrow W$, $S_A \rightarrow R$, $R \rightarrow S_R$, $W \rightarrow S_R$, $S_A \rightarrow S_A$, $S \rightarrow S_R$ $S_R \rightarrow S_A$, $S_R \rightarrow S_R$

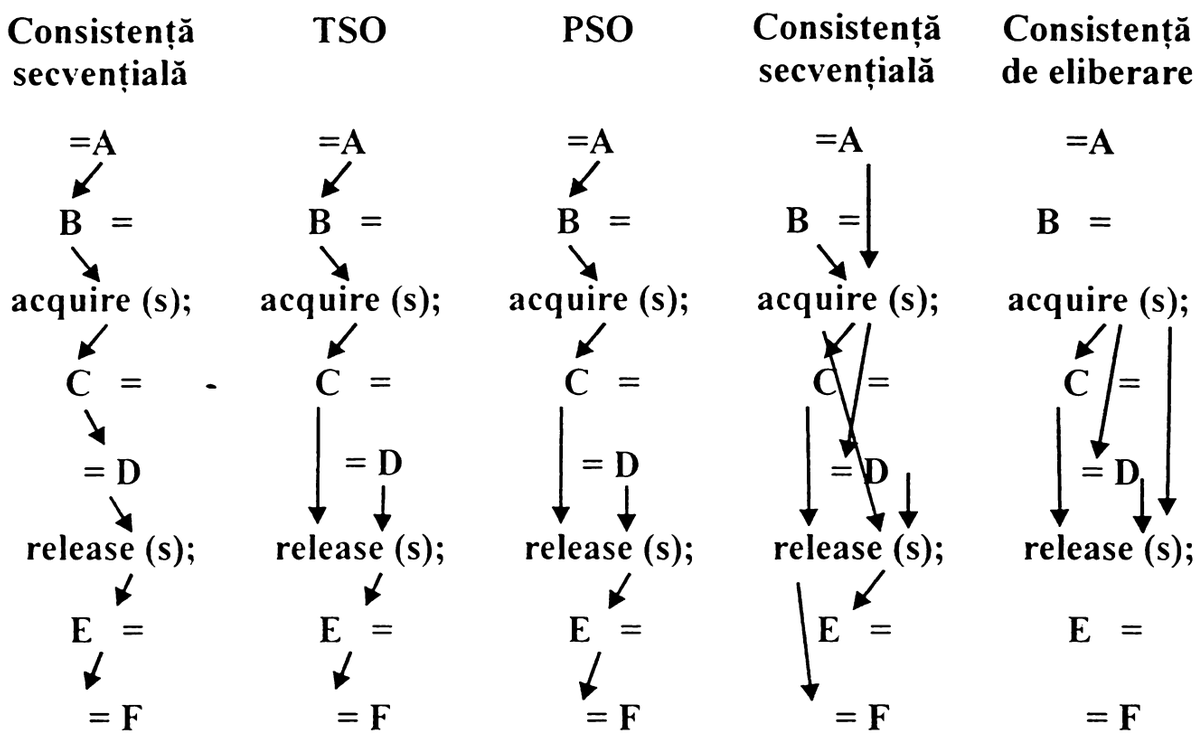


Fig.3.37 Reducerea numărului de ordine impuse pe măsură
ce modelele devin mai relaxate. Ordinele minime sunt prezentate cu săgeți

B. Performanța modelelor relaxate

Potențialul de performanță a modelelor relaxate depinde atât de capabilitățile sistemului cât şi de aplicație. La examinarea performanței modelului de consistență a memoriei se recurge în prealabil la definirea mediului hardware. Configurațiile hardware luate în considerare trebuie să aibă câteva caracteristici comune. Acestea sunt:

- Erorile cache consumă 50 de cicluri de ceas:
- CPU include un tampon de scriere cu o adâncime de 16:
- Memoriile cache au 64 KB şi au linii de 16 byte.

Se pot considera [HP96] patru modele hardware:

1. *SSBR (Staticaly Scheduled with Blocking Reads)* - Procesorul este planificat static şi citirile ce eşuează în cache sunt imediat blocate.
2. *SS (Staticaly Scheduled)*- Procesorul este planificat static dar citirile nu cauzează blocarea procesorului până când nu se utilizează rezultatul.
3. *DS16 (Dynamically Scheduled with a 16-entry reorder buffer)*- Procesorul este planificat dinamic şi are un tampon de reordonare ce permite înmagazinarea a până a 16 instrucţii de orice tip, incluzând 16 instrucţii de acces la memorie.
4. *DS64 (Dynamically Scheduled with a 64-entry reorder buffer)*- Procesorul este planificat dinamic şi are un tampon de reordonare ce permite manevrarea a până la 64 de instrucţii de oricare tip. Tamponul de reordonare este suficient de mare pentru a masca latenţa totală a eşecurilor cache.

În fig. 3.38 se prezintă performanţele pentru două dintre programele paralele de evaluare, LU şi Ocean, menţionate în paragrafele anterioare. Au fost luate în considerare cele patru metode hardware şi două modele de consistenţă - TSO (Total Store Order) şi RC (Release Consistency).

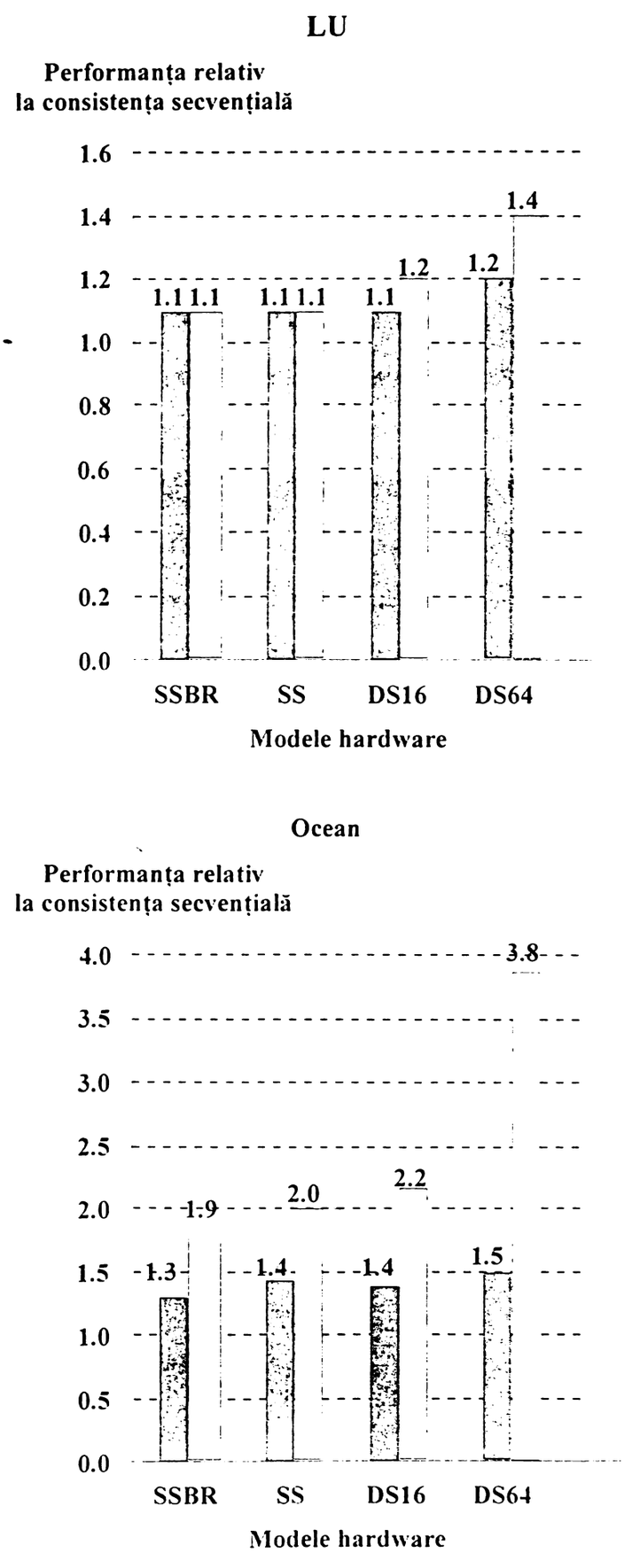


Fig. 3.38 Performanța modelelor de consistență relaxate prelevată la mai multe mecanisme hardware

Se remarcă că modelele relaxate oferă o performanță superioară la aplicații din clasa Ocean comparativ cu LU, deoarece Ocean are o mai mare rată de eşec şi are o fracțiune semnificativă a eşecurilor la scriere.

În proiectare se preferă mărirea taliei memoriei cache înainte de-a include citiri fără blocare. Se reduce astfel semnificativ rata de eşec iar modelele relaxate devin atractive.

3.3 CONCLUZII

Utilizarea MCH private în cadrul SMP-MP poate reduce semnificativ timpul mediu de acces la memorie, dar aceste memorii introduc problematica complexă a menținerii coerenței acestora. Caracteristicile arhitecturale ce afectează mecanismul de menținere a coerenței memoriilor cache sunt:

- *strategia de detectare a coerenței* ce determină când şi cum sunt definite în mod clar referirile la memorie pentru a se detecta existența unei posibile incoerențe între memoriile cache pentru date şi memoria centrală. Mecanismele de detectare a coerenței în mod *dinamic* examinează adresele de memorie generate în timpul execuției. Eliminarea ambiguităților referitoare la memorie provoacă o rată redusă a eşecurilor în acces, dar mecanismele dinamice tind să producă un trafic de rețea ridicat datorită mesajelor necesare pentru a menține coerența. Schemele *statice* de detectare a coerenței, în contrast cu cele dinamice, examinează referirile la memorie în timpul compilării. *Autoinvalidarea* utilizată în mecanismele statice reduce traficul de rețea în comparație cu acela produs de strategiile dinamice de detectare a coerenței.
- *strategia de întărire a coerenței* ce trebuie să prevină referirea de către un procesor a unei date neactualizate. Acest deziderat se realizează prin forțarea procesoarelor ca să invalideze blocurile în memoriile cache aparținătoare. Dacă blocul este referit din nou, se va genera un eşec care va obliga procesorul să caute valoarea curentă a blocului, fie în memoria principală, fie în memoria aparținătoare altui procesor. Prin intermediul unei strategii de *actualizare*, noua valoare a unui bloc creat printr-o operațiune de scriere este automat distribuită către toate procesoarele împreună cu o copie cache-ată a blocului. Atunci când aceste procesoare fac din nou o referire a blocului, ele nu

generează altă cerere de deservire a eşecului în acces. Prin urmare, strategia de actualizare tinde să producă rate reduse de eşec în comparaţie cu strategia de invalidare. Această performanţă se realizează însă prin creşterea semnificativă a traficului în reţea.

- *dimensiunea informaţiei referitoare la partajarea blocului* care este întreţinută de mecanismul de asigurare a coerenţei. Această dimensiune are un impact direct aşupra costului implementării mecanismului în termeni de număr de biţi necesari pentru a înmagazina informaţia de partajare şi la fel un impact direct asupra performanţei sistemului de memorie. Pentru a reduce necesităţile de capacitate de memorie, mecanismul de menţinere a coerenţei poate să înmagazineze o cantitate relativ redusă de informaţie referitoare la procesoarele care au o copie a blocului cache-at. Mecanismul trebuie în continuare să transmită mesaje de invalidare atunci când numărul de procesoare ce partajează un bloc depăşeşte resursele disponibile. Această abordare reduce necesitatea de memorie suplimentară dar creşte traficul în reţea în comparaţie cu schemele bazate pe director care înmagazinează o informaţie completă despre modul de partajare. Recent, schemele propuse care conţin *tabel director etichetat* pot realiza o foarte mică creştere a capacităţii de memorare prin înmagazinarea informaţiilor de partajare referitoare doar la acele blocuri care sunt cacheate la un moment dat. Aceşti directori pot să menţină un nivel redus al traficului în reţea deoarece sunt capabili să înmagazineze o informaţia suficientă referitoare la partajare pentru fiecare bloc cacheat.
- *dimensiunea blocului cache* reprezintă un factor important ce afectează performanţa sistemului de memorie. Această dimensiune reprezintă numărul de cuvinte înmagazinate în memoria cache, ca şi o singură unitate. Utilizarea blocurilor cache mai mari decât un singur cuvânt pot să permită procesoarelor să exploateze localitatea spaţială tipică unui comportament ce face referire la memorie. Referirile la memorie într-un SMP tind să se disperseze printre procesoarele care reduc localitatea spaţială disponibilă în comparaţie cu un sistem uniprocesor. În plus, blocurile mai mari decât un singur cuvânt introduc problema falsei partajări care tinde să provoace preferinţa pentru dimensiunile mici de blocuri cache. În câteva programe de aplicaţii este posibil să se reducă

rata de eşecuri în acces prin utilizarea blocurilor de cuvinte multiple dar studiile de simulare sugerează că blocurile de un singur cuvânt minimizează traficul în reţea prin reducerea atât a traficului de deservire a eşecurilor în acces cât şi a traficului de invalidare.

În final, este important să se sublinieze că în arhitecturile moderne de SMP se pot incorpora diferite mecanisme de menţinere a coerenţei în memoriile cache. De exemplu prototipul DASH are încorporat în mecanismul de menţinere a coerenţei atât un mecanism de menţinere a coerenţei bazat pe monitorizarea magistralei cât şi un mecanism de menţinere a coerenţei bazat pe director. Se poate oferi programatorului posibilitatea de a alege între strategii de întărire a coerenţei bazate atât pe actualizare cât şi pe invalidare. Suplimentar este posibil să se utilizeze informaţia obţinută în timpul de compilare pentru a creşte performanţa mecanismului de menţinere a coerenţei: de exemplu de a reduce talia directorului prin reducerea numărului de pointeri de coerenţă ce sunt necesari ca să fie alocaţi şi prin reducerea timpului necesar pentru a fi activaţi. Deoarece fiecare din factorii ce afectează mecanismul de menţinere a coerenţei produce facilităţi în termeni de rată de eşec în acces şi de trafic în circuit, este clar că abordările hibride vor oferi cea mai bună oportunitate pentru creşterea performanţei şi reducerea costului de implementare a mecanismului de menţinere a coerenţei în SMP-MP de capacitate mare.

4. ANALIZA ARHITECTURILOR SISTEMELOR DE CALCUL MIMD TOLERANTE LA DEFECT

4.1 TERMINOLOGIE

In acest paragraf vom introduce câteva noțiuni de bază necesare în analiza sistemelor tolerante la defecte.

Un *sistem tolerant la defect* (*fault-tolerant system*) este atributul ce permite unui sistem să devină tolerant la defect.

Calcul tolerant la defect (*fault-tolerant computing*) este procesul prin care se efectuează calcule într-un sistem de calcul în manieră tolerantă la defect.

Un *defect* (*fault*) este o imperfecțiune fizică ce apare în cadrul unor componente hard/soft.

O *eroare* (*error*) reprezintă manifestarea defectului.

Defectarea (*failure*) repezintă procesul prin care nu se efectuează acțiuni aşteptate. Reprezintă o performanță a unei funcții, în cantitate şi în calitate, sub normalul admis.

In literatură [Prad 96] se citează *modelul triunivers* (*three universe model*), o adaptare a modelului *în patru universuri* (*four universe model*) dezvoltat de Avizienis (1982), în care noțiunile prezentate anterior pot fi mai bine înțelese.

Un defect apare într-un *univers fizic* (semiconductori, elemente mecanice, surse da alimentare, etc) şi se manifestă printr-o eroare ce apare în *universul informațional* (cuvinte binare, imagini, voce digitală, etc).

Comportamentul eronat apare în *universul extern* sistemului. Se pot defini doi parametri importanți:

- *latența defectului* (*fault latency*) – timpul scurs între apariția unui defect şi apariția erorii cauzate de defect
- *latența erorii* (*error latency*) – timpul scurs între apariția unei erori şi apariția comportamentului eronat al sistemului.

Bazat pe modelul tri-univers, timpul total consumat între apariția defecțiunii şi apariția comportamentului eronat al sistemului va fi suma latenței defecțiunii şi a latenței erorii.

Pentru descrierea defectelor în mod adecvat, se utilizează patru atribute majore: *natura ,durata, extinderea* şi *valoarea Nelson şi Carroll* [Prad 96].

Natura defectului specifică tipul defectului: hardware, software, în circuite analogice, etc.

Durata specifică durata manifestării defectului. Există *defect permanent, tranzitoriu* şi *intermitent*.

Extinderea specifică dacă un defect este localizat la un modul dat sau afectează global modulele hard, soft sau le afectează pe ambele.

Valoarea unui defect poate fi *determinată* sau *nedeterminată*. Un defect determinat este unul a cărei stare rămâne nemodificată în timp până la apariția unei acțiuni externe. Un defect nedeterminat este unul a cărui stare la un moment T poate fi diferită de starea sa la $T + \Delta T$.

Există trei tehnici primare utilizate pentru a menține performanța normală a unui sistem într-un mediu în care defectele pot apare:

- *evitarea defectului* (*fault avoidance*);
- *mascarea defectului* (*fault masking*);
- *toleranța la defect* (*fault tolerance*).

Toleranța la defect poate fi obținută prin mai multe tehnici:

Reconfigurarea – procesul de eliminare a entității cu funcționare defectuoasă dintr-un sistem şi restaurarea sistemului la o stare operațională. Dacă se utilizează această tehnică atunci proiectantul trebuie să aibă în vedere *detectarea defectului, localizarea defectului, izolarea defectului* şi *recuperarea din defect*.

Detectarea defectului (*fault detection*) este procesul de recunoaştere a apariției unui defect.

Localizarea defectului (*fault location*) este procesul de determinare a locului unde a apărut defectul.

Izolarea defectului (*fault containment*) este procesul de izolare a unui defect şi de prevenire a extinderii lui prin sistem.

Recuperarea din defect (*fault recovery*) este procesul prin care se menține starea operațională a sistemului sau se recâştigă această stare prin reconfigurare.

Definiții echivalente pot fi date şi în universul informațional operând asupra noțiunii de eroare în loc de defect.

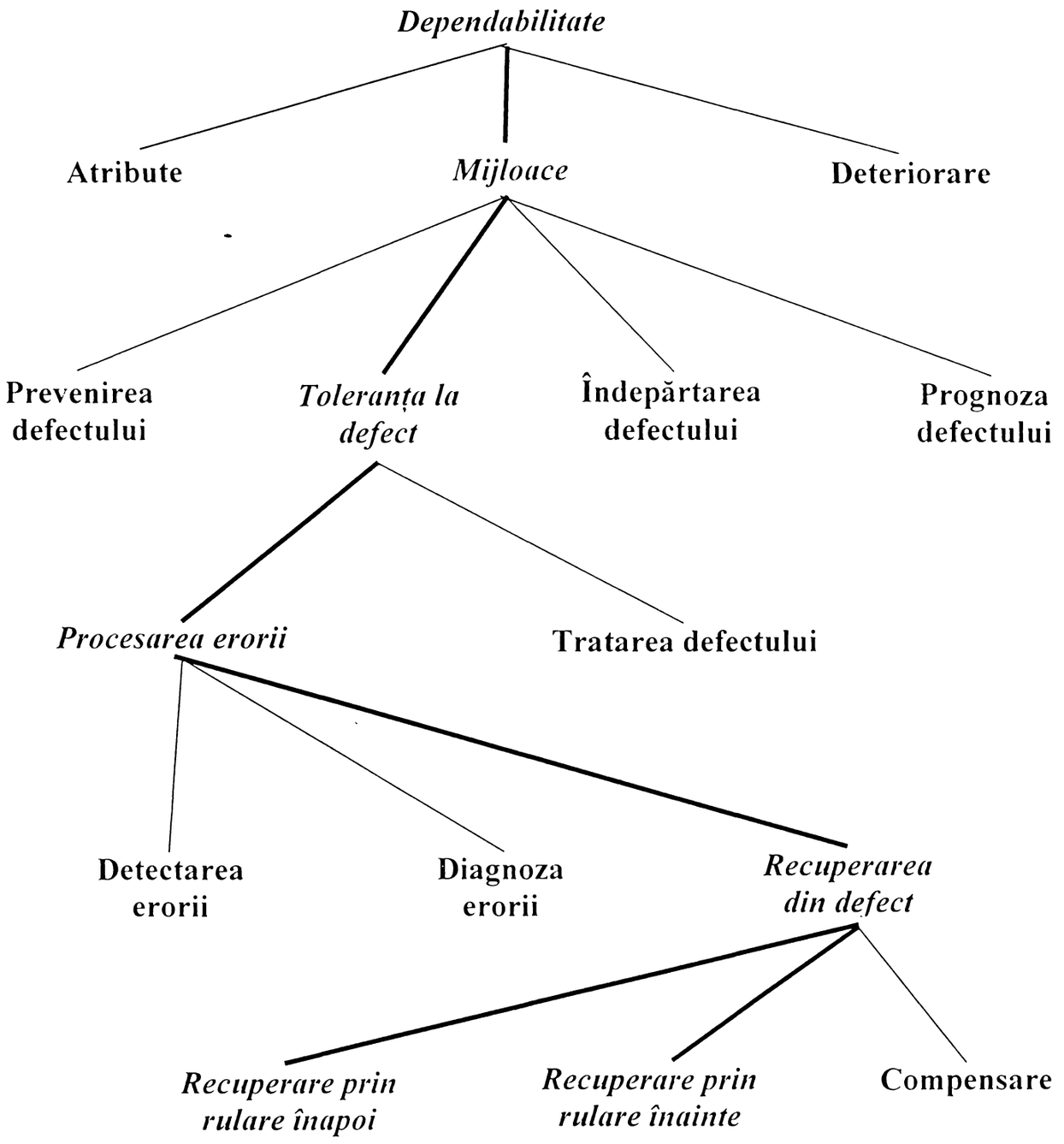


Fig. 4.1 Arborele dependabilității

4.1.1 Obiectivele toleranței la defect la sistemele multiprocesor

Toleranța la defect este un atribut structural în cadrul SMP pentru a atinge anumite cerințe de proiectare.

Cele mai proeminente cerințe adiționale, pe lângă cele vizând funcționalitatea şi performanța, sunt :

dependabilitatea, fiabilitatea, disponibilitatea, siguranța, performabilitatea, mentenabilitatea şi testabilitatea [Prad 96].

Toleranța la defect este unul dintre atributele sistemului capabil să satisfacă astfel de cerințe.

Dependabilitatea. *(Dependability)* Acest termen este utilizat pentru a incorpora conceptele de *fiabilitate, disponibilitate, siguranță, mantenabilitate, performabilitate* şi *testabilitate*.

Dependabilitatea se poate defini simplu [Prad 96] ca fiind calitatea serviciilor furnizate de un anume sistem de calcul. Componentele dependabilității sunt exemple de măsuri ale cantității de dependabilitate ale unui sistem. Arborele dependabilității este prezentat în fig.4.1.

Fiabilitatea *(Reliability)* unui sistem este o funcție de timp, $R(t)$, definită ca şi probabilitatea condițională ca sistemul să funcționeze corect în intervalul de timp, $[t_0, t]$, dat fiind faptul că sistemul a funcționat corect la t_0.

Cu alte cuvinte, *fiabilitatea* reprezintă probabilitatea ca sistemul să opereze corect într-un interval complet de timp.

Fiabilitatea este o probabilitate condițională ce depinde de starea operațională a sistemului la începutul intervalului de timp ales.

Nesiguranța în funcționare, opusul fiabilității, este o funcție de timp, $F(t)$, definită ca şi probabilitatea condițională ca sistemul să evolueze incorect în intervalul, $[t_0, t]$, cunoscându-se faptul că la t_0 el funcționează corect.

Disponibilitatea. *(Availability). Disponibilitatea* este o funcție de timp, $A(t)$, definită ca şi probabilitatea ca un sistem să fie operat corect şi să fie disponibil la orice moment t.

Disponibilitatea diferă de siguranța în funcționare în sensul că trebuie să se manifeste în orice moment şi nu în intervale temporale.

Siguranța. *(Safety). Siguranța* reprezintă probabilitatea, $S(t)$, ca un sistem să-şi efectueze funcțiile corect sau să-şi întrerupă funcționarea într-o manieră în care nu afectează operarea întregului sistem sau în care să compromită securitatea

personalului asociat cu sistemul. Siguranța este un parametru prin care se evidențează modalitatea prin care - dacă nu funcționează corect - se ajunge sigur şi controlat la starea de defecțiune.

Fiabilitatea şi siguranța diferă deoarece fiabilitatea reprezintă probabilitatea ca un sistem să-şi efectueze funcțiile corect, în timp ce siguranța este probabilitatea ca un sistem fie să-şi efectueze funcțiile corect fie să-şi întrerupă funcționarea într-o manieră care să nu dăuneze.

Performabilitatea. *(Performability). Performabilitatea* unui sistem este o funcție de timp, *P(L, t)*, definită ca şi probabilitatea ca performanța sistemului în orice moment *t* să aibă acelaşi nivel *L*.

Performabilitatea diferă de fiabilitate în sensul că în timp ce fiabilitatea este o măsură globală ce se referă la faptul că toate funcțiile sunt efectuate corect, performabilitatea indică că un subset de funcții este efectuat corect.

Degradarea grațioasă este strâns legată de performabilitate şi reprezintă abilitatea sistemului de a-şi coborî nivelul de performanță pentru a compensa defecțiuni hard sau software.

Mentenabilitatea. *(Maintanability). Mentenabilitatea* reprezintă o măsură prin care se indică că un sistem poate fi reparat odată ce el a căzut.

M(t) reprezintă probabilitatea ca un sistem defect să fie readus la o stare operațională într-o periodă de timp *t*.

Testabilitatea. *(Testability). Testabilitatea* reprezintă abilitatea de-a testa anumite atribute ale sistemului. Ea este strâns corelată cu mentenabilitatea datorită importanței minimizării timpului cerut pentru identificarea şi localizarea problemelor specifice.

4.1.2 Parametri de evaluare a dependabilității

Este necesară o abordare cantitativă a dependabilității prin introducerea unor parametri ca şi:

Rata de defectare şi funcția de fiabilitate *(Failure rate and the reliability function)*

Reprezintă numărul aşteptat de defectări ale unui sistem într-o perioadă dată de timp. Când are o valoare constantă se notează cu λ.

În literatură [Prad 96] se menționează definirea acestui parametru. Se consideră că *R(t)* – fiabilitatea unui sistem – reprezintă probabilitatea condițională ca elementul

component să opereze corect în intervalul $[t_0, t]$, presupunând că a operat corect la t_0. Presupunem că se rulează un test pe N componente identice plasând cele N componente în operare la t_0 şi înregistrând numărul de componente căzute şi a celor care sunt active la timpul t.

Fie $N_f(t)$ numărul de componente defecte la momentul t şi $N_o(t)$ numărul de componente ce au operat corect la momentul t. Se presupune că odată ce un element s-a defectat el rămâne în această stare un timp nedefinit.

Fiabilitatea componentelor la momentul t este dată de

$$R(t) = \frac{N_o(t)}{N} = \frac{N_o(t)}{N_o(t) + N_f(t)} \tag{4.1}$$

şi reprezintă probabilitatea ca un element component să fi supravieţuit în intervalul $[t_0, t]$.

Probabilitatea ca un element să nu fi supravieţuit în intervalul $[t_0, t]$ se numeşte *nefiabilitate* şi este dată de

$$Q(t) = \frac{N_f(t)}{N} = \frac{N_f(t)}{N_o(t) + N_f(t)}. \tag{4.2}$$

In orice moment t există relaţia $R(t) = 1.0 - Q(t)$.

Prin urmare:

$$\frac{dR(t)}{di} = \left(-\frac{1}{N}\right)\frac{dN_f(t)}{dt}$$

sau

$$\frac{dN_f(t)}{dt} = (-N)\frac{dR(t)}{dt}.$$

Funcţia obţinută

$$Z(t) = \frac{1}{N_o(t)}\frac{dN_f(t)}{dt} \tag{4.3}$$

se numeşte *funcţie hazard, rata hazardului* sau *funcţia rată a defectării.*

Exprimarea funcţiei $Z(t)$ în funcţie de $R(t)$, respectiv $Q(t)$

$$Z(t) = \frac{\frac{dR(t)}{dt}}{R(t)} \quad \text{şi} \quad Z(t) = \frac{\frac{dQ(t)}{dt}}{1 - Q(t)} \tag{4.4}$$

unde:

$\frac{dQ(t)}{dt}$ - *funcția de densitate a defectării.*

În literatură [Prad 96] se definesc $R(t) = e^{-\lambda t}$ ca fiind *legea defectării exponențiale* şi *funcția rată a defectării* asociată unei distribuții Weibull

$$Z(t) = \alpha\lambda(\lambda t)^{\alpha-1} \tag{4.5}$$

unde α şi λ reprezintă constante ce controlează variația funcției rată a defectării în timp.

Timpul mediu până la defectare *(Mean Time To Failure)*

Reprezintă un parametru util pentru a specifica calitatea unui sistem. *MTTF* este timpul aşteptat ca un sistem să opereze înainte ca prima defectare să apară.

Dacă avem *N* sisteme identice plasate în operare la *t=0* şi dacă măsurăm timpul în care fiecare sistem operează înainte de a cădea, atunci tipul mediu este *MTTF*.

Dacă fiecare sistem *i* operează pentru un timp t_i, înainte de a întâlni prima defectare, *MTTF* este:

$$MTTF = \sum_{i=1}^{N} \frac{t_i}{N}. \tag{4.6}$$

Relația între *MTTF* şi *R(t)* este:

$$MTTF = \int_0^{\infty} R(t)dt \tag{4.7}$$

valabilă pentru $R(\infty) = 0$.

Timpul mediu de reparare *(Mean Time To Repair)*

MTTR reprezintă timpul mediu necesar pentru a repara un sistem. Dacă al *i-lea* defect din cele *N* necesită un timp t_i pentru a-l repara, atunci *MTTR* se estimează astfel:

$$MTTR = \frac{\sum_{i=1}^{N} T_i}{N}. \tag{4.8}$$

Dacă considerăm că μ - *rata de reparare* – este numărul de reparații pe o perioadă de timp atunci

$$MTTR = \frac{1}{\mu}. \tag{4.9}$$

Timpul mediu între defectări *(Mean Time Between Failure)*

Trebuie de la început să menționăm semnificațiile lui *MTTF* şi *MTBF*.

MTTF reprezintă timpul mediu scurs până la prima defectare a unui sistem în timp ce *MTBF* este timpul mediu consumat între defectările unui sistem.

MTBF este calculat prin medierea timpului dintre defectări, incluzând orice timp necesar pentru a repara sistemul şi a-l plasa apoi în starea operaţională. Cu alte cuvinte, fiecare din cele N sisteme este operat pentru un timp oarecare T şi numărul de defectări întâlnit de către al *i-lea* sistem se înregistrează ca n_i, media numărului de defectări este:

$$n_{med} = \sum_{i=1}^{N} \frac{n_i}{N}$$

iar în final

$$MTBF = \frac{T}{n_{med}}. \tag{4.10}$$

Cu alte cuvinte, *MTBF* este timpul de operare total T, împărţit cu numărul mediu de defectări produse în timpul T.

Experienţa arată că:

$$MTBF = MTTF + MTTR. \tag{4.11}$$

Acoperirea defectului *(Fault coverage)*

Există două definiţii ale acoperirii defectului: una intuitivă şi cealaltă matematic formalizată. Definiţia intuitivă este aceea că acoperirea reprezintă o măsură a abilităţii sistemului de a efectua detectarea defectului, localizarea lui, izolarea şi/sau acoperirea lui.

Primele patru tipuri de acoperire a defectului sunt: *acoperirea detectării defectului*, *acoperirea localizării defectului*, *acoperirea izolării defectului* şi *acoperirea recuperării din defect*.

Acoperirea detectării defectului este o măsură a abilităţii sistemului de a detecta defecte. De exemplu, o cerinţă a sistemului poate fi că o anumită fracţiune din toate defectele trebuie să fie detectată; acoperirea detectării defectului este o măsură a sistemului de a îndeplini o astfel de cerinţă. *Acoperirea izolării defectului* este o măsură de a izola defecte. *Acoperirea recuperării din defect* reprezintă o măsură a abilităţii sistemului de a reveni din defecte şi de a menţine o stare operaţională.

Acoperirea la defect este matematic definţă ca şi probabilitatea ca, existând un defect, sistemul să-şi revină. Problema fundamentală la acoperirea la defect este că se

calculează foarte laborios. Aici nu se va insista asupra tehnicilor de calcul al acestui parametru.

4.2 TOLERANȚA LA DEFECT PRIN UTILIZAREA REDUNDANȚEI

4.2.1 Toleranța la defect prin utilizarea *redundanței statice*

Există trei aplicații distincte ale *redundanței statice* în mediul multiprocesor: *redundanța pentru fiabilitate* și *disponibilitate*, *redundanța pentru siguranță* și *redundanța pentru tolerarea defectelor non-clasice*.

4.2.1a Redundanța pentru *fiabilitate* și *disponibilitate*

Prima abordare a toleranței la defect în SMP este la nivel de sistem, implicând utilizarea redundanței statice unde N copii ale unui modul efectuează calcule simultan pentru a fi verificate și validate. Există și alte abordări la alte nivele. Fiecare procesor poate fi replicat și rezultatul calculului fiecărui procesor validat sau întregul SMP poate fi replicat și rezultatul combinat validat. O a treia opțiune împarte cele P procesoare ale SMP în P/N grupuri de N procesoare, fiecare grup validându-și rezultatele înainte de comunicarea cu alte grupuri.

De asemenea, pentru a furniza comunicări robuste, toate tranzacțiile critice între grupuri pot fi replicate și ulterior validate. Replicarea și validarea în diverse grade a fost utilizată în mai multe proiecte din perioada anilor '60 [Prad96].

4.2.1b Redundanța pentru *siguranță*

Fiabilitatea se referă la probabilitatea ca sistemul să producă o ieșire corectă. Siguranța este definită ca fiind probabilitatea ieșirii sistemului de a fi corectă sau ca eroarea apărută să fie detectabilă. O siguranță ridicată este asigurată făcând neglijabilă probabilitatea apariției unei erori nedetectate la ieșire. Când se sesizează la ieșire o eroare ce nu poate fi detectată, se poate declanșa o procedură de recuperare. O schemă de toleranță la defect trebuie să fie aleasă atunci când îndeplinește cerințele de fiabilitate – siguranță. Trebuie menționat că atunci când un sistem este fiabil este de asemenea și sigur. Reciproca nu este întotdeauna valabilă. Un sistem poate fi sigur într-un grad înalt doar dacă are un timp mediu până la defectare mic. În general siguranța unui sistem este cel puțin egală cu fiabilitatea.

Strategiile de proiectare pot atinge atât o fiabilitate cât și o siguranță ridicate, utilizând sisteme formate din module identice multiple ce efectuează toate aceeași funcție.

Modelul generic este ilustrat în figura 4.2. Sistemul este format utilizând module identice multiple, toate executând aceeași funcție. Ieșirile modulelor sunt arbitrate iar

ieşirea arbitrului constituie ieşirea sistemului. Aceasta are două componente: *ieşirea de date* şi *fanionul de indicare a nesiguranţei* ce marchează eroare la datele de ieşire. Fanionul de indicare a nesiguranţei, atunci când este poziţionat, indică apariţia unei erori la ieşirea de date. În această situaţie se apelează la un mecanism de recuperare. Astfel de sisteme se numesc SMR (Safe Modular Redundant). Un sistem SMR conţinând *n* module este un sistem *n*SMR. Fiabilitatea şi siguranţa unui sistem *n*SMR depinde atât de fiabilitatea modulului individual cât şi de strategia particulară de arbitrare selecţionată.

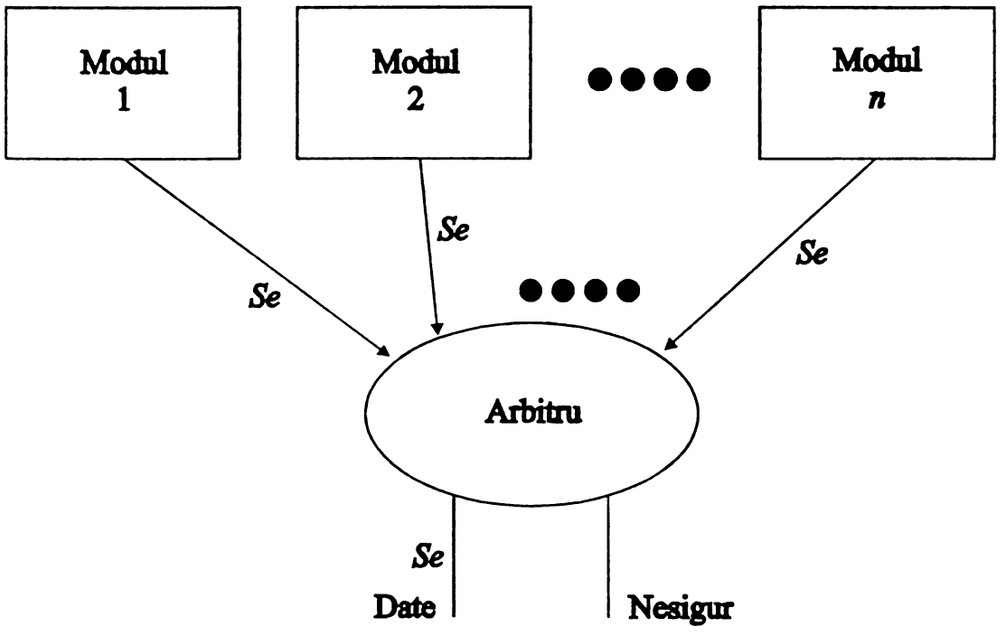


Fig. 4.2 Un sistem SMR

O strategie de arbitrare reprezintă funcţia implementată de către un arbitru pentru a decide ce ieşire corectă există şi când erorile de la ieşirile de modul exced capacitatea de corectare de o astfel de manieră încât ieşirea corectă nu poate fi furnizată. Când ieşirea corectă nu poate fi furnizată, fanionul de indicare a nesiguranţei poate fi poziţionat de către arbitru şi astfel se poate renunţa la evaluarea ieşirii.

În general în cadrul unei abordări uzuale se presupune că fiabilitatea şi siguranţa în funcţionare ar putea să fie ameliorate prin adăugarea unor capacităţi suplimentare de redundanţă. În literatură [Prad96] se arată că o redundanţă augmentată nu trebuie în mod necesar să contribuie la o îmbunătăţire atât a fiabilităţii cât şi a siguranţei în funcţionare. Mai exact, pentru un model de eroare simetric, care modelează erori în circuitele VLSI trebuie să se reţină următoarele: în sisteme cu module non-redundante

(module cu ieşiri necodate) există un sistem *n*SMR în comparație cu care nici un sistem (*n*+1)SMR nu poate atinge o fiabilitate mai ridicată şi nici o siguranță în funcționare mai bună, ceea ce înseamnă că adăugarea unui modul nu ameliorează în mod simultan fiabilitatea şi siguranța. Oricum, fiind dat un sistem *n*SMR întotdeauna există un sistem (*n*+2)SMR cu o fiabilitate mai ridicată (siguranță) şi cel puțin acelaşi nivel de siguranță (fiabilitate). Astfel, adăugarea unui minim de două module poate fi întotdeauna efectivă în ameliorarea atât a fiabilității cât şi a siguranței. Prin urmare se poate demonstra că dacă există module individuale având o redundanță inclusă, întotdeauna există un sistem (*n*+1)SMR cu o fiabilitate mai mare şi cu o siguranță în funcționare comparabilă cu oricare alt sistem *n*SMR dat. Astfel, dacă se utilizează module cu capacitate inclusă de detectare, adăugarea unui singur modul poate fi suficientă. Dacă modulele nu au capacitate de detectare inclusă atunci adăugarea unor capacități suplimentare de redundanță nu garantează întotdeauna ameliorarea siguranței. Prin urmare, adăugarea unui singur modul poate să întărească atât fiabilitatea cât şi siguranța doar în condițiile existenței unor capacități incluse de detectare a erorii. Acest lucru e valabil doar în ipoteza simplificatoare că toate configurațiile de eroare sunt aproximativ simplificatoare. În sistemele critice din punct de vedere al siguranței, una dintre problemele majore este reprezentată de acuratețea modelului de defect. Prin urmare, printre modelele de defect permanente tranzitorii şi intermitente se utilizează un model cu grad mare de generalitate denumit *modelul Bizantin*.

4.2.1c Redundanța pentru tolerarea defectelor *non-clasice*

În timp ce cea mai simplă abordare pentru mascarea redundanței implică calcule triplicate, unde votarea asupra rezultatelor are loc pentru a tolera o singură eroare permanentă sau una intermitentă, votarea devine mult mai complicată în prezența defectelor arbitrare non-clasice. În mediile ce necesită o extrem de înaltă fiabilitate, metodele de votare trebuie să fie furnizate cu posibilitatea de o oferi toleranță împotriva defectelor arbitrare. Chiar şi defectele maligne, unde două sau mai multe noduri defecte pot coopera, trebuie să fie tolerate.

În acest scop, a fost propus *modelul de defect al Acordului Bizantin* în 1982 de către Lamport, Shostak şi Pease [LSP82]. Acest model permite o comportare arbitrară pentru modulul ce defectează. Un astfel de modul poate să producă nu numai valori incorecte dar de asemenea să transmită valori diferite la destinații diferite în loc de valori identice cum ar fi de aşteptat. Sub acest model, două sau mai multe module ce

defectează pot accidental sau deliberat să coopereze de o astfel de manieră încât să producă un efect mult mai dezastruos asupra sistemului, chiar dacă defectele maligne la nivel de modul au fost tolerate.

Agrementul Bizantin, reprezintă o analogie tradițională utilizată pentru prezentarea dificultăților întâlnite în găsirea unui agrement între procesoarele ce operază corect în prezența unor procesoare ce defectează, chiar dacă aceste procesoare manifestă un comportament ce poate fi interpretat ca fiind intenționat malign.

În SMP critice din puncte de vedere al siguranței, unii proiectanți susțin acest model pentru a depăși efectul defectelor complexe dependente de timp ce sunt mult mai ușor acoperite de acest model.

În raport cu nivelul de fiabilitate cerut, proiectarea unui sistem bazat pe agrementul bizantin este totuși simplu. Un astfel de sistem conține doar un număr minim, în prealabil specificat, de procesoare și interconexiuni, furnizează sincronizarea lor și utilizează protocoale simple de schimb de informații.

Cu titlu de exemplu, amintim că pentru un protocol ce tolerează f defecte, bazat pe un sistem Bizantin f-rezilient, trebuiesc îndeplinite următoarele cerințe:

- Cel puțin $(3f+1)$ membrii trebuie să participe
- Cel puțin $(2f+1)$ căi disjuncte de comunicație trebuie să existe între membri
- Cel puțin $(f+1)$ runde de comunicare trebuie să aibă loc

Astfel, un protocol pentru un sistem rezilient de tip 1-Bizantin trebuie să execute două runde de comunicare între cel puțin patru membri sincronizați conectați la cel puțin trei căi disjuncte de comunicație.

Pentru a specifica cerințele ce trebuie satisfăcute de un protocol bazat pe Agrementul Bizantin, în literatură [Prad96] se utilizează exemplul clasic al unui sistem de calcul compus din noduri ce comunică între ele. Nodul principal trebuie să trimită o comandă de tip ATAC sau RETRAGERE către toate nodurile subordonate. Nodul principal și nodurile subordonate utilizează un protocol de tip Agrement Bizantin pentru a conveni asupra comenzilor ce emană de la nodul principal. Nodurile trebuie să comunice prin mesaje și să ajungă la o decizie finală. Un nod subordonat care este defect poate, prin urmare, să transmită mesaje incorecte. Cerințele ce trebuie să fie satisfăcute de un algoritm bazat pe acest tip de agrement poate fi rezumat astfel:

C1: toate nodurile subordonate corecte agrează aceeași comandă

C2: dacă nodul principal funcționează corect toate nodurile subordonate ce funcționează corect sunt de acord asupra comenzii furnizate de către nodul principal

De exemplu se consideră patru noduri *A*, *B*, *C* şi *D*. În ipoteza că nodul *D* doreşte să transmită un mesaj către *A*, *B* şi *C* Agrementul Bizantin impune ca toate nodurile ce funcționează corect într-un set {*A*, *B*, *C* şi *D*} să fie de acord asupra aceluiaşi mesaj. Dacă *D* este un nod principal şi *A*, *B* şi *C* noduri subordonate, *D* poate trimite un mesaj fie ATAC, fie RETRAGERE, situație prezentată în fig. 4.3.

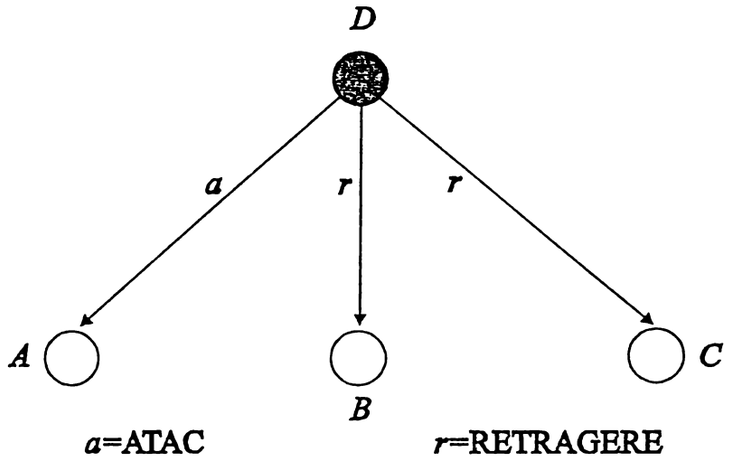


Fig.4.3 Agrementul Bizantin

Nodurile subordonate libere de defect pot să fie de acord asupra comenzii trimise de *D* dacă *D* este liber de defect. Dacă *D* este defect atunci nodurile subordonate libere de defect sunt solicitate să fie de acord asupra unei comenzi identice, indiferent de comanda recepționată de la *D*. Se poate considera următorul scenariu: *D* defectează şi *A*, *B* şi *C* sunt libere de defect. *D* trimite comanda ATAC spre *A* şi retragere spre *B* şi *C*. Pentru a detecta un astfel de caz nodurile subordonate trebuie să schimbe între ele mesajul recepționat de la *D*. După ce schimbul a fost efectuat, fiecare dintre subordonați are trei copii ale acestui mesaj. Subordonatul acceptă majoritatea acestor copii ca fiind valoare agreată. Astfel, fiecare dintre nodurile subordonate înțelege comanda RETRAGERE ca fiind o valoare agreată, situație reprezentată în fig. 4.4.

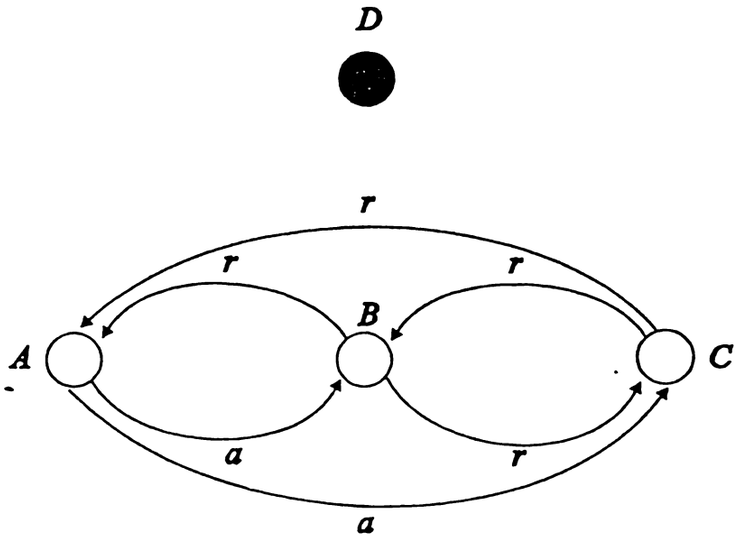


Fig. 4.4 Agrementul Bizantin obținut după aplicarea protocolului

Un alt scenariu posibil este acela în care două dintre nodurile subordonate, *A* şi *B* sunt defecte şi nodul principal este liber de defect. Nodul principal emite comanda ATAC către toate cele trei noduri subordonate. Nodurile *A* şi *B* se comportă în mod defectuos şi trimit comanda RETRAGERE nodului *C*, situație prezentată în figura 4.5.

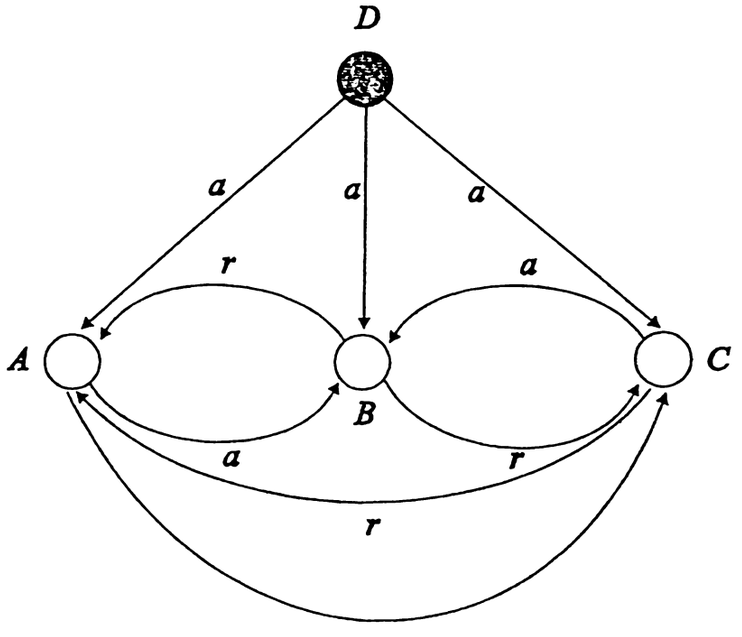


Fig. 4.5 Un agrement Bizantin imposibil

Prin urmare, *C* va obține comanda retragere ca o majoritate a celor trei copii ale comenzii recepționate. Astfel, două noduri subordonate defecte pot încălca acest algoritm bizantin. Este demonstrat că orice sistem ce poate tolera *t* defectări după

modelul bizantin trebuie să conțină cel puțin $3t+1$ noduri. Aceasta înseamnă că pentru a tolera defectarea celor două noduri subordonate, sistemul prezentat ca și exemplu ar trebui să aibe un nod principal și șase noduri subordonate. Aceasta înseamnă că un model de reziliență pentru m defecte necesită cel puțin $(3m+1)$ noduri ce comunică între ele. În literatură se prezintă un algoritm OM(m) bazat pe algoritmul Bizantin ce tolerează m defecte de tip bizantin. Numărul total n de noduri trebuie să satisfacă relația $n \geq 3m+1$. Termenii ATAC și retragere pot fi interpretați ca fiind mesaje binare.

Algoritm A. OM(0)

1. Valoarea emisă de nodul principal este transmisă fiecărui nod subrodonat.
2. Fiecare nod subordonat utilizează valoarea recepționată de la nodul principal sau utilizează valoarea RETRAGERE dacă nu a primit nici un mesaj.

Algoritm B. OM($t>0$)

1. Valoarea provenind de la nodul principal este transmisă la fiecare nod usbordonat.
2. Pentru fiecare i, fie v_i valoarea pe care nodul subordonat unic a recepționat-o de la nodul principal sau să fie RETRAGERE dacă nu s-a recepționat nimic. Subordonat lui i acționează ca și nod principal în algoritmnul OM(t-1) pentru trimite valoarea v_i la fiecare din celelalte (n-2) noduri subordonate.
3. Pentru fiecare i și pentru fiecare $j \neq i$ fie v_j valoarea nodului subordonat i transmisă către nodul subordonat j în pasul 2 (utilizând algoritmul OM(t-1) sau RETRAGERE dacă nu s-a recepționat nimic. Nodul subordonat i utilizează valoarea *majoritate* (v_1, v_2, ..., v_{n-1}).

În literatură [Prad96] se propune o formă diferită a protocolului Agrementului Bizantin denumită *Protocol de Agrement Degradabil*. În acest protocol se definesc doi parametri m și u unde $u \geq m$. Paramentrul m reprezintă numărul de defecte până la care protocoalele definite anterior pot fi efective, iar u este cel mai mare număr de defecte până la care protocolul degradat poate fi util. Acest protocol definit de paramentrii m și u este definit ca și *Protocolul de Agrement Degradabil m/u*. În continuare prezentăm acest protocol.

Protocolul de Agrement Degradabil

Fie f numărul de noduri defecte.

- Dacă $f \leq m$, atunci condițiile D.1 și D.2 enunțate mai jos trebuie să fie satisfăcute.

- Dacă $m<f\leq u$ atunci condițiile D.3 şi D.4 enunțate mai jos trebuie să fie satisfăcute.

(D.1) Dacă transmițătorul este liber de defect atunci toate receptoarele libere de defect trebuie să fie de acord asupra valorii transmise de transmițător.

(D.2) Dacă transmițătorul este defect, atunci toți receptorii liberi de defect trebuie să fie de acord asupra unei valori identice.

(D.3) Dacă transmițătorul este liber de defect, atunci receptorii liberi de defect trebuie să fie partiționați în cel mult două clase. Receptorii liberi de defect dintr-o clasă trebuie să fie de acord asupra valorii transmise de transmițător, şi receptorii liberi de de defect din cealaltă clasă trebuie să accepte valoarea de defect.

(D.4) Dacă transmițătorul este defect, atunci receptorii liberi de defect trebuie să fie partiționați în cel mult două clase. Receptorii liberi de defect dintr-o clasă trebuie să fie de acord asupra valorii de defect şi receptorii liberi de defect din cealaltă clasă trebuie să agreeze asupra unei valori identice.

Fie N numărul de noduri din sistem. Dacă $N>2m+u$ atunci agrementul degradabil m/u definit anterior asigură că cel puțin $m+1$ noduri libere de defect agrează o valoare identică chiar dacă numărul de defecte este mai mare decât m dar cel mult u. Astfel, se poate obține o degradare grațioasă. De notat că degradarea grațioasă este posibilă până la u defecte chiar când $u\geq N/3$.

În mod specific, pentru a obține un agrement degradabil m/u sistemul trebuie să consiste din cel puțin $2m+u+1$ noduri, inclusiv transmițătorul, şi aceste $2m+u+1$ noduri sunt suficiente.

Mulți algoritmi bazați pe agrementul bizantin au fost propuşi în diverse ipoteze referitoare la sistemul luat în considerare. De exemplu sistemele FTPP (*Fault Tolerant Parallel Processor*) utilizează astfel de protocoale în diverse circumstanțe.

4.2.2 Toleranța la defect prin *redundanța dinamică*

O tehnică la toleranță la defect cu costuri scăzute în SMP, utilizează *redundanța dinamică*. În această tehnică se prevăd capacități interne de detectare a defectelor în procesoarele ce defectează. Când se detectează un defect, sistemul este reconfigurat prin activarea unui procesor de rezervă sau a unei capabilități de procesare alternative. Această abordare furnizează un mecanism ce indică apariția unei erori în timp ce

sistemul operează. Acest lucru poate fi obținut utilizând fie resurse hardware, fie redundanță temporală. Odată ce prezența unui procesor defect a fost detectată, este demarată o procedură de localizare şi de diagnosticare a erorii. Procesorul defect este înlocuit cu un procesor de rezervă prin proceduri de reconfigurare. În final, se efectuează o recuperare din defect, unde procesorul de rezervă, utilizând informații tipice de exploatare a PV-urilor, preia calculele procesorului defect din locul de unde acesta a ieşit din sistem.

4.3 DETECTAREA DEFECTELOR ÎN SISTEMELE MULTIPROCESOR

În vederea detectării defectelor în sistemele multiprocesor există câteve abordări [Prad96], [MP89], [PF90]. Acestea sunt:

- *testare off-line* planificată pentru depistarea defectelor permanente;
- *duplicarea* şi *compararea* utilizând redundanța de spațiu si de timp;
- aplicarea unor *tehnici de codificare* şi de *diagnosticare*.

4.3.1 Detectarea defectelor prin *duplicare* şi *comparare*

Operațiile de *duplicare* şi *comparare* pot fi efectuate în mai multe moduri. Fiecare procesor a SMP poate fi duplicat şi rezultatele comparate înainte de a fi comunicate perechilor de procesoare. Această tehnică a fost aplicată la blocurile constitutive ale lui Intel 432.

Un mod echivalent de a trata această opțiune constă în divizarea celor P procesoare ale unui SMP în $P/2$ perechi. Memoria globală comună care constă din M module de memorie este şi ea divizată în $M/2$ perechi. Duplicarea modulelor de memorie nu se utilizează deoarece codificarea poate oferi tehnici mult mai eficiente de detecție şi corecție pentru aceste module. Comparatoarele pot fi încapsulate în interiorul fiecărui modul procesor şi de memorie şi rezultatul calculelor poate fi realizat hardware la fiecare ciclu de maşină în timpul fiecărei operații cu referire la memorie.

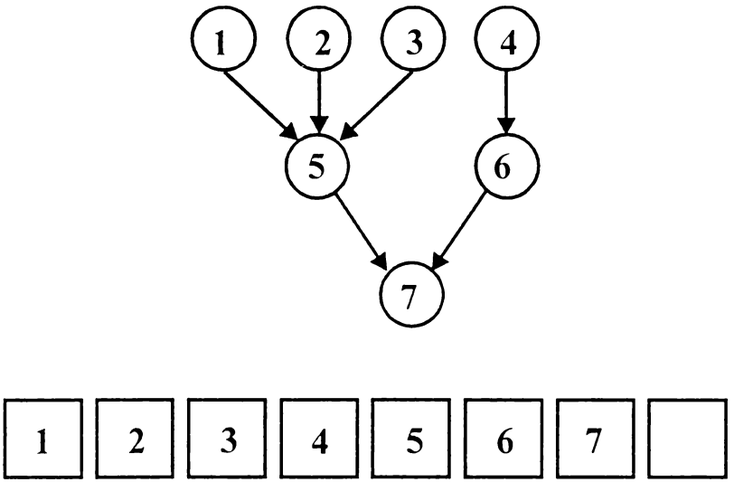
Dacă se detectează o eroare de către o pereche de procesoare, ambele sunt deconectate de la sursa de alimentare şi calculele se efectuează pe cele $P - 2$ procesoare ce rămân şi care sunt configurate ca şi $(P - 2)/2$ perechi.

Intr-o altă abordare, duplicarea şi compararea sunt utilizate cu *redundanță temporală*. O astfel de tehnică se utilizează când constructorul nu-şi poate permite duplicarea

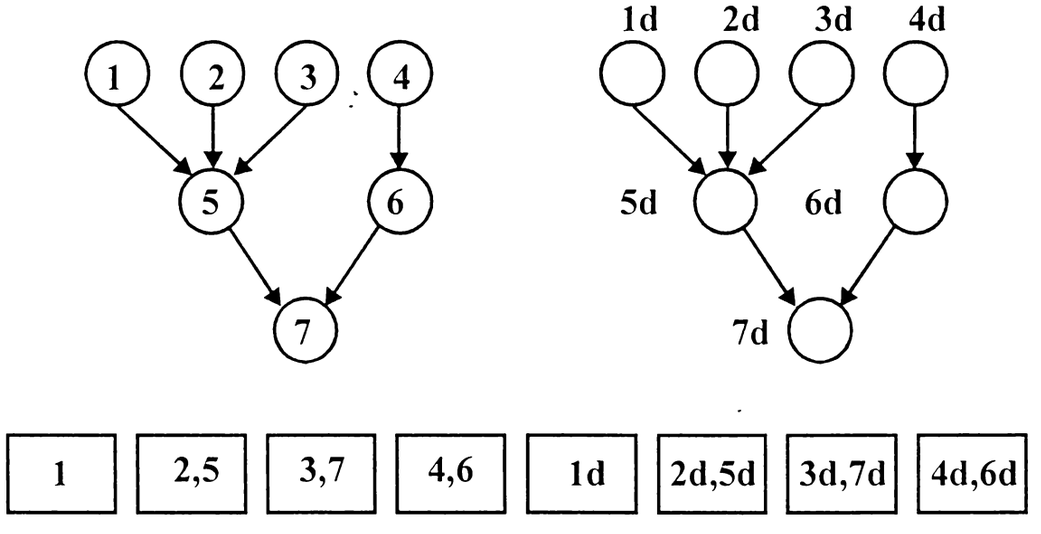
datorită restricțiilor de cost, greutate, putere, etc. ca de exemplu în aplicațiile spațiale. In principiu, cele *P* procesoare dublează acelaşi calcul paralel în timp asupra unui set diferit de procesoare, astfel încât fiecare procesor afectează doar o copie a rezultatului. Acest rezultat se poate obține prin divizarea celor *P* procesoare în *P*/2 grupe de câte două procesoare, fiecare din ele comparând rezultatele perechii înainte de a comunica cu alte procesoare.

În figura 4.6 se prezintă un exemplu de *graf de sarcină (task graph)* care este duplicat şi comparat pentru detectarea defectelor. Cazul (a) prezintă graful original ce este mapat pe opt procesoare. Cazul (b) prezintă duplicarea grafului şi maparea fiecărei copii pe un set disjunct de patru procesoare. Rezultatele celor două copii sunt comparate de două ori, de fiecare dată de câte un procesor din fiecare subgrupă. In cea mai defavorabilă situație procedura de mai sus poate să consume 100% un timp suplimentar.

Dacă graful nu are dependențe, sarcinile pot fi mai eficient mapate pe toate procesoarele unui SMP şi toate procesoarele sunt ocupate tot timpul. În prezența dependențelor, unele procesoare vor fi în stare de aşteptare activă deoarece există sarcini nepregătite pentru prelucrare. În astfel de situații se poate mapa graful original pe *P*/2 procesoare (ca şi în fig. 4.6 b), dându-se o mai bună utilizare a procesoarelor, iar cele *P*/2 procesoare ce rămân sunt utilizate pentru calculul dublat al grafului. Compararea poate fi efectuată şi prin tehnici software, cu mențiunea că în această situație granularitatea calculului, înainte de comparare, poate fi mai mare decât un ciclu maşină, deoarece compararea software durează mai mult decât cea hardware. Operația de comparare poate fi efectuată eficient prin utilizarea *punctelor de control* (*check point*).



a. Maparea grupului original



b. Maparea grupului duplicat

Fig. 4.6 Exemplu de mapare a grafurilor duplicate pe seturi disjuncte de procesoare.

4.3.2 Detectarea defectului utilizând *diganosticarea* şi *tehnici de codificare*

Diagnoza utilizează programe de diagnosticare ce pot fi utilizate pentru detectarea defecțiunilor hard în procesoare, memorii şi RIN. Această metodă detectează defecte permanente dar nu poate fi aplicată pentru defecte tranzitorii şi intermitente. Pentru detectarea defectelor tranzitorii şi intermitente se utilizează tehnici concurente de

detectare a defectelor ce utilizează metode de *autotestare* [Prad 96]. *Tehnicile de codificare* pot furniza detectări de defect la preț scăzut pentru magistrale, memorii şi registre. Verificări simple de paritate în magistrale şi registre sunt un exemplu de utilizare efectivă a codificării. Violarea parității provoacă situația de excepție. Revenirea este manipulată prin reîncercarea efectuării instrucției (instruction retry) sau prin proceduri similare. Arhitecturile cele mai moderne oferă capabilități de reîncercare pentru aproape toate instrucțiile. Un exemplu îl constituie sistemele IBM unde verificări intensive de paritate sunt prevăzute intern pentru a obține o diagnoză rapidă.

Implementarea toleranței la defect a decodificatoarelor şi a codificatoarelor este o chestiune de maximă importanță.

In codificarea utilizată în tehnicile de calcul, se poate avea uneori încredere în informația apriori referitoare la localizarea erorii.

4.3.3. Controlul erorii în memoriile de mare viteză.

Codurile corectoare de erori se utilizează atât la memoriile cache de mare viteză cât şi la memoriile principale datorită faptului că acestea sunt vulnerabile la erori soft. Pentru ca un cod să fie util pentru memoriile de mare viteză, structura sa trebuie să permită o codificare şi o decodificare paralelă, rapidă, compexitatea circuitului de verificarea parității utilizată în componența decodificatoarelor şi a codificatoarelor poate fi un factor major în determinarea vitezei. Examinarea structurii *matricii de verificare la paritate*, cunoscută sub numele de matrice **H**, dă informații referitoare la complexitatea circuitului de verificare a parității.

De exemplu, se consideră un cod de lungime 6, $n = 6$, cu trei biți informaționali, $k = 3$ şi trei biți de verificare, $r = 3$. Cele două matrici **H** prezentate în continuare au aceeaşi capabilitate de corectare a erorii, d_0, d_1 şi d_2 reprezentând data iar c_0, c_1 şi c_2, verificarea de paritate. Bitul de verificare a parității c_i reprezintă verificarea de paritate efectuată asupra lui d_i dacă coloana i a matricii **H** conține un 1 în coloana d_j. Deoarece toate coloanele în matricea **H** sunt disjuncte, codul poate corecta toate erorile singulare, dar circuitul de verificare la paritate pentru $\mathbf{H}_1$ este mai puțin complex decât pentru $\mathbf{H}_2$. Se consideră ecuațiile pentru c_0, c_1 şi c_2 - biții de control la paritate – separat pentru $\mathbf{H}_1$ şi $\mathbf{H}_2$. $\mathbf{H}_1$ necesită doar trei porți SAU-EXCLUSIV

cu 2 intrări pentru calculul lui c_0, c_1 şi o poartă SAU-EXCLUSIV cu trei intrări pentru calculul lui c_2. Datorită porţii SAU-EXCLUSIV cu trei intrări, codificatorul şi decodificatorul utilizând $\mathbf{H}_2$ vor fi mai lente şi mai complexe.

$$\mathbf{H}_1 = \begin{array}{c} \begin{matrix} d_0 & d_1 & d_2 & c_0 & c_1 & c_2 \end{matrix} \\ \left[\begin{matrix} 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 \end{matrix}\right] \end{array} \qquad \begin{matrix} c_0 = \overline{d_0 \oplus d_2} \\ c_1 = \overline{d_1 \oplus d_2} \\ c_2 = \overline{d_0 \oplus d_1} \end{matrix} \qquad (4.12)$$

$$\mathbf{H}_2 = \begin{array}{c} \begin{matrix} d_0 & d_1 & d_2 & c_0 & c_1 & c_2 \end{matrix} \\ \left[\begin{matrix} 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 \end{matrix}\right] \end{array} \qquad \begin{matrix} c_0 = \overline{d_0 \oplus d_1} \\ c_1 = \overline{d_0 \oplus d_2} \\ c_2 = \overline{d_0 \oplus d_1 \oplus d_2} \end{matrix} \qquad (4.13)$$

Numărul de 1-uri din matricea **H** determină complexitatea totală a circuitului de verificare a parităţii. Cu cât mai puţini de 1, cu atât mai simplu circuitul. Prin urmare, circuitul de verificare a parităţii cel mai lent corespunde cu linia cu numărul maxim de 1-uri şi numărul de 1-uri în orice linie la un minim.

Coduri detectoare/corectoare de erori pentru un bit. În memoriile de mare viteză *coduri corectoare de erori pentru un singur bit* (*single-bit error-correcting codes*) şi *coduri detectoare de erori pentru doi bit* (*double-bit error-detecting codes*)- coduri SEC-DED- sunt cele mai utilizate. Aceasta pentru că multe memorii RAM sunt organizate pe un bit; prin urmare orice defect într-un circuit integrat se manifestă ca şi o eroare pe un bit.

Se consideră două *r*-tuple de pondere impară. Ponderea se referă la numărul de 1-uri. De notat că suma a două *r*-tuple de pondere impară este o *r*-tuplă de pondere pară. Datorită acestei proprietăţi, un cod SEC-DED cu *r* biţi de verificare poate fi construit după cum urmează: aici matricea **H** constă din vectori coloană de pondere impară. Astfel, sindromul de eroare pe doi-bit este o *r*-tuplă de pondere pară. Acest cod este

diferit de *codul SEC-DED Hamming* original, a cărui matrice **H** are în plus la matricea **H** a codului SEC adăugat, un vector coloană de 1-uri.

Pentru a obține un circuit de codificare/decodificare de mare viteză, numărul de 1-uri în fiecare linie este egal sau cât mai apropiat de numărul mediu. Lungimea de cod maximă este egală cu numărul maxim de *r*-tuple de pondere pară şi prin urmare $n = 2^{r-1}$. In continuare se prezintă un exemplu din această clasă de coduri:

$$\mathbf{H} = \left[\begin{array}{cccc|cccc} d_0 & d_1 & d_2 & d_3 & c_0 & c_1 & c_2 & c_3 \\ 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \end{array}\right] \quad (4.14)$$

Sindrom: $S = D \cdot \mathbf{H}^{\mathrm{T}}[S_0, S_1, S_2, S_3]$

$$S_0 = \overline{d_0 \oplus d_1 \oplus d_2 \oplus c_0}$$

$$S_1 = \overline{d_0 \oplus d_1 \oplus d_3 \oplus c_1}$$

$$S_2 = \overline{d_0 \oplus d_2 \oplus d_3 \oplus c_2}$$

$$S_3 = \overline{d_1 \oplus d_2 \oplus d_3 \oplus c_3}$$

Pentru orice cod SEC-DED, probabilitatea de funcționare eronată la apariția unor erori triple sau pe mai mulți biți trebuie să fie minimală. O funcționare eronată se referă la o decodificare eronată ce se transformă într-un cod corector eronat.

In concluzie, codurile SEC-DED cu coloane de pondere impară au două avantaje practice: simplitate constructivă a codificatorului/decodificatorului şi posibilitate scăzută a unei decodificări eronate.

Coduri detectoare-corectoare de erori pe byte. Unele memorii sunt organizate în capsule de mare densitate,având o lățime de *b*-bit. Dacă apare o defecțiune, cuvântul extras se comportă de parcă ar avea un bloc de *b*-bit (byte) defect. In acest tip de aplicații este de dorit să existe un cod corector de erori capabil de detectare/corectare a erorilor la nivel de byte.

Coduri corectoare de erori la nivel de byte. In literatură [FP 90] se face o prezentare exhaustivă a acestei clase de coduri. Vom prezenta modul de construcția matricii **H** pentru un cod corector de eroare la nivel de byte, astfel: Se aleg ca şi coloane ale

matricii **H** toate *r*-tuplele nonzero de elemente dintr-un câmp finit *F*, în particular dintr-un câmp Galois $GF(2^b)$, astfel că nici o coloană din **H** să fie multiplu al altei coloane. Astfel, fiecare pereche de coloane este linear independentă şi prin urmare există o distanţă Hamming minimă de 3 pentru cod. Codul cunoscut sub acronimmul de SbEC, este capabil să corecteze toate erorile singulare la nivel de cuvânt de *b*-bit.

Implementarea acestui tip de cod necesită transformarea matricei **H** din $GF(2^b)$ într-o formă binară. Prin utilizarea unor funcţii polinomiale primitive binare, *g (x)* de grad *b*, se poate defini o matrice nesingulară **T** exprimată ca şi o matrice binară *(b*x*b)*. Setul acestor matrici reprezintă un câmp ce este izomorfic la $GF(2^b)$. Prin urmare, elementele lui $GF(2^b)$ pot fi exprimate ca şi $\{\mathbf{0}, \mathbf{T}, \mathbf{T}^2, \mathbf{T}^3 \ldots, \mathbf{T}^{2b-2}, \mathbf{T}^{2b-1} = \mathbf{I}\}$, unde **I** este matricea *(b*x*b)* de valoare 0.

Simbolurile *n* şi *k* denotă lungimile de cod şi lungimile de informaţie ale acestui cod peste $GF(2^b)$. Codul derivat *Sb*EC este un cod *(N,K)* în formă binară, unde $N = n \cdot b$ şi $K = k \cdot b$. Similar, numărul de biţi de verificare este $R = r \cdot b = n - k$.

Lungimea maximă (în bit) a acestui cod de tip Hamming este dată de $N_H = b \cdot n = b \cdot ((2^{br} - 1)/(2^b - 1))$.

Deoarece codurile SbEC nu garantează detectarea erorilor aleatorii pe doi bit, respectiv pe doi bytes, aceste coduri nu sunt utilizate în sistemele de calcul. In schimb, calculatoarele utilizează coduri corectoare de erori pe cuvânt de b-bit şi coduri detectoare de erori duble pe cuvânt de b-bit, denumite *Sb*EC-*Db*ED.

4.4.STRATEGII DE RECUPERARE PENTRU SISTEME MULTIPROCESOR

Recuperarea unui SMP tolerant la defect, reprezintă ultimul pas în procedura de revenire a sistemului după ce defectul a fost detectat, izolat şi sistemul reconfigurat în jurul procesorului (procesoarelor) defect(e) [Sita93], [VP93], [WFP90].

Tehnicile de recuperare sunt diferite pentru SMP-MP respectiv SMP-TM.

Soluţia cea mai trivială de recuperare în cazul apariţiei unui defect în unul sau mai multe procesoare în cadrul unui multiprocesor o reprezintă terminarea programului în lucru şi reexecutarea întregului program, pentru toate procesoarele, de la început. Este procedura *restartului global* (*global restart*). Evident această soluţie este costisitoare

datorită timpului consumat. Practic timpul de rulare se dublează. Din această cauză această soluție nu este considerată ca şi o procedură standard de recuperare din defect.
Recuperarea din defecțiune poate avea trei forme [Prad 96]:

- *recuperare prin rulare înapoi (rollback recovery)*: Transformarea stării eronate constă din aducerea înapoi a sistemului la o stare ocupată tot timpul înainte de apariția erorii;
- *recuperare prin rulare-înainte (forward recovery)*: Transformarea stării eronate constă din găsirea unei noi stări, din care sistemul să poată opera, de obicei într-un mod degradat;
- *compensarea(compensation)*: Starea eronată conține suficientă redundanță pentru a permite transformarea sa într-o stare liberă de eroare.

Problema recuperării prin *rulare înapoi* este complexă în cazul SMP deoarece procese multiple pot accesa memoria şi pot avea copii diferite sau eronate ale aceloraşi variabile şi prin urmare producând o stare inconsistentă când eroarea este detectată. În consecință schemele în care se aplică tehnica recuperării prin *rulare înapoi* trebuie să asigure memorarea informațiilor referitoare la stările libere de erori într-un loc sigur de unde să poată fi apoi regăsite şi utilizate pentru restartarea programului dintr-o stare consistentă.În esență recuperarea prin *rulare înapoi* constă din salvarea periodică a unui proces în timpul evoluției corecte a unui program de o astfel de manieră încât restartarea să fie posibilă dintr-o astfel de stare salvată. Stările salvate ale procesului se numesc *puncte de verificare (checkpoints)* -PV- iar tehnica de creare şi de exploatare a astfel de puncte se numeşte *checkpointing*.
Starea unui sistem, în general, implică setul de registre ale procesorului, numărătorul de program, starea memoriei cache şi uneori chiar memoria cel puțin până la ultimul punct de verificare – PV.
Aceste PV-uri sunt memorate în unități de memorie sigure, discuri sau memorii protejate de coduri corectoare de erori.

A. Tehnica recuperării prin rulare înapoi utilizând puncte de verificare

Această tehnică ce utilizează PV-uri este o metodă de a asigura toleranță la defect împotriva defectelor tranzitorii şi a celor intermitente. In continuare se vor descrie câteva metode de implementare a acestei tehnici [BP92], [BP93], [Ng91].

- ***Puncte de verificare bazate pe memoria cache a procesului***

Datele punctului de verificare (checkpoint data) reprezintă starea celui mai recent PV, iar *datele active* reprezintă ceea ce este accesat după verificarea PV-ului.

Ierarhia memoriei oferă o soluție naturală pentru această problemă în sensul că datele active pot fi plasate în registrele procesorului şi în memoria cache în timp de datele PV pot fi plasate în memoria principală a sistemului. Astfel operația de bază pentru starea unui PV constă în a rămâne în memoria principală, iar procesorul avansează actualizând doar memoria cache şi registrele sale. Un PV este inițiat prin o *tentativă eşuată de acces la cache (cache miss)* ce forțează o linie cache modificată spre memoria principală. Un PV este obținut prin copierea tuturor registrelor modificate şi a liniilor cache în memoria principală. O *rulare înapoi (rollback)* se obține prin îndepărtarea tuturor registrelor modificate şi a liniilor cache şi prin restaurarea stării anterioare utilizând datele PV-ului. Această schemă devine operațională prin utilizarea unei politici de actualizare de tip "*copiază-înapoi*". Astfel memorările sunt efectuate în cache şi nu în memoria principală. Memoria principală este actualizată când apare un acces ratat, concretizându-se în înlocuirea unui linii viciate (dirty-line). O linie sau o pagină se consideră *viciată (dirty)* dacă ea a fost modificată de către deținătorul ei de la ultimul său PV. Dacă acest lucru nu s-a produs, atunci linia (pagina) este *curată (clean)*. Prin urmare un acces ratat la cache forțează un PV.

În fig. 4.7 se prezintă separarea stării active şi a stării PV-ului.

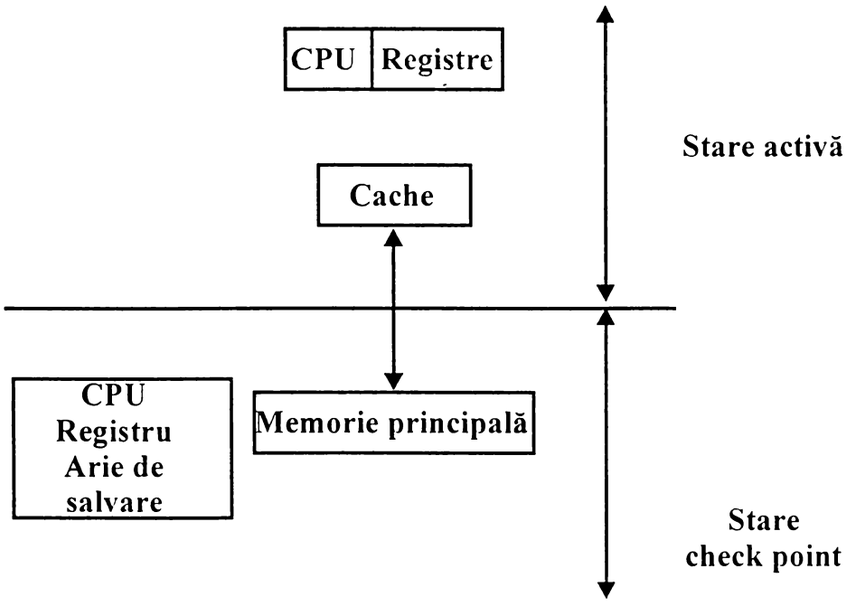


Fig. 4.7 Crearea de PV bazate pe procesor şi recuperarea prin revenire

Un controler ce controlează PV-ul şi procesul de revenire trebuie să aibă următoarele abilităţi care să:

- salveze registrele procesorului în memorie (sau în cache);
- aducă toate liniile cache modificate în memoria principală;
- încarce registrele CPU din memorie (sau din cache);
- invalideze toate liniile modificate în cache.

Dacă sistemul este într-o stare oprită cu memoria principală conţinând un PV valid, cu registrele CPU pentru această stare salvate în memoria principală (aria de salvare) atunci controlerul încarcă registrele din aria de salvare şi sistemul începe să ruleze. Evident iniţial registrele şi cache-ul sunt goale. Datele încep să fie încărcate la cerere în cache. Cache-ul utilizează o politică *copiază înapoi*, aşa că scrierile în memorie nu depăşesc spaţiul cache. Orice schimbări în stare de la ultimul PV se produc doar în cache şi în registre.

Aceste două seturi de locaţii înmagazinează starea activă. Dacă se detectează o eroare tranzitorie (de exemplu o eroare de paritate pe o magistrală internă), sistemul trebuie readus până la un PV anterior. Deoarece acesta este conţinut în întregime în memoria principală a sistemului, registrele CPU şi memoria cache trebuie restaurate la starea lor de la începutul intervalului. Memoria cache distinge între liniile nemodificate şi cele modificate aşa că doar liniile modificate sunt luate în considerare, ele reprezentând starea activă. Prin urmare, liniile cache modificate trebuie să fie selectiv invalidate.

In mod alternativ, întreaga memorie cache ar putea fi iniţializată. Următorul pas îl constituie reîncărcarea registrelor CPU din starea salvată în memorie. Acum sistemul a fost readus înapoi la PV anterior şi rularea programului poate reîncepe.

In orice schemă ce utilizează PV-uri este esenţial să existe posibilitatea manipulării defectelor în timpul exploatării PV-urilor. In literatură [Prad 96] se descriu mai multe scheme de principiu ce sunt utilizate în calculatoare de firmă şi în care se regăsesc principiile enunţate anterior. În fig 4.8 se prezintă şase paşi pentru redirecţionarea informaţiei din cache către memoria principală. Această tehnică poate tolera o singură defecţiune în orice punct.

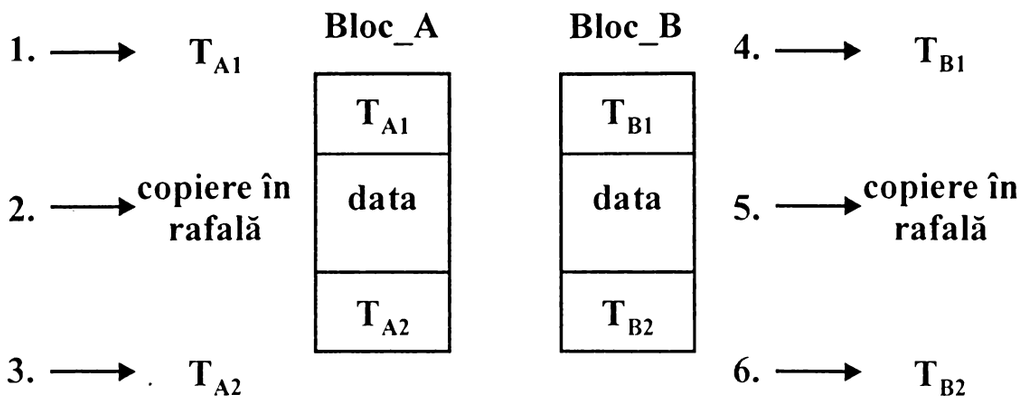


Fig. 4.8 Tehnici tolerante la defect pentru copierea în cascadă a cache-urilor

Conținutul cache-ului este scris secvențial în cele două blocuri de memorie. O *marcă ce marchează timpul (time stamp)* este scrisă în blocul de memorie înainte de *transferul în rafale (flush)*. Aceasta poate fi T_{A1} sau T_{B1}. După transferul în rafale eticheta va fi T_{A2} sau T_{B2}. Deoarece cele patru mărci de timp sunt scrise secvențial, o defecțiune poate fi descrisă de aceste etichete ce sunt diferite. Utilizând aceste patru mărci de timp, se poate corecta o defecțiune o procesului de checkpointing. TABEL VII descrie diferitele condiții ale mărcilor de timp găsite după defectare. Deşi, din punct de vedere logic, există patru mărci de timp distincte, în practică doar trei mărci sunt necesare deoarece $T_{A2} = T_{B1}$. În consecință, paşii de recuperare subliniați în liniile 3 şi 4 din TABEL VII sunt aceiaşi. În mod clar, dacă ei sunt egali, atunci procesul de creare şi gestionare a PV-urilor a fost efectuat cu succes. Dacă T_{A1} este singura marcă de timp scrisă, atunci defecțiunea a apărut în timpul transferului spre blocul A.

Noul PV (în blocul A) este incomplet şi PV anterior (blocul B) trebuie să fie utilizat. Astfel, procedura de recuperare va reveni înapoi la PV anterior prin copierea blocului A în blocul B.

Dacă ambii T_{A1} şi T_{A2} sunt scrişi, $T_{A1} = T_{A2}$, atunci noul PV a fost scris cu succes. Această situație derivă în două cazuri distincte de defectare ambele necesitând aceeaşi acțiune. În primul rând, dacă T_{B1} nu a fost scris, atunci defecțiunea a apărut între două transferuri. Oricum, doar dacă T_{B2} nu a fost scris, eroarea apare în timpul transferului spre B. În celălalt caz, noul PV din blocul A este copiat în blocul B.

TABEL VII. Condiții de defectare

Condiția	Defecțiune	Acțiune
$T_{A1} = T_{A2} = T_{B1} = T_{B2}$	Nici una	Nici una
$T_{A1} > T_{A2} = T_{B1} = T_{B2}$	Transfer în A	Copiază B în A
$T_{A1} = T_{A2} > T_{B1} = T_{B2}$	Intre	Copiază A în B
$T_{A1} = T_{A2} = T_{B1} > T_{B2}$	Transfer în B	Copiază A în B

Talia memoriilor cache afectează direct frecvența creerii şi gestionării PV-urilor. Cu cât mai mare este talia unui cache, cu atât mai mică va fi necesitatea de a forța un PV din cauza condiției de memorie cache plină. Pe de altă parte, frecvența utilizării PV-urilor este direct proporțională cu rata de defectare şi cu abilitatea de recuperare. În literatură [Prad 96] se oferă mai multe soluții de ameliorarea tehnicilor de creare şi gestionare a PV-urilor.

- ***Puncte de verificare virtuale***

O deficiență majoră a tehnicilor de creare şi exploatare a PV-urilor bazate pe memoria cache este aceea că frecvența de creare şi testare a PV-urilor, ce depinde de talia cache-ului, poate fi foarte ridicată afectând negativ performanțele.

Când frecvența de checkpointare este ridicată, o alternativă o constituie plasarea PV-urilor şi a recuperării prin *rulare înapoi* într-o memorie virtuală, prin extragerea lor din cache-ul procesorului.

Metoda *punctelor de verificare virtuale* (*virtual checkpoints*) este utilizată pentru a reduce penalitățile în performanțe ale unei testări a PV-urilor cu frecvență ridicată, prin utilizarea unui mecanism de translație şi a unei memorii adiționale. Metoda combină concepte provenite de la două tehnici de recuperare utilizate la baze de date [Prad 96]: *paginarea din umbră* (*shadow paging*) şi *paginarea geamănă* (*twin paging*).

Termenul de *date ale punctului de verificare* se referă la starea în cel mai recent PV iar cel de *date active* la acelea ce sunt accesate după PV. Conceptul de *suportare a datelor active* este implementat prin alocarea dinamică a unei copii secundare a paginii virtuale. Paginile active pot fi identificate prin utilizarea unui contor de PV-uri asociat fiecărei pagini.

Un aspect fundamental al acestei tehnici este acela al suportării a două clase de date în sistemul memoriei virtuale (date active şi PV-uri). Fiecare clasă suportă înmagazinarea tradiţională pe două niveluri a memoriei virtuale –memoria reală şi discul de paginare. Există două cerinţe importante ce se impun pentru această tehnică. Prima este abilitatea de a detecta toate paginile active. Aceasta deoarece o *rulare înapoi* la un PV anterior se obţine prin purjarea tuturor paginilor active. A doua cerinţă este abilitatea de a face permanente toate paginile active la momentul verificării PV-urilor. Aceste pretenţii sunt satisfăcute având un contor de PV global (V) ce acoperă toate contoarele de date şi de PV-uri locale (v) pentru pagini individuale. În esenţă, contorul global este copiat în contorul local la fiecare referire. Prin urmare prima cerinţă este îndeplinită deoarece toate datele active au contorul local v egal cu contorul global V. Contorul V este incrementat când este preluat un PV. Astfel, a doua cerinţă este îndeplinită deoarece toate paginile active versiuni PV când V este incrementat. Figura 4.9 ilustrează aceste concepte de bază. Pagina virtuală k nu a fost referită de la PV anterior. Pagina j a fost accesată în intervalul curent şi are atât o versiune activă cât şi una PV.

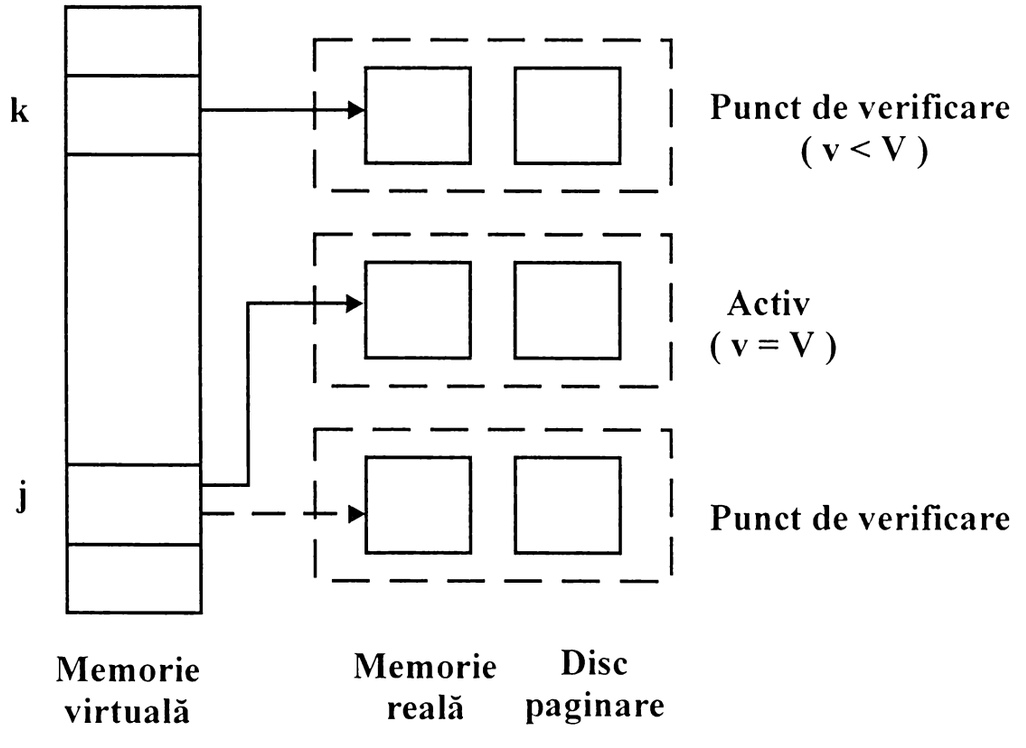


Fig.4.9 Conceptul de bază

Pentru a implementa această schemă, se prevăd câteva câmpuri care trebuie să fie adăugate la mecanismul de translatare virtuală. Trebuie să existe informații pentru a distinge între o versiune activă şi una de PV a fiecărei pagini. În continuare vom descrie translatarea unei singure pagini virtuale. Aceasta înseamnă că notarea va exclude orice referință la numărul paginii virtuale. Pentru fiecare pagină virtuală, mapările sunt replicate şi referite ca şi m_0 şi m_1. O mapare m_l conține mapări în cadrul real (r_l) şi copia disc (d_l) ca şi în fig. 4.10.

Fiecare pagină are un câmp de un bit, l, care poate fi imaginat ca şi un comutator ce indică cea mai recentă mapare utilizată (de exemplu m_0 şi m_1). Astfel, notarea m_l se referă la maparea ce a fost utilizată ultima. In plus, fiecare pagină are un număr (v) a unui PV local de k-bit ce conține o copie a unui număr (V) de PV global în timpul celei mai recente referiri. Numărul (V) este o valoare globală care este incrementată la fiecare PV. Fiecare cadru real, desemnat de r_l , conține un bit de schimb (c_l) care indică dacă rata din cadrul real a fost schimbată şi este diferită de copia disc la d_l.

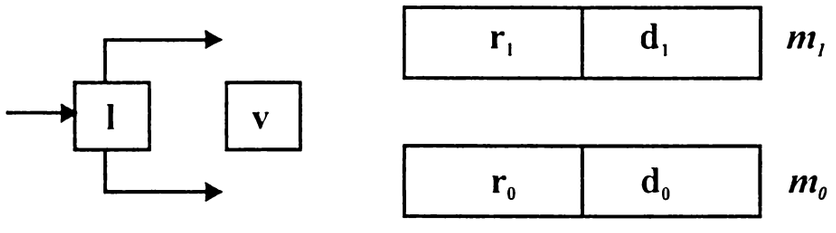


Fig. 4.10 Maparea unei singure pagini

O caracteristică importantă a acestei scheme este că acțiunile de preluare ale unui PV nu sunt concentrate la timpul actual ale PV ci sunt distribuite în timpul ce urmează PV-ului. Procesarea pentru paginile individuale este amânată până la ultima referire la pagina ce urmează după PV. Pentru a determina dacă procesarea amânată trebuie să apară, valorile V şi v trebuie să fie comparate la fiecare referire. Astfel, când o pagină este referită, apare fie cazul unde procesarea PV-ului trebuie să apară ($v \neq V$) sau un acces la pagina activă ($v = V$). Fig. 4.11 prezintă o situație unde PV-urile au fost prelevate la timpii t_{c1} şi t_{c2}.

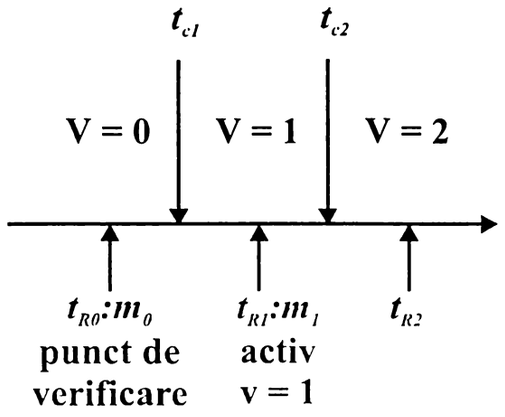


Fig.4.11 Cazul 1: Prima referire după PV

Să considerăm evenimentele la t_{R2}. Pagina activă adresată de m_1 (l=1)a fost ultima referită la t_{R1} (astfel, v=1). Referința la t_{R2} este prima referință după PV (deoarece v≠V) şi conținutul paginii m_1 trebuie să fie rezervat ca şi pagină PV. Mai departe, *conținutul* paginii m_1 trebuie să fie utilizată în timp ce *resursele* vechii pagini m_0 de tip PV sunt utilizate. Odată ce data validă a fost copiată în m_0 , bitul l este inversat şi devine l=0 astfel că m_0 devine pagina activă şi m_1 devine PV. În final, V este copiat în numărul PV-ului local pentru această pagină aşa că la următorul acces în intervalul de procesare al PV-ului, apare o translatare normală. Fig. 4.12 prezintă situația la următoarea referire în acest interval de preocesare al PV-ului. O referire la timpul t_{R3} urmează după data la m_0, deoarece V=v.

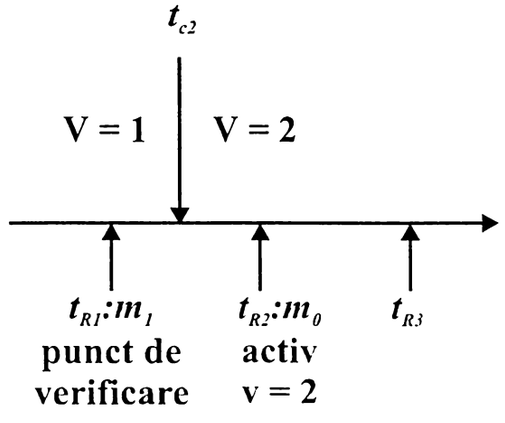


Fig. 4.12 Cazul 2: Pagină referită anterior

O rulare înapoi este efectuată prin îndepărtarea oricărei date ce a fost modificată după PV-ul anterior. Dacă pagina nu a fost încă referită după PV-ul anterior, atunci pagina este într-o stare obținută prin rularea înapoi şi nu trebuie efectuată nici o operație (de exemplu cazul 1 din fig. 4.11. Dacă pagina a fost referită dupa PV-ul anterior, atunci există o pagină activă ce trebuie îndepărtată.

De exemplu, dacă o defecțiune apare la t_{R3}, în fig. 4.12, se doreşte îndepărtarea lui m_0 şi restuararea lui m_1. Astfel pentru toate paginile cu $V=v$, valoarea v este decrementată şi bitul l este inversat.

4.4.1 Recuperarea din defect prin *rulare înapoi* în sistemele multiprocesor

Recuperarea prin rulare înapoi reprezintă un subiect complex în multiprocesoarele ce comunică între ele deoarece orice recuperare prin *rulare înapoi* trebuie să țină cont de toate comunicațiile şi de efectul lor după ultimul punct de verificare.

4.4.1.a Recuperarea prin *rulare înapoi* la SMP - MP

Memoriile cache utilizate în SMP - MP, ataşate fiecărui procesor, utilizează o strategie de *scriere - înapoi* (*write - back*) pentru a reduce traficul la memoria principală. Această strategie de menținere a coerenței admite existența la un moment dat a mai multor copii ale aceleaşi date în diferite memorii cache. Aceasta presupune utilizarea unor protocoale de menținere a coerenței în timpul modificării unei anumite copii. De exemplu orice procesor ce doreşte să modifice o copie poate transmite un mesaj astfel încât toate procesoarele ce dețin copii trebuie în prealabil să şi le invalideze. De asemenea, copia din memoria principală trebuie să fie actualizată în cel mai scurt timp posibil după ce procesorul modifică copia din cache. Utilizând tehnica *scrie - prin* (*write - through*), modificarea se obține instantaneu, în acelaşi timp cu modificarea copiei cache. Utilizând procedura *scrie - înapoi* (*write - back*), această actualizare este obținută când conținutul cache-ului este copiat înapoi datorită unei tentative eşuate de acces. Cele mai multe scheme utilizează această proprietate a procesoarelor cu cache-uri actualizate prin scriere înapoi, pentru a face o recuperare rapidă. Această optimizare nu se poate obține utilizând tehnica *scrie - prin* [Prad 96].

Prin urmare cele mai multe scheme de recuperare din defect pentru SMP - MP utilizează protocolul *scrie - înapoi* pentru menținerea coerenței.

Prezența cache-urilor implică faptul că nu numai un procesor poate avea copii ale aceleaşi date, unde o modificare subsecventă poate să conducă la situația în care mai

multe procesoare să aibă copii diferite ale aceleaşi date, deci să producă incoerenţa memoriilor cache şi principală. În consecinţă se utilizează schema ce etichetează datele în cache-uri pentru menţinerea coerenţei. Se utilizează [Prad96], [AFM90], [BP93], [HM87], [JF91] scheme bazate pe magistrală singulară şi bazate pe director. Cele mai multe scheme bazate pe magistrală utilizează politicile *invalidare de scriere* şi *actualizare de scriere*.În alte lucrări aceste protocoale sunt denumite *scrie – invalidează* ,respectiv *scrie – actualizează.*

Reamintim că esenţa schemei *invalidare de scriere* presupune că ori de câte ori într-o locaţie de memorie trebuie scris, se trimite un semnal ce invalidează toate copiile preexistente ale aceleaşi date dispuse în alte cache-uri înainte de actualizarea copiei locale din cache.

A doua categorie de scheme bazate pe magistrală, cele care utilizează tehnica *actualizare de scriere*, este mai bună dacă scrierile sunt mai puţin frecvente [Prad 96], deoarece ele implică livrarea datelor de scris la toate cache-urile simultan ori de câte ori un acces de scriere la un bloc cache partajat este generat. Schemele bazate pe director utilizează un *tabel (directory)* pentru a păstra evidenţa stărilor fiecărui bloc de memorie. Diferenţele specifice între scheme rezidă în modul în care tabelul este înmagazinat şi actualizat.

Tehnica de creare şi exploatare a PV-urilor utilizând memoriile cache, este potrivită pentru construcţia unei maşini ce poate tolera defecte tranzitorii.

Utilizând această structură ca şi o bază, se poate focaliza un efort suplimentar pe detectarea erorilor mai eficient decât obţinerea unor capabilităţi combinate de detectare şi corectare ale lor.

Problemele ce apar în astfel de scheme se referă la situaţiile în care procese multiple, rulând pe procesoare diferite, crează dependenţa de date între ele. Apare problema *propagării rulării înapoi* (*rollback propagation*) care constă în posibilitatea ca un procesor care rulează înapoi poate necesita rularea înapoi a altui proces şi aşa mai departe. O tehnică de evitare constă în îndepărtarea dependenţelor de date prin menţinerea unui singur PV global.

Acest PV poate fi adresat prin metode variate ce vor fi prezentate în continuare. În [Prad 96] se prezintă cazul unui SMP - MP, unde fiecare procesor are un cache, şi o magistrală comună partajată. Consistenţa cache-urilor este asigurată printr-un protocol ce reclamă existenţa a trei linii adiţionale cu următoarele funcţii:

- *partajarea unui bloc pe magistrală*;

- *stabilirea unui punct de rulare înapoi*;
- *rularea înapoi până la punctul anterior*.

O altă abordare a fost propusă în cazul SMP - MP cu memorii cache private, unde memoria partajată este ataşată procesoarelor prin intermediul unei RIN. Pentru a se împiedica propagarea rulării înapoi, se generează PV-uri ori de câte ori alte procesoare citesc o linie de cache modificată de la ultimul PV. Se utilizează termeni ca şi: *identificatori de puncte de valorificare*, *stivă de recuperare*, *numărător de lungime k*, etc.

Un avantaj semnificativ al SMP - MP îl constituie faptul că îndepărtarea unui proces defectat de la un procesor defect spre unul bun, este o operaţiune relativ uşoară. Când un procesor cade el poate fi îndepărtat din operarea întregului SMP, sarcina fiind preluată de celelalte.

Dacă defecţiunea procesorului este una tranzitorie, atunci el poate rula înapoi până la PV anterior sau până la începutul procesorului. O chestiune interesantă este aceea a procesorului pe care să se reexecute procesul, în sensul că pot exista şi alte procesoare care să efectueze această operaţie. Uzual, procesul eronat poate fi reexecutat pe acelaşi procesor şi o defecţiune repetată poate indica un defect hard şi necesitatea de a deplasa procesul la alt procesor.

În SMP - MP, această situaţie conduce la activarea stării PV a procesorului defect într-un procesor diferit. Aceasta poate fi obţinută uşor deoarece într-un SMP - MP, tot ceea ce trebuie făcut este să se încarce starea PV - ului din procesul defect într-un procesor corespunzător. Pe de altă parte, într-un SMP - TM, neexistând memorie globală partajată pentru comunicare, rularea înapoi poate constitui o procedură complexă.

4.4.1.b Recuperarea prin *rulare înapoi* în SMP - TM

Sistemele multiprocesoare cu memorie distribuită (SMP – TM) constau dintr-un set interconectat de procesoare cu memorii locale ce comunică între ele prin intermediul transferului de mesaj. Nu există o memorie globală partajată şi atunci când un procesor se defectează, informaţia este comunicată la alte procesoare din sistem într-o perioadă finită de timp după apariţia defectului.

În literatură [Prad 96], [Gros 99] etc, se descriu mai multe scheme pentru crearea şi exploatarea PV-urilor în cadrul SMP - TM.

Un SMP - TM constă dintr-un set finit de N procese secvențiale P_1, P_2, ..., P_N ce comunică şi se sincronizează doar prin mesaje. Fiecare proces execută un algoritm local şi rulează pe un procesor.

Apariția de acțiuni efectuate de algoritmi locali formează *evenimente*. Există trei tipuri de evenimente: *transmisii* (*send*), *recepții* (*receive*) şi *interne*. S-a definit următoarea relație între două evenimente: evenimentul a se produce în mod direct înaintea evenimentului b dacă şi numai dacă:

1. a şi b sunt evenimente în acelaşi proces şi a apare înainte de b; sau
2. a este transmisia mesajului m de către un proces şi b este recepționarea lui m de către alt proces.

Închiderea tranzitivă a relației *se întâmplă direct înainte* este relația *se întâmplă înainte*. Dacă evenimentul a se produce înainte de b, b se produce după a.

Diagramele de timp reprezintă o metodă convenabilă pentru reprezentarea calculului distribuit.

În fig. 4.13 este reprezentată o diagramă de timp pentru un sistem de trei procese. Liniile orizontale sunt axele de timp ale proceselor, punctele sunt evenimente iar săgețile reprezintă mesaje de la procesul transmițător la cel receptor.

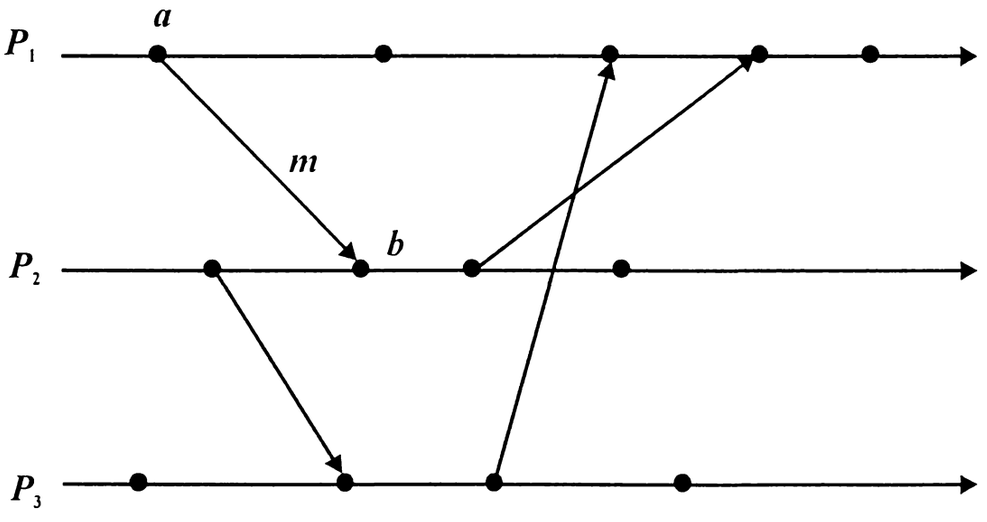


Fig. 4.13 O diagramă de timp pentru un calcul distribuit

O *stare globală* a unui SMP - TM este un set de stări locale, una pentru fiecare proces. *Starea locală* a unui proces P este definită de către starea sa inițială şi de către secvența de evenimente ce a apărut la P. Starea canalelor corespunzând unei stări globale S este setul mesajelor transmise dar încă nerecepționate în S. Se poate

reprezenta o stare globală pe o diagramă de timp printr-o tăietură în diagramă. O *stare a sistemului consistentă* (*consistent system state*) este una în care fiecare mesaj ce a fost recepționat este de asemenea prezentat ca fiind transmis în starea transmițătorului, adică evenimentul de transmisie s-a produs înainte de evenimentul recepție. Informal, o stare globală (tăietură) este consistentă dacă nici o săgeată nu punctează de la dreapta spre stânga de-a lungul tăieturii. În fig. 4.14, *C* este o tăietură *consistentă* şi *C'* este una *inconsistentă*.

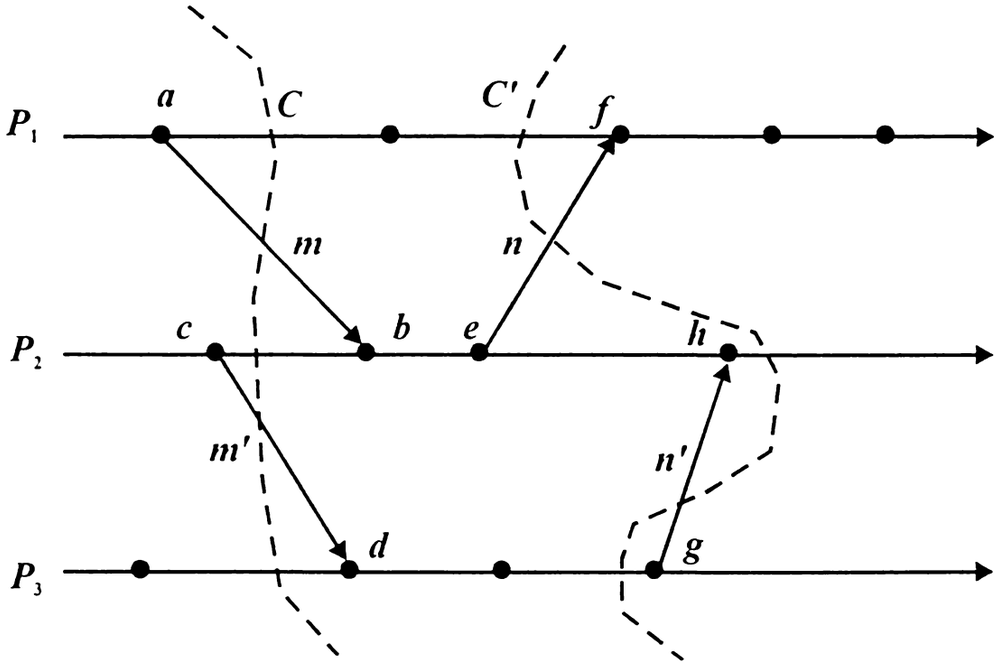


Fig. 4.14 Tăieturi consistente şi inconsistente

Un punct de verificare (PV) reprezintă starea locală salvată a unui proces. Un set de PV - uri, unul per proces, este consistent dacă stările salvate formează o stare globală consistentă [Gros 99]. Orice PV global consistent poate fi utilizat pentru restaurare după apariția unei defecțiuni. O stare globală consistentă se numeşte *linie de recuperare* (*recovery line*). În scopul de-a minimiza timpul pierdut datorită unei defecțiuni, sistemul trebuie să fie rulat înapoi spre cea mai recentă linie de recuperare. Cu ocazia recuperării pot apare următoarele patru probleme:

- *mesaje orfane* (*orphan messages*)
- *mesaje lipsă* (*missing messages*)

- *mesaje duplicat* (*duplicate messages*)
- *blocaj activ* (*livelock*)

Mesajele orfane pot apare în următoarea situație. Fie două procese P_i și P_j. P_i este restartat din punctul de verificare C_i^s și P_j din C_j^r, situație prezentată în figura 4.15. Dacă procesul P_i nu regenerează mesajul m_1 după recuperare, datorită nondeterminismului lui P_i atunci mesajul m_1, este un *mesaj orfan*. Procesul P_j depinzând de un astfel de mesaj orfan, devine un *proces orfan*. În mod normal, un mesaj m_1 trimis de P_i spre P_j este un mesaj orfan în corespondență cu perechea ordonată de puncte de verificare locale (C_i^s, C_j^r) dacă evenimentul său de recepție aparțin lui C_j^r în timp ce evenimentul de transmisie nu aparțin lui C_i^s.

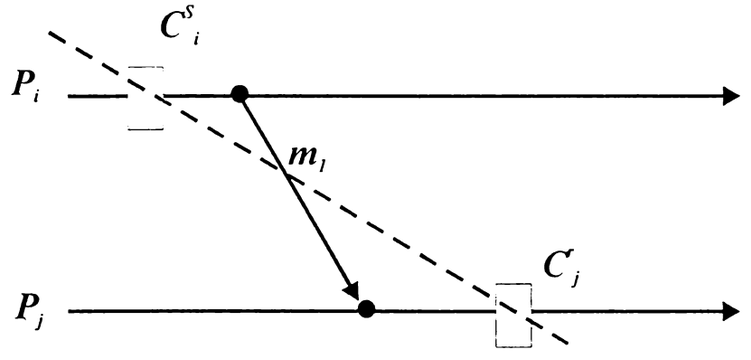


Fig. 4.15 Mesaj orfan

Pentru a preveni apariția mesajelor orfane au fost propuse câteva soluții [Prad 96], [Gros 99].

- Mesaje orfane pot fi evitate dacă procesele sunt deterministice și sunt executate pe procesoare ce se blochează la defect (fail - stop processors).
- Fiecare proces orfan trebuie să fie rulat înapoi la un PV înainte ca mesajul ce a cauzat această stare a procesului, să fie recepționat. Dacă nu se ia această precauție mai multe procese pot deveni orfane, care de asemenea trebuie rulate înapoi în timpul recuperării, producând *efectul domino-ului*.

Mesaje pierdute pot apare în următoarea situație. Fie două procese P_i și P_j. P_i este restartat din PV-ul notat C_i^s și P_j din C_j^r (fig. 4.16). Dacă procesul P_j nu execută un eveniment de recepție a mesajului m_2, acest mesaj devine pierdut. Formal, un mesaj m_2 trimis de către P_i spre P_j este denumit *pierdut* în raport cu perechea ordonată (C_i^s, C_j^r) dacă evenimentul său de transmisie aparține lui C_i^s iar recepția nu aparține lui C_j^r.

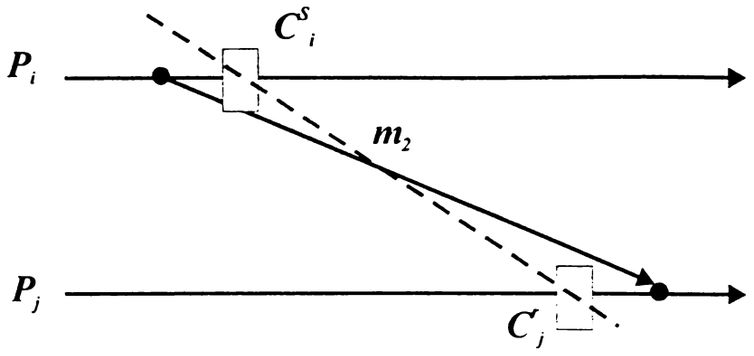


Fig. 4.16 Mesaj pierdut

Apariția mesajelor duplicat poate fi evidențiată în figura ce reprezintă cazul mesajelor orfane. Dacă evenimentul de transmisie este reexecutat în timpul calculului, mesajul m_1 este livrat de două ori spre P_j afectând apoi starea locală a lui P_j de două ori. Acest mesaj duplicat trebuie să fie manipulat cu atenție. Soluții posibile sunt: Suprimarea regenerării mesajelor duplicat, procesul receptor recunoaşte mesajul ca fiind un duplicat etc.

Blocajul activ (*livelock*) în situația în care procesul de recuperare este asincron iar procesele nu se pot recupera simultan. Această recuperare asincronă poate introduce *blocaje active* adică situații în care o singură defectare poate cauza un număr infinit de rulări înapoi, împiedicând sistemul să progreseze. Se consideră următoarea situație prezentată în fig. 4.17. Procesul P_i este rulat înapoi spre C_i^s. Această rulare înapoi poate implica rularea înapoi a lui P_j spre C_j^r datorită mesajului m_1[1].

Notația m_i^j se referă la a j - a situație a transmiterii lui m_i. Să presupunem că P_j a transmis mesajul m_2 înainte de a fi rulat înapoi. Atunci m_2 poate fi recepționat în situația de rulat înapoi a lui P_i. Apoi P_i poate rula înapoi a doua oară pentru a restaura consistența. Dacă m_1 a fost recepționat în situația de rulat înapoi a lui P_j, aceasta presupune o a doua rulare înapoi a lui P_j spre C_j^r. Prin urmare, P_i şi P_j pot fi forțate să ruleze înapoi pentru totdeauna, chiar dacă nu au apărut defecțiuni suplimentare.

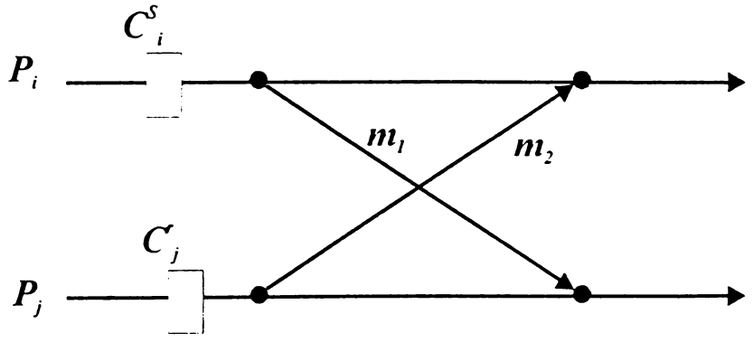


Fig.4.17 Situația blocajului activ

Se pot lua în considerare următoarele soluții pentru evitarea blocajului activ: ignorarea mesajelor de la un moment anterior al unui proces sau utilizarea unui protocol în două faze (procesele să ruleze înapoi la același moment).

Tehnicile de recuperare din defect prin rulare înapoi pentru SMP - TM pot fi clasificate în două clase principale [Gros 99]. Această taxonomie ia în considerare modalitățile prin care problemele prezentate anterior pot fi rezolvate.

Tehnicile existente de creare şi gestionare a PV-urilor pot fi clasificate în următoarele categorii:

- *gestionarea sincronă a punctelor de verificare* (*syncronous checkpointing*) - procesele sincronizează gestionarea PV-urilor de-o astfel de manieră că întotdeauna este menținut un set consistent de PV-uri în sistem;
- *gestionarea asincronă a punctelor de verificare* (*asyncronous checkpointing*) - fiecare proces îşi ia PV-urile sale în mod independent existând riscul apariției efectului domino;
- *gestionarea quasi-sincronă a punctelor de verificare* (*quasi - syncronous checkpointing*) - procesele iau atât PV-urile induse cât şi cele luate independent.

Tehnicile de recuperare din defect prin rulare înapoi, bazate pe înregistrare (*log - based rollback recovery techniqnes*), utilizează înregistrarea mesajelor suplimentar la gestionarea PV-urilor.

Aceste tehnici pot fi clasificate în trei categorii în concordanță cu modalitățile în care sunt evitate procese orfane: înregistrare pesimistă, optimistă şi cauzală.

- în *înregistrarea pesimistă* (*pessimistic logging*) crearea proceselor orfane este evitată prin efectuarea unor acțiuni corespunzătoare în timpul unei execuții;
- în *înregistrarea optimistă* (*optimistic logging*) crearea unor procese orfane este evitată prin efectuarea de acțiuni în timpul recuperării;
- *înregistrarea cauzală* (*causual logging*) combină tehnicile anterioare.

4.4.2 Recuperarea din defect prin *rulare-înainte* în sistemele multiprocesor.

Caracteristica comună la toate schemele de *recuperare prin rulare-înainte* este aceea că, după detectarea unei defecțiuni, sistemul îndepărtează starea eronată curentă şi determină starea corectă fără nici o pierdere în calcul. Există două metode diferite de abordare: una se bazează pe redundanță hardware, alta pe cea software. Abordările utilizând redundanța hardware pot fi apoi clasificate în două categorii :

- abordări utilizând redundanța statică;
- abordări utilizând redundanța dinamică.

4.4.2.a Abordări utilizând redundanța statică

Tehnicile bazate pe redundanța statică sunt foarte similare în principiu: toate utilizează acest tip de redundanță pentru mascarea defectelor.

- ***Redundanța cu mascare activă***

Se utilizează un nivel adecvat de replicare pentru a tolera defecțiunile, utilizând sisteme *TMR (triple-modular-redundant)* ce maschează un defect singular fără a pierde din performanțe.

- ***Redundanța activă utilizând module cu blocare pe defect***

Module multiple ale fiecărui procesor execută activ fiecare proces. Fiecare proces este presupus ca fiind cu *blocare pe defect (fail-stop)*. Astfel, dacă un procesor se defectează, el se opreşte şi celelalte continuă execuția task-ului fără a se produce degradări în performanțe.

În literatură [Prad 96], se citează sistemul Stratus în care fiecare subsistem este duplicat, formând o pereche iar una din replici este identificată ca fiind de rezervă. Fiecare subsistem şi rezerva sa sunt cu autotestare prin duplicare. Circuitele sunt astfel replicate de patru ori. Toate copiile hardware sunt strâns sincronizate.

Când se detectează un defect într-un subsistem prin mecanismul propriu de autotestare, se deconectează subsistemul iar rezerva sa porneşte fără nici o întrerupere sau rulare înapoi.

- ***Redundanța activă utilizând autodiagnoze***

Identificarea procesorului defect se face prin utilizarea task-urilor de autodiagnoză în locul mecanismelor de autoverificare.

Mecanismul de *duplicare reconfigurabilă* este descris în [Prad 96]. În duplicarea reconfigurărilor, procesul este replicat pe două procesoare. Se compară apoi în mod continuu ieşirile, orice nepotrivire semnalând o defecțiune la cel puțin un procesor din

pereche. După ce defectul a fost detectat, fiecare procesor rulează task-uri de auto-diagnoză pentru a localiza defectul. Odată ce procesorul defect este identificat, ieşirea procesorului liber de defect poate fi acceptată ca fiind corectă.

Utilizarea task-urilor de autodiagnoză în locul autoverificării concurente conduce la un calcul suplimentar pentru determinarea procesorului defectat.

4.4.2.b Abordări utilizând redundanța dinamică

Scheme de recuperare prin *rulare-înainte* bazate pe redundanță dinamică şi pe crearea şi gestionarea de PV-uri, prin care se încearcă rularea înapoi chiar şi în prezența defecțiunilor, au fost prezentate în [Prad 96].

Pentru a ilustra principiul de bază utilizat în recuperarea prin *rulare-înainte* se consideră un sistem duplex ce detectează defecte prin crearea şi verificarea PV-urilor din două module din sistem, în mod periodic, şi apoi comparând stările lor. Când este detectată o defecțiune, o schemă cu *rulare-înapoi* ar încerca să restabilească starea celor două module din PV-ul anterior. Schemele cu *rulare-înainte* încearcă să determine care din cele două module procesoare (PM) este liber de eroare.

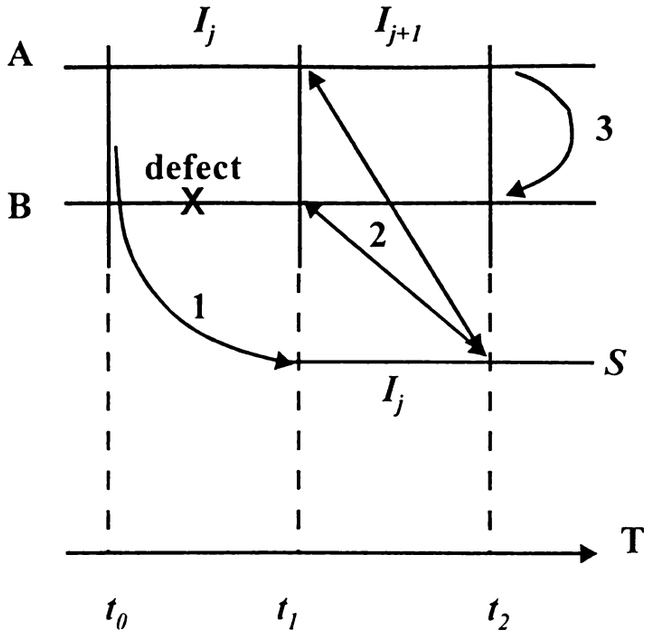
În literatură [Prad96], [LFA90], [LFA91], [PV94a], [PV94] se prezintă mai multe *scheme de creare şi gestionare a punctelor de verificare prin rulare-înainte* (*Roll-Forward Checkpointing Scheme*) - RFCS.

O schemă RFCS este un mecanism pentru identificarea modulului de procesare defect în cadrul unui sistem duplex prin *relansarea executării programului* (*computation retrying*) pe un *modul partener de rezervă* (*spare module*).

Odată ce PM defect este identificat, starea acestuia este făcută consistentă în raport cu starea PM-ului liber de eroare din cadrul sistemului duplex.

Calculul se reîncearcă în modulul de rezervă, concurent cu execuția în continuare a task-ului pe cele două module din cadrul sistemului duplex.

În fig. 4.18 se prezintă executarea a două copii ale unui task, denumite A şi B. În situația în care B defectează în intervalul de checkpointare I_j se produc următoarele acțiuni. Semnăturile lui A şi B se vor compara la sfârşitul intervalului I_j la timpul t_1 Această comparare la t_1 va activa relansarea în manieră concurentă a intervalului de verificare I_j, pe modulul de rezervă, după cum urmează. În timpul acestei relansări, atât A cât şi B continuă să avanseze spre următorul interval de checkpoint I_{j+1}.



1 : Starea de copiere în modulul de rezervă

2 : Starea de comparare a modulului de rezervă cu starea lui A şi B

3 : Starea de copiere din A şi B

Fig. 4.18 Relansarea concurentă în cadrul unei scheme RFCS

La începutul relansării concurente, PV-ul anterior (prelevat la t_0) este încărcat într-un modul de rezervă. Intervalul de verificare I_j, în care a apărut defectul, este relansat în modulul de rezervă. În mod concurent, A şi B continuă execuția următorului interval I_{j+1}. Cum A şi B nu au fost în aceeaşi stare la t_1, este puțin probabil că vor fi în aceeaşi stare la sfârşitul lui I_{j+1}, care a fost prelevată la t_2.

După ce modulul S completează intervalul I_j la timpul t_2, starea lui S este comparată cu starea lui A şi B la t_1.

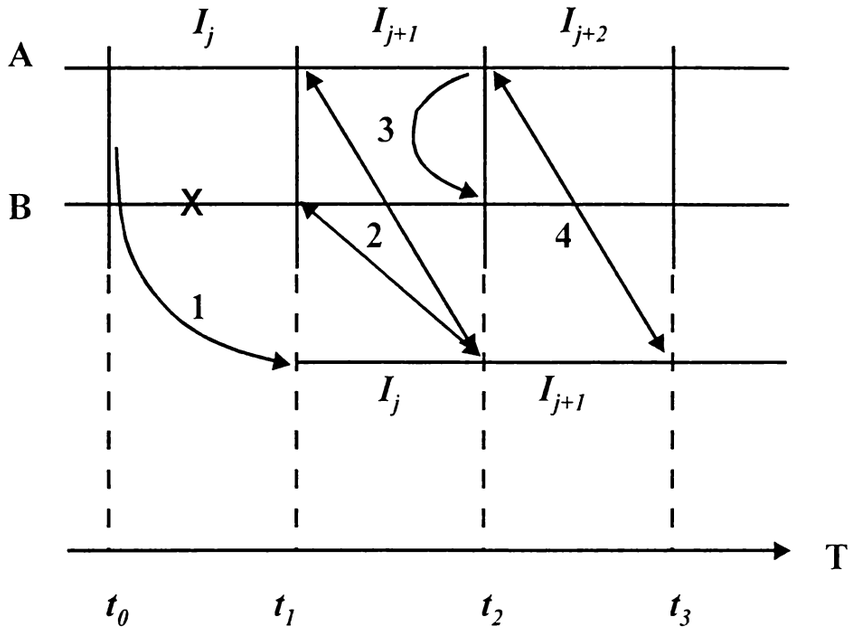
Dacă A şi S nu se defectează în I_j, atunci starea lui S la t_2 se va potrivi cu starea lui A la t_1. Când apare această similitudine, s-a presupus că A este liber de eroare la t_1.

Acum, starea lui B este făcută identică cu starea lui A la t_2. Astfel, dacă A nu defectează în intervalul I_{j+1}, atunci A şi B vor fi ambele în starea corectă la momentul t_2.

Pentru a determina dacă A a defectat în I_{j+1}, modulul de rezervă trece la executarea intervalului I_{j+1}. Acest caz este prezentat în fig. 4.19

După ce modulul de rezervă parcurge acest interval, starea sa este comparată cu starea lui A la sfârşitul lui I_{j+1}.

Dacă cele două stări se potrivesc, atunci se presupune că a a fost liber de defect în I_{j+1}; astfel sistemul duplex este rulat înapoi. O analiză detailată a funcționării schemelor RFCS este prezentată în [Prad 96]. În toate schemele RFCS se presupune că există înglobate capabilități de detectare a defectului în modulele procesoare.



4. Starea de comparare a modulului A şi a celui de rezervă

Fig. 4.19 Tehnica de relansare concurentă în schemele RFCS.

Schemele RFCS evită rularea înapoi chiar şi în prezența unui singur defect. Defectul este astfel tolerat fără o penalizare în performanță.

Relativ recent (1994) s-au produs variante ale schemei clasice RFCS. Presupunând că fiecare modul are un anumit număr de capabilități de detectare a defectului conținute în module - verificări la paritate, capabilități de detectare a situațiilor de excepție , etc. - se pot conceptualiza patru scenarii diferite, prezentate în TABEL VIII.

TABEL VIII. Patru scheme diferite cu *rulare înainte*

Strategie de recuperare	**Resurse utilizate**	
	Cu modul de rezervă	**Fără modul de rezervă**
Optimistă	*Rulare înainte* I	*Rulare înapoi* I
Pesimistă	*Rulare înainte* II	*Rulare înapoi* II

În cadrul unei strategii de recuperare *optimistă*, se acordă încredere în cel mai mare grad capabilităților de detectare conținute în module.

De exemplu se consideră că un defect a fost detectat în modulul B de către un astfel de mecanism intern, situație prezentată în fig. 4.20.

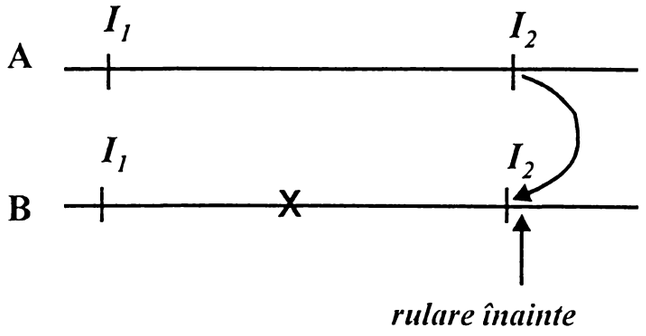


Fig. 4.20 Schemă optimistă cu sau fără modulul de rezervă

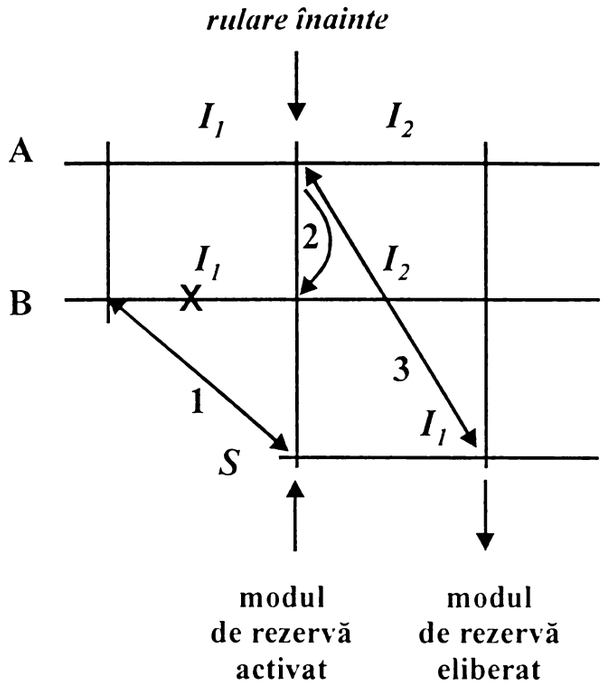
La sfârşitul intervalului de checkpointare I_1, starea lui A poate fi copiată în B şi procesul ar putea rula înainte. Aceasta nu va solicita utilizarea unui modul de rezervă, chiar dacă el este disponibil. Această schemă este denumită *Rulare - înainte I.*

În schema *pesimistă*, modulul de rezervă este activat în mod automat şi se execută relansarea concurentă şi recuperarea ca şi în fig. 4.21 a. Această schemă este denumită *Rulare - înainte II.* Se observă că deoarece modulul B a fost tot timpul suspectat ca fiind defect, doar în situația în care A a prezentat o defecțiune ce a scăpat capabilității de detectare conținută în modul (a lui A în intervalul I_1) s-a acționat de-o manieră mai conservatoare. În acest caz, s-ar putea ca A să se fi defectat în intervalul I_1 (ceea ce ar corespunde la o dublă eroare în timpul lui I_1) şi procesul să ruleze înapoi când modulul de rezervă nu are aceeaşi stare cu a după reexecutarea lui I_1.

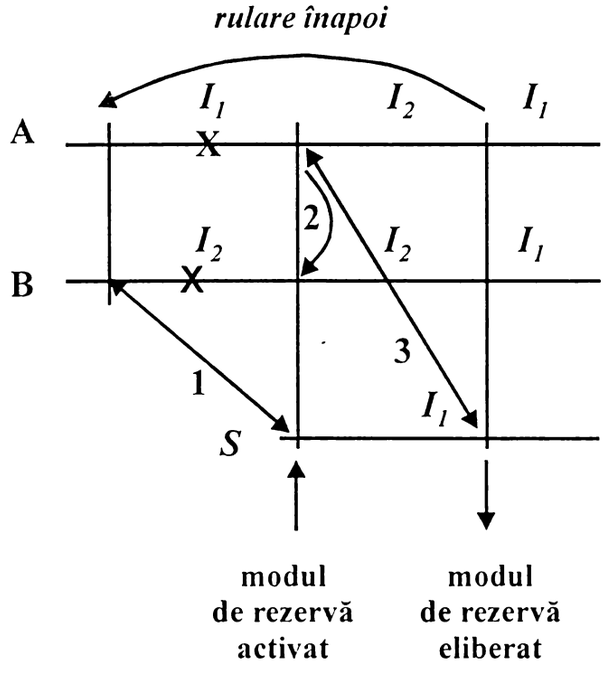
Oricum, în ipoteza că nu sunt disponibile module de rezervă, schema *optimistă* (*Rulare-înapoi I*) va rula înainte, ca şi în fig. 4.20, în timp ce schema *pesimistă* (*Rulare înapoi II*) va rula înapoi atât A cât şi B pentru a reexecuta I_1 , ca şi în cazul schemelor standard cu rulare înapoi.

O analiză prezentată de Pradhan şi Vaidya în [Prad 96] arată că un câştig semnificativ în performanță poate fi obținut cu un sacrificiu de talie redusă în ceea ce priveşte siguranța în funcționare.

În fig. 4.22 se prezintă comparativ trei sisteme în spațiul siguranță în funcționare - performanță. Se observă că este convenabil un sistem ale cărui caracteristici sunt descrise de curba 1. Acest sistem care permite o mică reducere în siguranța în funcționare va avea însă un semnificativ câştig în performanță. Schemele optimiste descrise oferă potențial pentru scheme ca şi cele descrise în fig. 4.23 şi fig. 4.24.



a. Schemă pesimistă cu modulul de rezervă rulând înainte, cu toate defectele singulare



b. Schemă pesimistă cu modulul de rezervă rulând înapoi, cu defecte duble

Fig. 4.21 Scheme pesimiste.

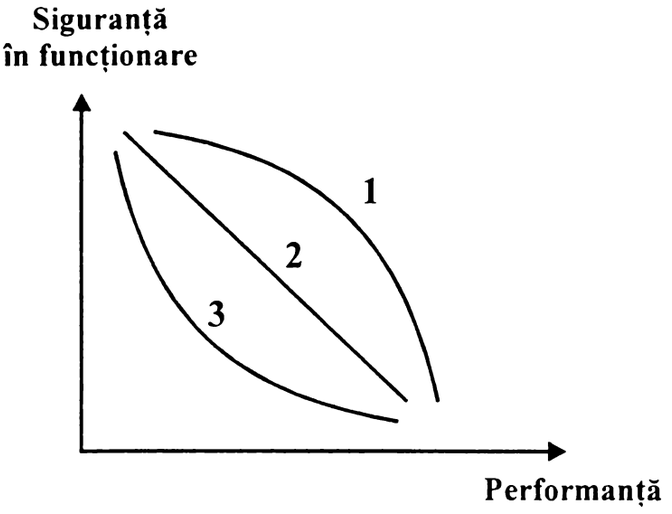


Fig. 4.22 Trei scheme diferite cu *rulare înainte*

În fig. 4.24 se ilustrează o comparație tipică de performanță indicată de o formulare analitică. Aici $\lambda = 10^{-6}$ reprezintă rata asumată de defectare şi $c = 0.8$ este gradul de acoperire al detectării defectului. În general, timpul mediu de tratare şi de reuperare din eroare când a apărut o defecțiune, este mai mic pentru schemele cu *rulare-înainte* în ambele scenarii : optimistă şi pesimistă. Acest parametru reprezintă o metrică importantă deoarece fără nici o defecțiune, toate sistemele operează similar. Prin urmare este important să se cunoască comportamentul sistemului după ce acesta a experimentat o situație în care a fost nevoit să se recupereze din defect.

În ceea ce priveşte siguranța în funcționare, se pot accepta valori diferite pentru capabilitățile de detecție conținute în module. Parametrul c poate lua valori de la 0 - nu există nici o capabilitate de detecție - până la 1 - capabilitate perfectă de detecție. Se observă că atunci când $c = 0$, *schemele Rulează - înainte I = Rulează - înainte II* şi *Rulează - înapoi I = Rulează - înapoi II* din punct de vedere al nesiguranței în funcționare. Cu alte cuvinte când nu există capabilități de detecție înglobate, schema optimistă şi corespondenta sa pesimistă au siguranțe în funcționare identice. Deoarece nu există detectare încapsulată nu există nici o modalitate de-a identifica modulul defect fără a efectua compararea între module.

Pe măsură ce c creşte apare o diferență între ambele abordări ca şi în fig. 4.24. Aici rata de defectare este presupusă ca fiind 10^{-3} şi $n = 10$ reprezintă intervalul de creare şi gestionare a PV-urilor. Schema pesimistă suferă mai puțin de degradarea siguranței în funcționare de cât cea optimistă.

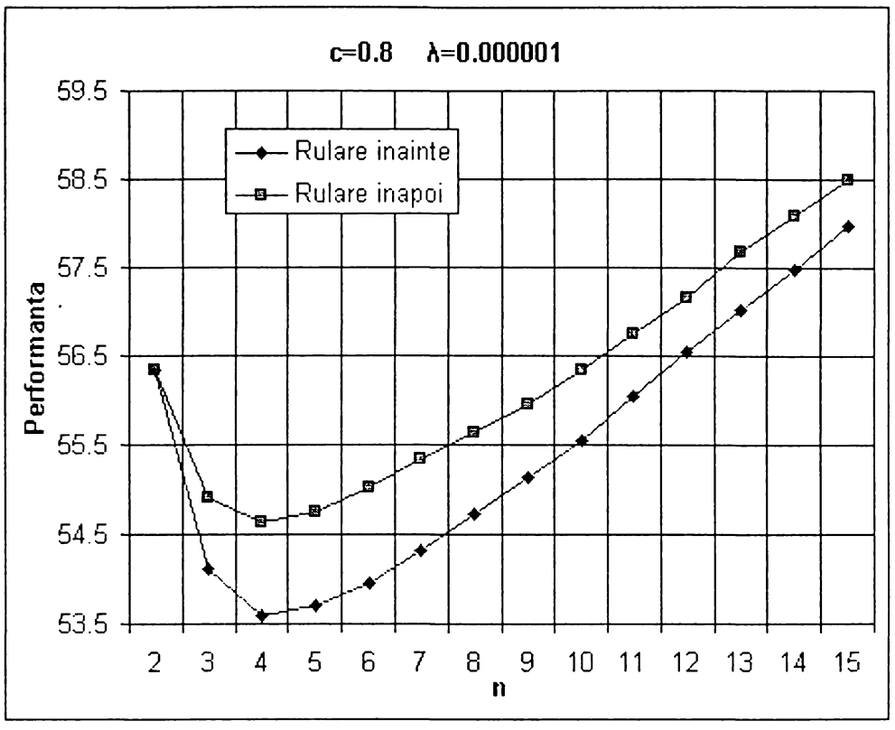


Fig. 4.23 Comparația în perfomanță dintre schemele optimiste şi pesimiste
Timpul mediu de completare în prezența unui defect

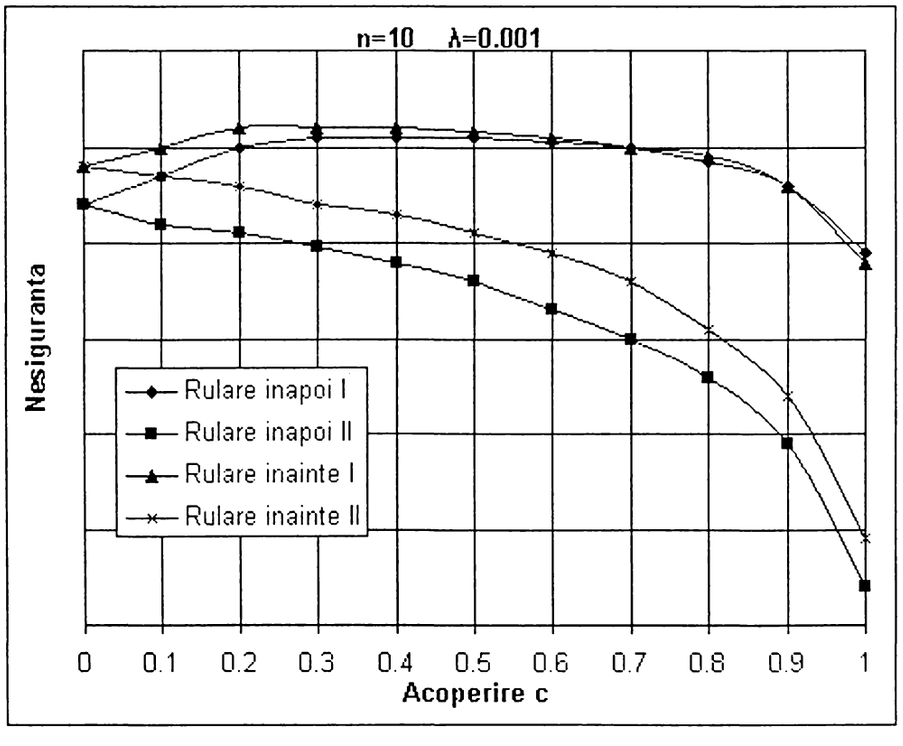
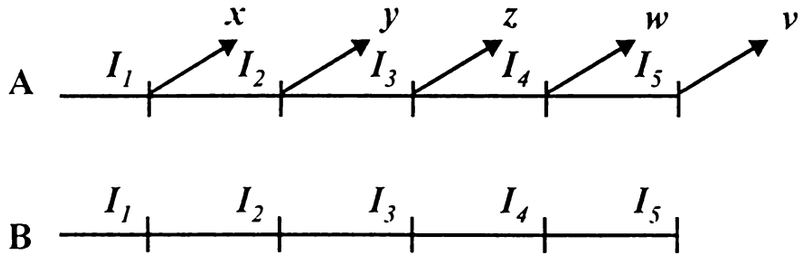


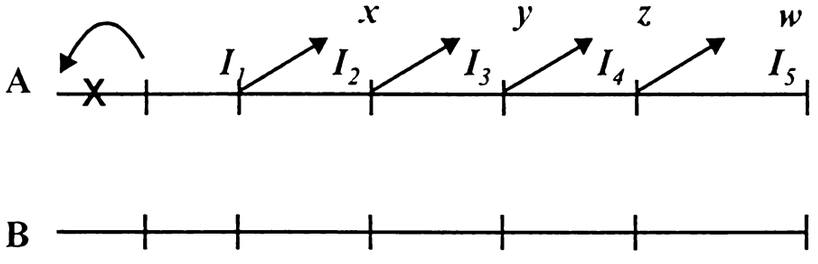
Fig. 4.24 Comparații ale siguranței în funcționare între schemele optimiste şi pesimiste

Dar, în realitate, diferența este mică în termeni de nesiguranță în funcționare. Presupunând însă că se poate tolera această diferență, se preferă schema optimistă deoarece are performanțe superioare în situația apariției unui defect. Pe de altă parte, dacă siguranța în funcționare este pe primul loc atunci se pot alege scheme pesimiste care sunt mai conservatoare. Trebuie notat că efectul utilizării modulelor de rezervă poate să reducă siguranța în funcționare, dată fiind o strategie fixată de recuperare, dar pot ameliora performanțele.

În fig. 4.25 se prezintă comportamentul schemelor cu *rulare - înapoi* în cazul apariției unui defect.



a. În absența defectului



b. Pierderea unui interval când apare o eroare la toate ieşirile

Fig. 4.25 Întârzierea permanentă la ieşirile schemelor cu *rulare-înapoi* în cazul apariției unui defect.

Unul dintre avantajele importante ale schemelor cu *rulare - înainte* este degradarea minimă în performanțele de I/E. Se consideră schema standard cu *rulare – înapoi*. În mod normal, ieşirile se vor poziționa la sfârşitul unui interval după ce o comparație indică că nu a apărut nici o eroare în timpul intervalului. Se consideră un proces care are ieşirea x la sfîrşitul lui I_1, y la sfârşitul lui I_2, z la sfârşitul lui I_3 şi w la sfârşitul lui I_4.

Dacă apare un defect în I_1, intervalul I_2 va fi utilizat pentru reexecutarea lui I_1 . Astfel, toate ieşirile după I_1 vor experimenta o întârziere de un interval de checkpointare, caz prezentat în fig.4.26.

În cazul schemei cu *rulare - înainte*, în cazul aceluiaşi scenariu, ieşirile vor fi întârziate temporar ca şi în fig. 4.25. Ieşirile *x* şi *y* sunt singurele întârziate şi toate celelalte ieşiri vor apare la intervalul de timp planificat, în mod regulat. Mai exact, ieşirea *x* va fi întârziată două intervale, în timp ce *y* doar unul. Toate ieşirile subsecvente se vor poziţiona fără nici o întârziere, dacă nu există defect în sistem.

De exmplu ieşirile *z*, *w* şi *v* vor apare la sfârşitul lui I_3, I_4 şi I_5 ca şi în fig. 4.26.

Acesta este acelaşi timp ce se aşteaptă să apară când nu sunt defecte, ca şi în fig. 4.25 a.

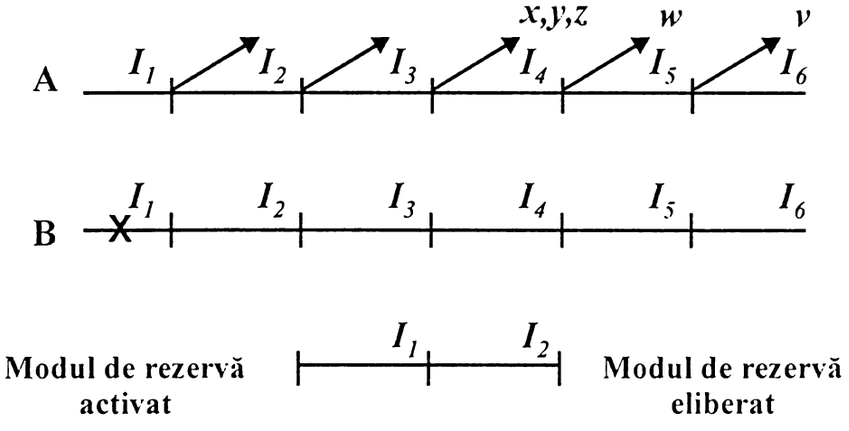


Fig. 4.26 Întârziere temporară în ieşirile schemelor cu *rulare - înainte* în situaţia unui defect

4.5 RECONFIGURAREA ÎN SISTEMELE MULTIPROCESOR

Sistemele multiprocesor tolerante la defect trebuie să aibă capacitatea de *reconfigurare*.

Într-o primă abordare, scopul este obținerea unui nivel controlabil de performanță degradată după ce defectele au fost identificate şi izolate în tendința lor de propagare. Aceasta implică posibilitatea de a identifica un subset de procesoare active, conectate în aceeaşi topologie ca şi cea originală, dar de o talie mai redusă. De exemplu, dacă topologia originală fără defecte este o *plasă* $N \times N$, după apariția defectului se încearcă identificarea unui set de $M \times M$ procesoare, unde $M<N$. În astfel de scheme, aplicațiile de calcul paralel care rulează pe aceste multiprocesoare utilizează proprietatea topologiei multiprocesorului şi pot evolua pe orice măsură a topologiei. Cea mai simplă abordare a reconfigurării software în *hipercuburi*, identifică subcubul liber de erori de cea mai mare talie şi rulează aplicațiile pe el. Dificultatea în această abordare este că degradarea în performanță poate fi până la 100% pentru primul defect în *hipercub* deoarece se va utiliza un subcub la jumătate din dimensiunea inițială.

A doua abordare în scopul obținerii toleranței la defect în RIN statice este aceea prin care interconexiunea este privită ca şi o *cale* de a conecta un mare număr de procesoare ce pot comunica prin transmiterea de mesaje de-a lungul legăturilor. Intr-o astfel de situație, toleranța la defect poate fi uşor obținută printr-o *degradare grațioasă* (*graceful degradation*), prin *redistribuirea sarcinii de calcul* (*computational load*) în jurul procesoarelor defecte şi *rerutarea* mesajelor. In această ipostază este necesară investigarea algoritmilor de rutare ce vor transmite mesaje ce trebuie să fie rutare spre procesoarele active, evitând procesoarele defecte. Această abordare este adecvată pentru scenariile unde aplicațiile ce se execută pe aceste multiprocesoare nu necesită o topologie specifică de interconectare dar reclamă ca procesoarele să fie conectate.

A treia abordare necesită proiectarea unei *topologii augmentate* (*augmented topology*) prin adăugarea de procesoare suplimentare şi conexiuni adiționale structurii originale, astfel că atunci când apar defecte în procesoare sau în conexiuni, topologia originală să fie identificată. In această abordare, reconfigurarea este obținută prin substituirea elementelor defecte cu cele suplimentare pentru a se menține nivelul de performanță nemodificat.

In continuare se vor detalia aceste abordări pentru mai multe topologii de multiprocesoare. Se va accentua asupra reconfigurării rețelelor de interconectare, acestea având un rol major în crearea statutului de sistem tolerant la defect.

4.5.1 Reconfigurarea în SMP – MP bazate pe *magistrală*

Magistrala este cea mai comună RIN în care toate procesoarele şi toate modulele de memorie sunt conectate la o resursă comună. SMP – MP bazate pe magistrală sunt cel mai vulnerabile la defecte.

SMP – MP bazate pe magistrală, tolerante la defect, au avantajul că perechile de module procesor şi module de memorie pot fi extrase usor din sistem atunci când au fost identificate ca fiind defecte. Defectele sunt gestionate prin utilizarea *magistralelor redundante*. Într-un sistem redundant toate procesoarele şi toate memoriile sunt conectate la un număr de magistrale ca şi în fig.3.4. Funcționarea SMP – MP cu *magistrală multiplă* a fost descrisă *în extenso* în § 3.1.3.

Când o magistrală se defectează, sistemul operează ca şi un sistem cu B–1 magistrale, din cele B existente. Arbitrii trebuie să fie reconfigurați pentru a opera cu B–1 magistrale.

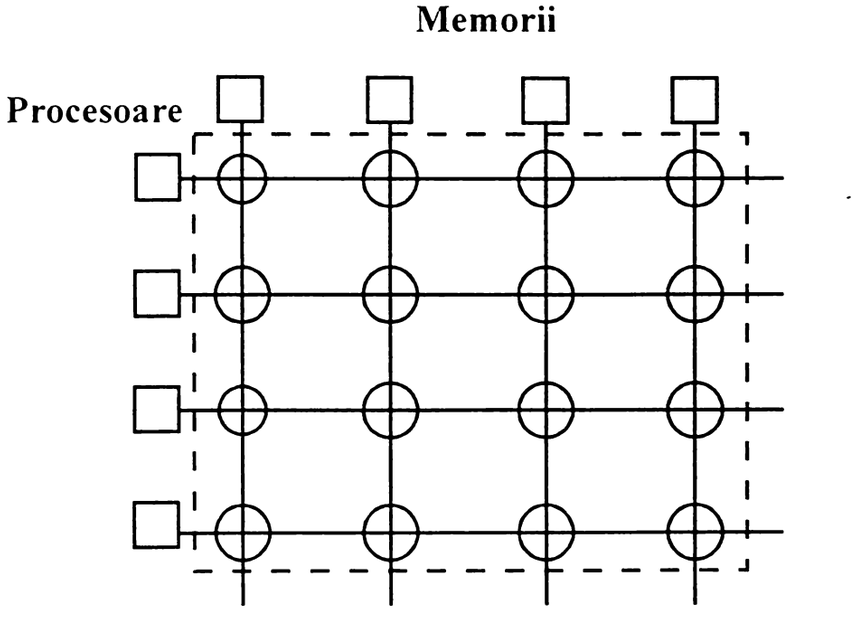
În literatură [Prad 96] se prezintă mai multe metode de reconfigurare a acestor sisteme. În principiu toate permit interconectarea a N procesoare utilizând un numar fixat de procesoare / magistrală, cu fiecare procesor conectat la două magistrale. Aceste topologii permit un anumit grad de toleranță la defect.

4.5.2 Reconfigurarea rețelelor de tip *crossbar*

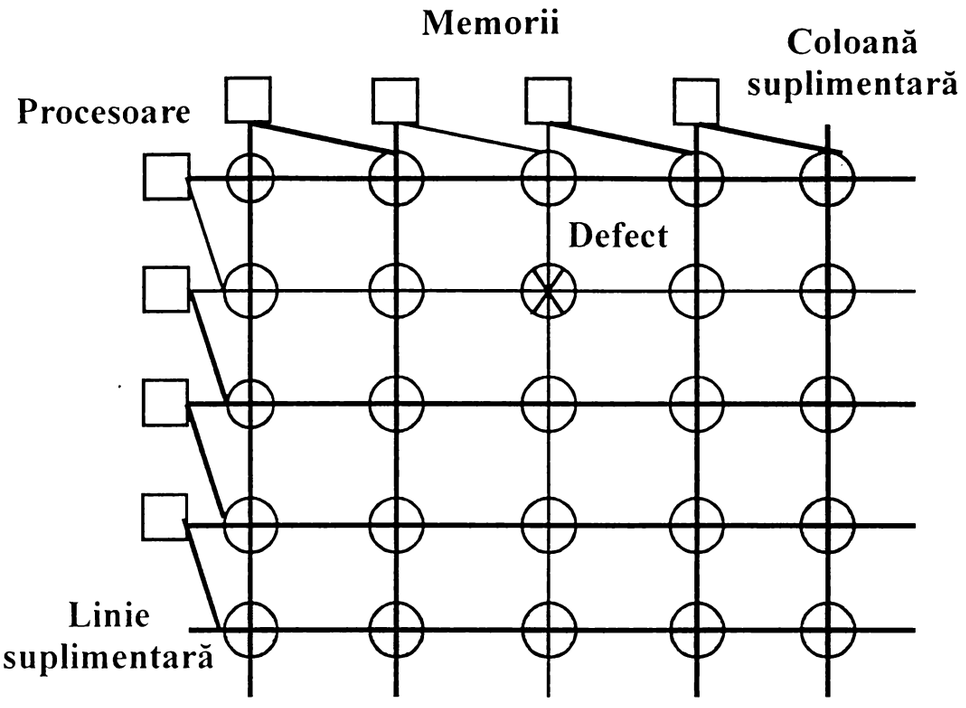
RIN *crossbar* a fost prezentată în §2.4.4. Reamintim că o altfel de conexiune se caracterizează prin prezența a N^2 comutatoare ce conectează N surse cu N destinații. Este permisă conectarea simultană între toate perechile procesor-memorie. Logica de arbitrare este încorporată în comutator. Un mod echivalent de-a construi un *crossbar* $N \times N$ este acela prin care este privit ca şi o colecție de N multiplexoare $N \times 1$, cu cele N intrări legate la fiecare din multiplexoare. Fiecare din cele $N \times 1$ MUX-uri poate fi implementat ca şi un arbore de 2×1 MUX-uri având $N - 1$ astfel de comutatoare.

O metodă de-a obține o mai mare fiabilitate la astfel de rețele este *replicarea* lor în întregime, adică păstrarea a R copii a fiecărei din cele N MUX-uri $N \times 1$.

O altă metodă utilizează *o linie* şi *o coloană* suplimentare astfel că se utilizează un *crossbar* $(N + 1) \times (N + 1)$ cu fiecare port conectat la două linii sau coloane de comutatoare. La defectarea unui comutator, se deconectează întrega linie şi coloană de comutatoare şi se activează coloana şi linia suplimentară. Schema este prezentată în fig. 4.27 cu linie îngroşată fiind prezentată reconfigurarea *crossbar-ului* în caz de defect a unui comutator.



a) RIN *crossbar* inițial



b) RIN *crossbar* redundant

Fig. 4.27 RIN *crossbar* tolerantă la defect.

4.5.3 Reconfigurarea rețelelor *hipercub*

Multicalculatoarele de tip *hipercub*, apărute relativ recent, oferă o abordare spre calculatoarele masiv paralele, relativ ieftine şi fezabile. Ele conectează un număr mare de procesoare $P = 2^d$, ce au memorie locală, utilizând conexiuni directe în conformitate cu topologia de interconectare de tip *d-cub* (*binary d-cube*).

În literatură [Prad 96], [Ban90] se prezintă mai multe soluții de rezolvare a problemei toleranței la defect pentru acest tip de multicalculatoare.

O soluție constă în reconfigurarea *hipercubului d*-dimensional prin augmentarea cu un port adițional în dimensiunea $d + 1$. Extra portul este utilizat pentru a conecta un *procesor de extensie* (*spare processor*). Soluția este prezentată în fig. 4.28a. Procesorul de extensie se poate conecta la portul adițional al fiecărui procesor din *hipercub*, astfel că atunci când un procesor se defectează, se poate trece peste acesta activându-se conexiunile corespunzătoare spre vecinii procesului defect din topologie. Această aparent simplă abordare implică un grad $N = 2^d$ pentru procesorul de extensie.

Soluția problemei constă în utilizarea unui set de comutatoare *crossbar* într-o încapsulare VLSI ce implementează această conectare. Se consideră un *hipercub* cu 16 procesoare astfel că fiecare procesor să aibă în mod normal patru porturi.

Se adaugă un extra port pentru asigurarea toleranței la defect. Cele 16 procesoare sunt divizate în două grupe de câte 8 procesoare. Cele opt conexiuni ce ies din fiecare procesor al subcubului sunt conectate la un comutator *crossbar* 8×5. Fiecare subcub de 8 procesoare are un comutator *crossbar* corespunzător (în acest caz, două comutatoare). Ieşirile celor două comutatoare *crossbar* sunt conectate împreună cu cele cinci porturi ale procesorului de extensie. Când un procesor dintr-un subcub se defectează (de ex. procesorul 4), procesorul de extensie activează cele patru conexiuni ale subcubului defect prin comutatorul *crossbar* corespunzător (de ex. procesoarele 0, 5, 6) şi o conexiune a altor comutatoare *crossbar* (de ex. procesor 12) în alte subcuburi la care procesorul defect a fost conectat, ca şi în fig. 4.28b.

Schema prezentată poate fi extinsă pentru utilizarea mai multor procesoare de extensie unde un astfel de procesor a fost afectat la un subcub de procesoare, în loc de întreg *hipercubul*.

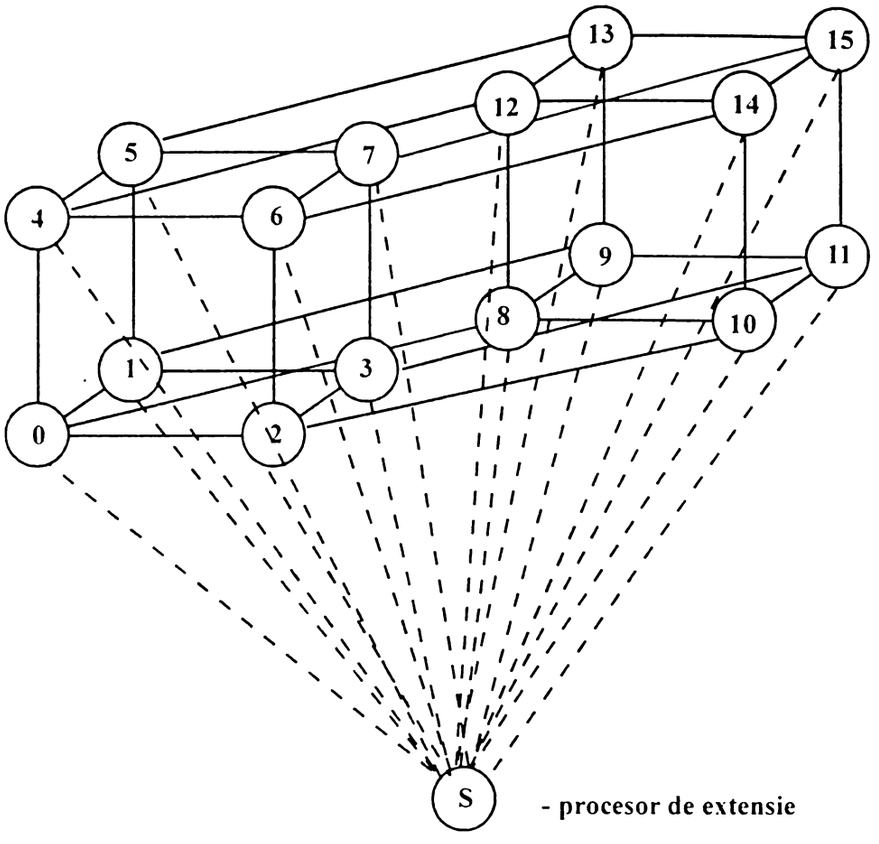
Alte metode hardware pentru reconfigurarea unui *hipercub* sunt descrise în [Prad 96].

În prima schemă hardware pentru reconfigurare este presupus că procesoarele de extensie pot fi ataşate la procesoare specifice ale *hipercubului*. În această abordare, două tipuri de noduri (noduri P şi noduri S) sunt proiectate în *hipercub*. În fig. 4.29a se prezintă un *hipercub* cu 16 procesoare unde patru din noduri (0, 7, 9 şi 14) sunt *noduri* S, restul fiind *noduri* P. Un nod P constă dintr-un procesor de calcul CPU (computation processor) conectat printr-o magistrală internă la o memorie locală, şi o logică de rutare a mesajului constând dintr-o unitate DMA şi un comutator *crossbar* $(d + 1) \times (d + 1)$ pentru *hipercubul* cu 2^d procesoare. Un nod S constă din două copii ale CPU şi ale memoriei locale conectate la două magistrale interne. Unitatea DMA şi logica de rutare a mesajului sunt partajate între cele două unități de procesare, dintre care una este activă în condiții normale iar alta este de extensie. În caz de defectare a oricărui element de procesare (CPU sau memoria locală), fie prin nodul S sau prin nodul P, cuplul procesor de extensie - memorie de la nodul S corespunzător este conectat la linie. Canalele de rutare şi logica de rutare a mesajului ce aparțin nodului S sunt partajate între procesoarele active şi cele de extensie prin intermediul magistralelor interne.

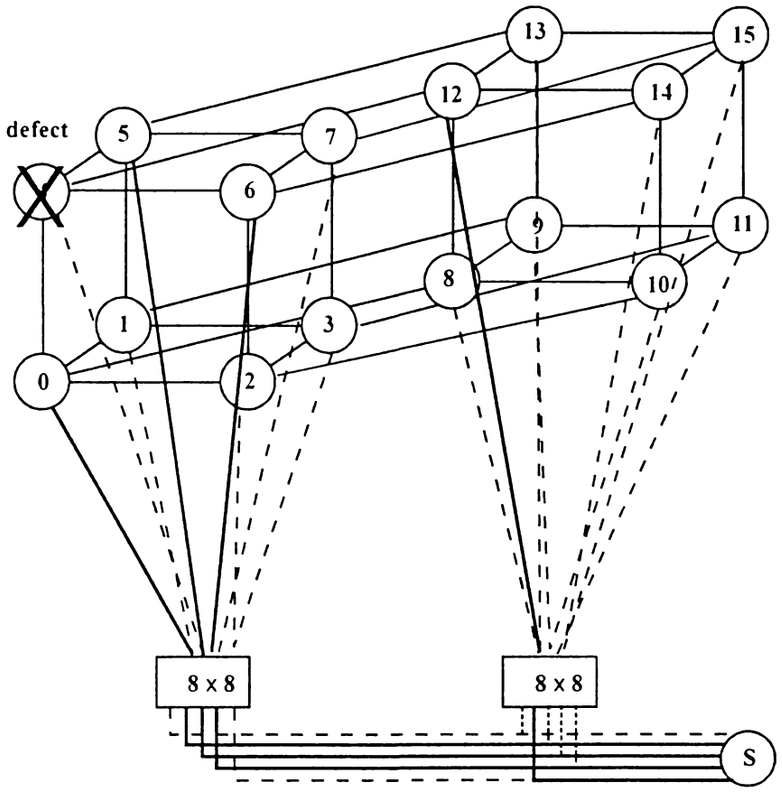
O chestiune importantă o reprezintă modalitatea optimă de alocare a procesoarelor de extensie la un *hipercub*. Evident, alocând un procesor de extensie pentru fiecare procesor, fiecare nod din *hipercub* devine un nod S. Dacă un procesor de extensie este asignat unui set de procesoare active, atunci costul hardware-ului adițional poate fi minimizat.

Trebuie rezolvată problema mapării nodurilor S în topologia *hipercubului* astfel încât fiecare procesor din sistem să fie adiacent la cel puțin un procesor de extensie. S-au dezvoltat algoritmi adecvați pentru rezolvarea acestei probleme. Fig. 4.29a arată un exemplu de alocare a patru procesoare de extensie (noduri S) la un *hipercub* de 16 procesoare.

Fiind dat un set de defecte multiple într-un astfel de sistem, s-a dezvoltat o strategie de reconfigurare bazată pe un model de graf bipartit . Nodurile acestui graf corespund nodurilor defecte şi nodurilor de extensie disponibile, iar muchiile dintre ele măsoară distanța dintre noduri ce trebuie să fie traversate între ele. Algoritmul de reconfigurare oferă o mapare optimă a nodurilor defecte pe nodurile disponibile prin minimizarea dilatării totale a conexiunilor logice de mapare de-a lungul multiplelor conexiuni fizice. Figura 4.29b prezintă un *hipercub* cu patru defecte localizate la nodurile 0, 5, 6, 15. Figura 4.29c prezintă reconfigurarea unui *hipercub* cu 16 procesoare supus la patru defecte. Defectul în nodul 0 este înlocuit printr-un nod de extensie 4, defectul din nodul 3 este înlocuit prin nodul 12, defectul din nodul 6 prin nodul 2 şi defectul din nodul 15 este înlocuit prin nodul de extensie 10.

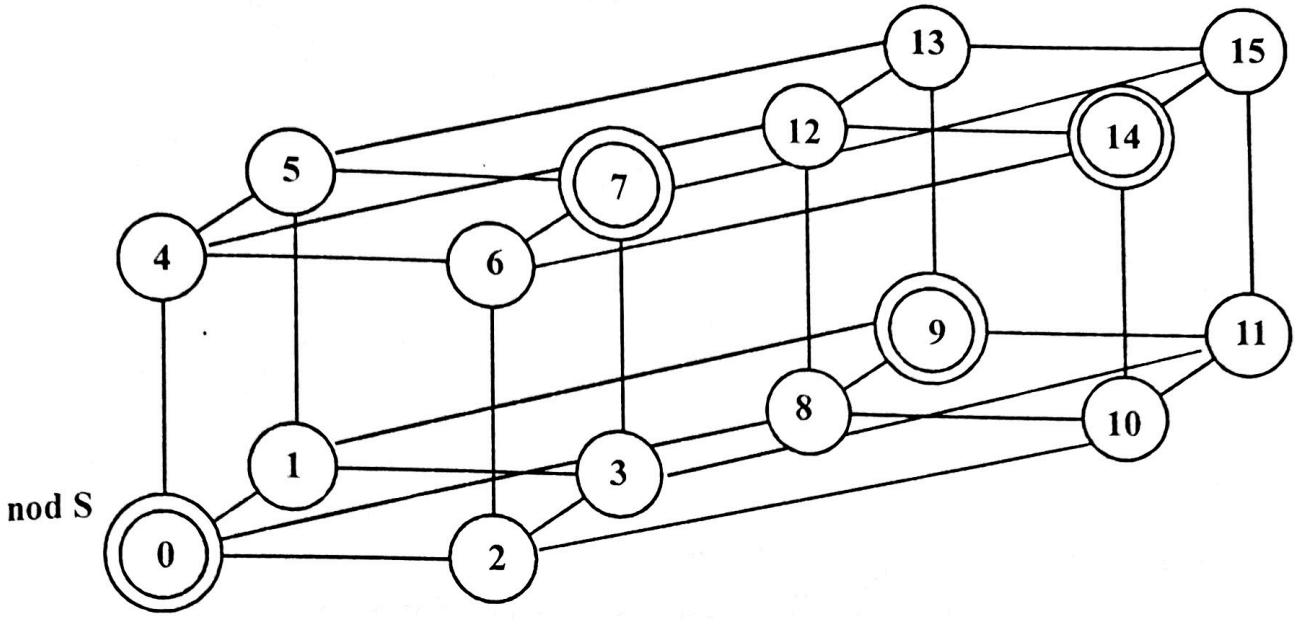


a. Conceptul de adăugare a unui procesor de extensie prin intermediul extra portului

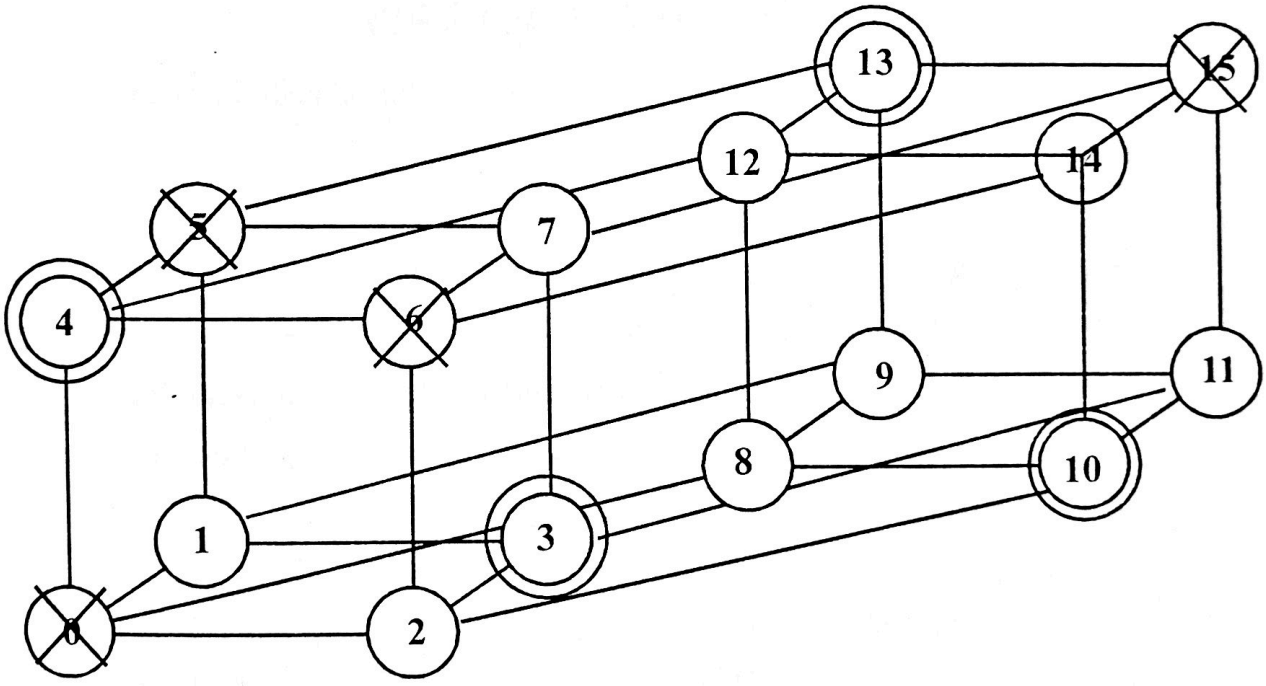


b. Implementarea unui extraport utilizând comutatoare *crossbar*

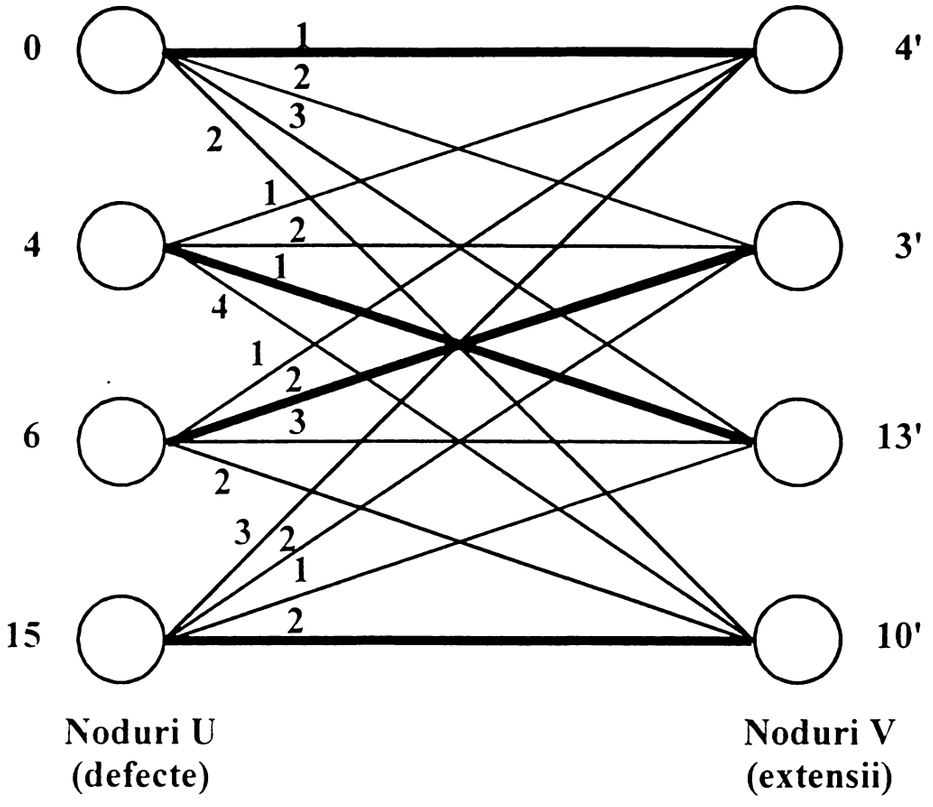
Fig. 4.28 Reconfigurarea de *hipercub* prin extraporturi



a. Alocarea de noduri de extensie



b. Localizarea a patru defecte în *hipercub*



c. Grafic de reconfigurare

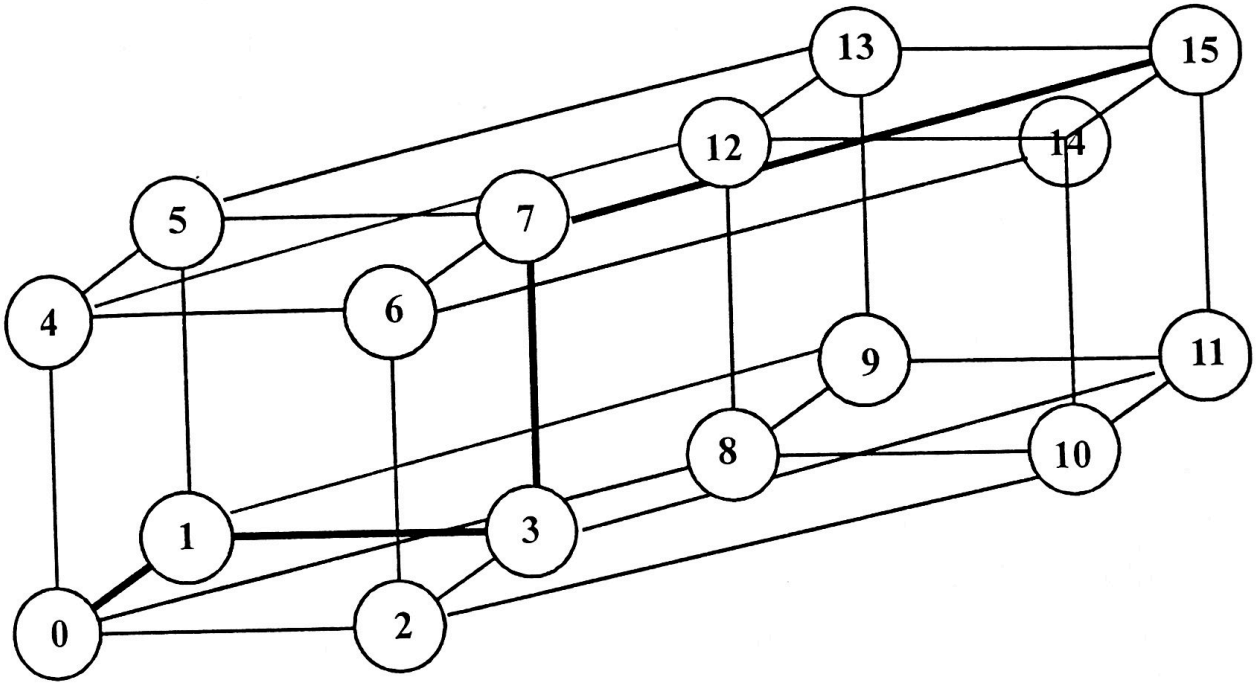
Fig. 4.29 Reconfigurare a *hipercubului* utilizând noduri de extensie

Rutarea tolerantă la defect în rețele hipercub.

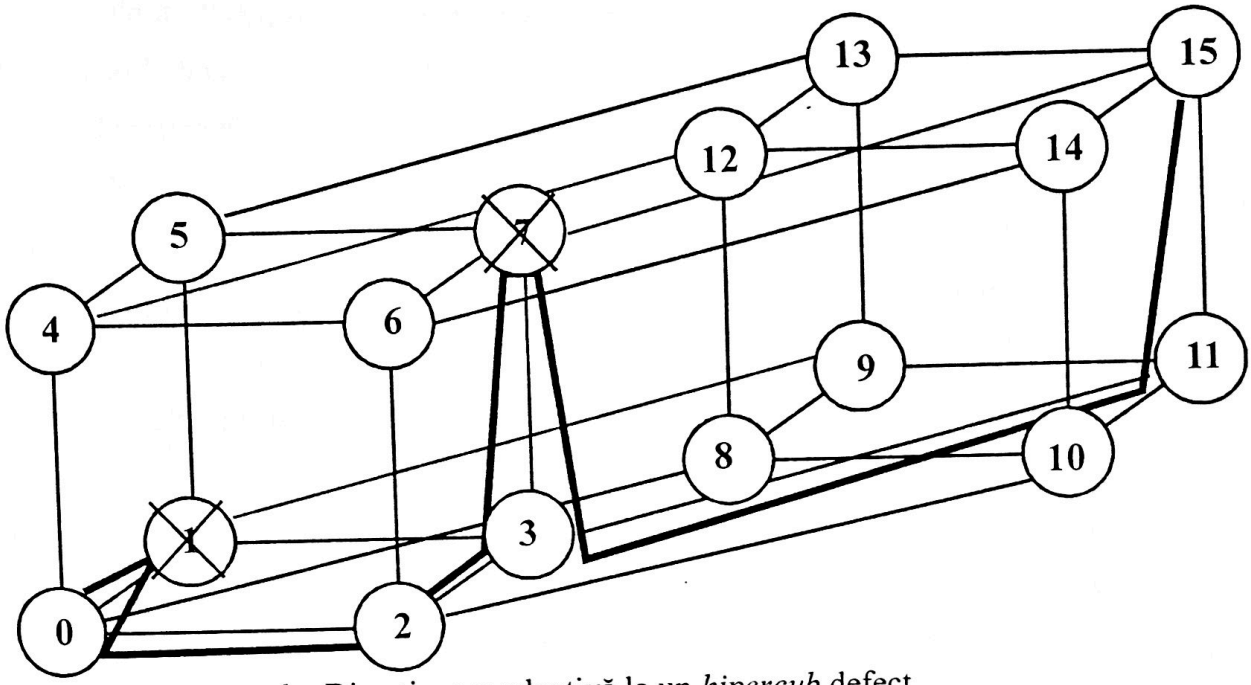
Procedura de rutare fixă standard se numeşte *rutarea e-cub*. În algoritmul *e-cub* diferențele în bit dintre sursa şi destinația unei căi se rezolvă de la dimensiuni reduse la dimensiuni mai mari. Figura 4.30a prezintă o cale de rutare *e-cub* într-un *hipercub* de 16 procesoare. În caz de defecțiune în *hipercub*, mesajele trebuie să fie rerutate utilizînd o cale ca cea prezentată în fig. 4.30b.

Există şi *algoritmi de rutare adaptivi*. Într-un astfel de algoritm, fiecare mesaj poartă un *antet* - o listă ordonată a dimensiunilor de traversat - şi o *etichetă* de *n* bit. Lista este inițializată cu ordinul dimensiunilor ce vor fi traversate de o rutare *e-cub*. La fiecare nod din cale, conexiunile sunt încercate în ordinea dată în listă pînă când este întâlnită una fără defect. Conexiunea bună este reținută şi aceea dimensiune este extrasă din listă. Dacă toate dimensiunile din listă poziționează calea pe conexiuni defecte, atunci o dimensiune de extindere este aleasă din exteriorul listei. Conexiunea de această dimensiune este luată în considerare şi lista se măreşte prin adăugarea acestei dimensiuni la sfârşit.

În [Prad 96] se descriu şi alţi algoritmi de rutare. Este de notat că generalizări ale *hipercuburilor* şi algoritmi de rutare au fost propuşi de Agrawal şi Bhuyan în 1988 [Agra88].



a. Direcţionare *E-cub* la un *hipercub* tolerant la defect



b. Direcţionare adaptivă la un *hipercub* defect.

Fig. 4.30 Direcţionare în *hipercuburi* cu funcţionare corectă şi în cele cu defecte

4.5.4 Reconfigurarea în rețele *plasă*

RIN *plasă* bidimensionale au fost propuse pentru calitățile lor de scalabilitate, modularitate şi expandabilitate. Fig. 4.31 prezintă 16 procesoare dispuse într-o plasă bidimensională. O modificare a *plasei* o reprezintă *torul* unde de asemenea există conexiuni finale (end-around) pentru procesoarele de pe direcțiile externe X şi Y.

Există mai multe strategii pentru a face aceste RIN *tolerante la defect*.

O primă strategie se bazează pe ideea de-a face aceste RIN capabile ca, în caz de defecțiune, să sufere o degradare grațioasă. Pentru arii bidimensionale de calculatoare, se propune *ştergerea unei întregi linii sau coloane de procesoare* la care aparține un procesor defect. În fig. 4.31 se prezintă organizarea de bază a unei scheme de reconfigurare în care se utilizează comutatoare speciale.

Dându-se un set de procesoare defecte, un set de coloane şi linii este şters din topologia plasei în vederea obținerii unei *subplase fără defecte* (*fault-free submesh*).

Din nefericire o astfel de schemă degradează performanțele în mod semnificativ pentru un singur defect. Acest lucru se întâmplă datorită faptului că se exclud un număr mare de procesoare active (o linie sau o coloană) în timpul reconfigurării.

O altă strategie ce a fost propusă constă în utilizarea unui set de *tehnici de extensie* (*sparing techniques*) pentru ariile bidimensionale prin utilizarea unor circuite complexe de comutare a nodului.

În metoda de reconfigurare directă, se adaugă câteva coloane şi linii în plus la aria $N \times N$ de procesoare. Reconfigurarea este obținută de-a lungul unei direcții specifice prin efectuarea unei redenumiri globale a tuturor procesoarelor în care indicii logici ai procesoarelor (i, j) sunt mapați în indici fizici (k, h) a procesoarelor active. Se utilizează o *funcție de mapare*. Dându-se un procesor defect (i, j), este necesar să se decidă asupra unei direcții (orizontale sau verticale) pentru reconfigurare ; aceasta înseamnă că celula defectă va fi înlocuită cu celula de extensie în direcție orizontală sau verticală printr-o *conexiune de ocolire* (*by-pass link*) în direcție orizontală sau verticală.

Ordinea paşilor de reconfigurare este critică. Strategia recomandată a fost să se clasifice primul defect într-o coloană când se scanează de jos în sus, ca fiind un defect vertical, şi toate celelalte defecte ca fiind orizontale. Dacă nici o linie nu are mai mult de un defect orizontal, atunci reconfigurarea este posibilă. În fig. 4.32 se prezintă un

exemplu de reconfigurare directă a unei arii de 5 × 5 augmentate cu o coloană şi cu o linie suplimentară . Sunt necesare conexiuni şi comutatoare auxiliare.

Schema prezentată poate tolera doar câteva configuraţii de defect. În orice linie doar un defect poate fi marcat ca şi orizontal şi în orice coloană doar un defect poate fi vertical.

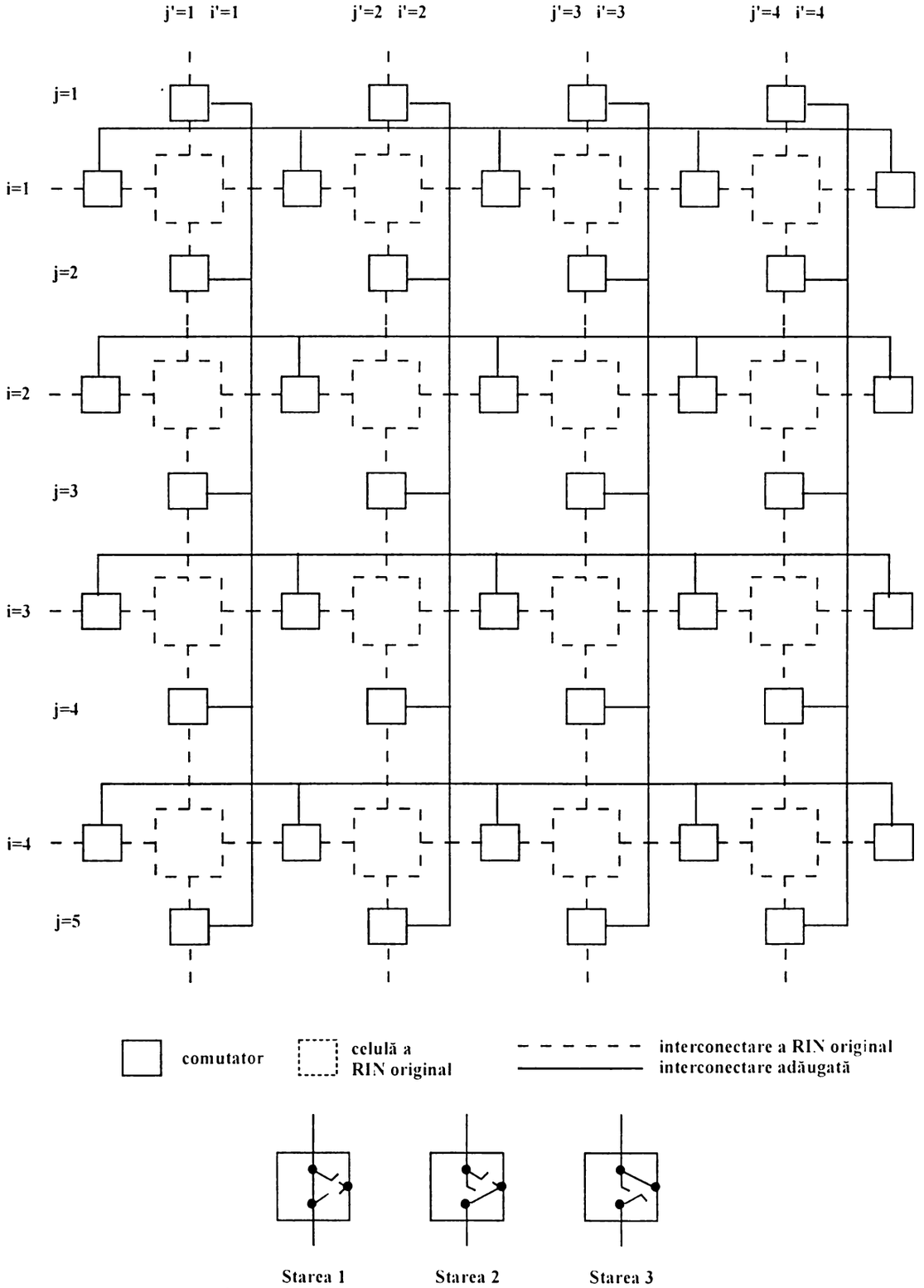


Fig. 4.31 Reconfigurarea unei RIN *plasă* prin eliminarea unei linii sau coloane

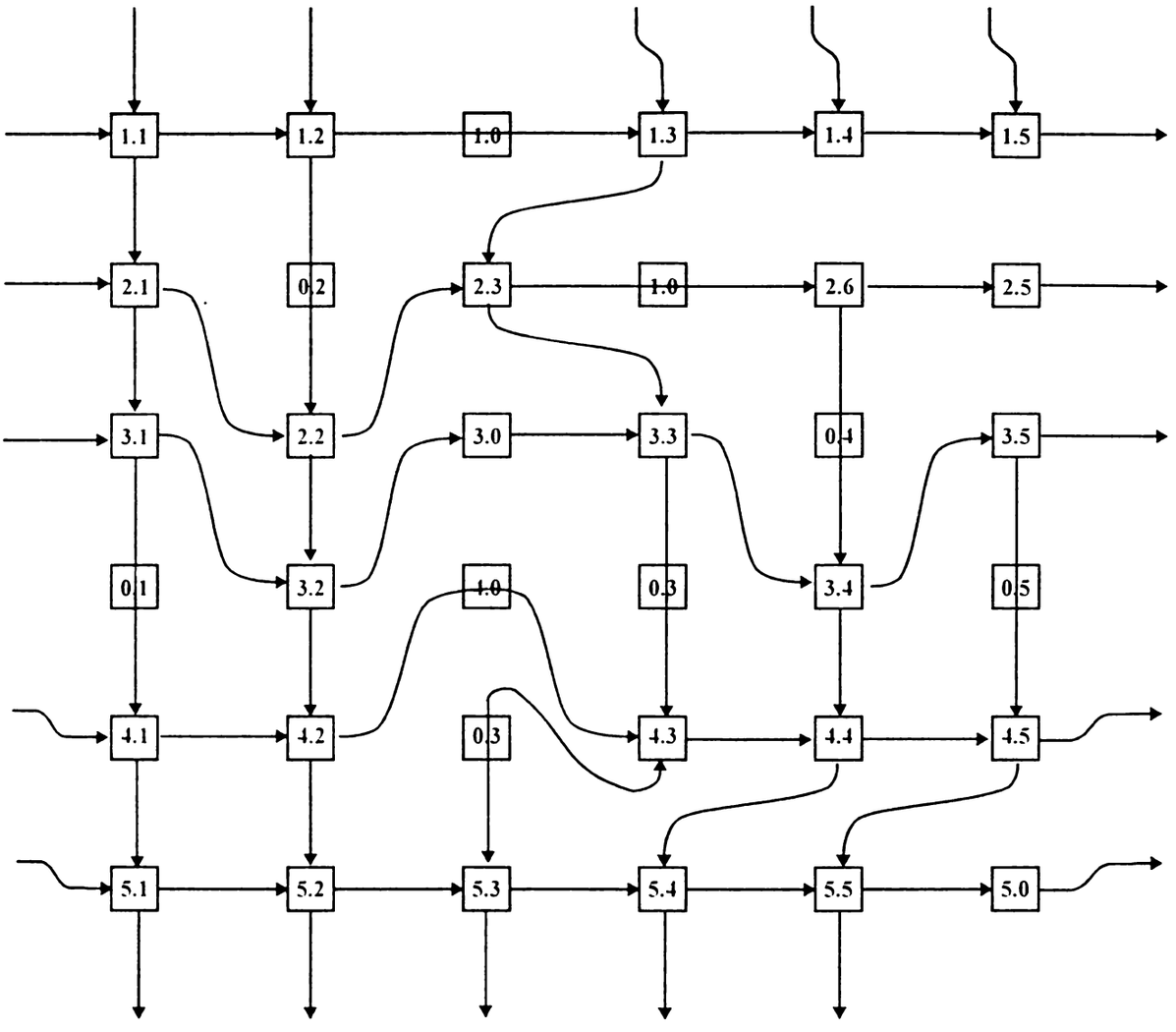


Fig. 4.32 Reconfigurarea unei RIN *plasă* prin mapare logică

4.5.5 Reconfigurarea în rețele *arbore*

RIN *hipercub* pot conecta P procesoare utilizând un grad de log (P) şi o distanță maximă de log (P) iar *plasa* poate conecta P procesoare utilizând un grad de 4 şi o distanță maximă de $\sqrt{P}$. RIN *arbore* pot conecta P - 1 procesoare utilizând un grad de 3 şi o distanță maximă de $2(\log(P) - 1)$. Sistemele bazate pe acest tip de rețea sunt configurate ca şi arbori binari unde fiecare nod este conectat la doi "copii" şi la "părintele" său într-o manieră recursivă. Astfel de arhitecturi sunt excelente pentru prelucrarea algoritmilor paraleli recursivi.

Există mai multe tehnici de transformare a RIN *arbore* în RIN *arbore tolerant la defect*.

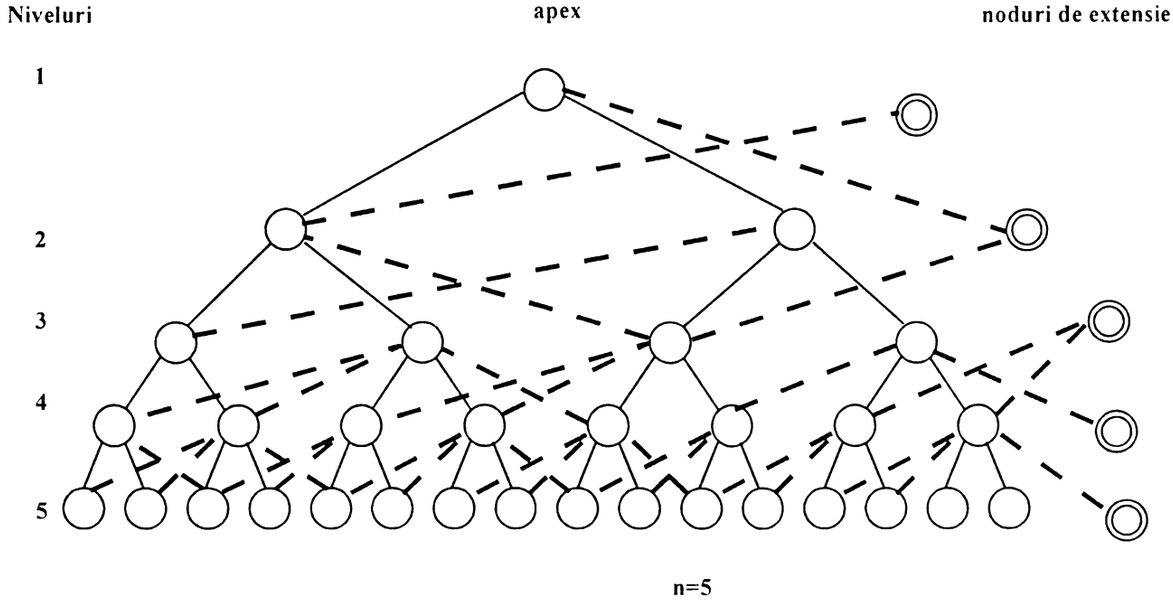
Una dintre cele mai vechi metode de reconfigurare dezvoltă arbori toleranți la un defect cu structură optimă .

Abordarea începe cu un arbore binar complet de *m* niveluri, utilizând *m* extensii, una pe nivel. Câteva conexiuni suplimentare sunt prevăzute de la nodurile de extensie ale unui anumit nivel spre nodurile arborelui binar ale următorului nivel superior astfel că în caz de defect, nodul defect şi conexiunile sale sunt înlocuite de conexiunile suplimentare.

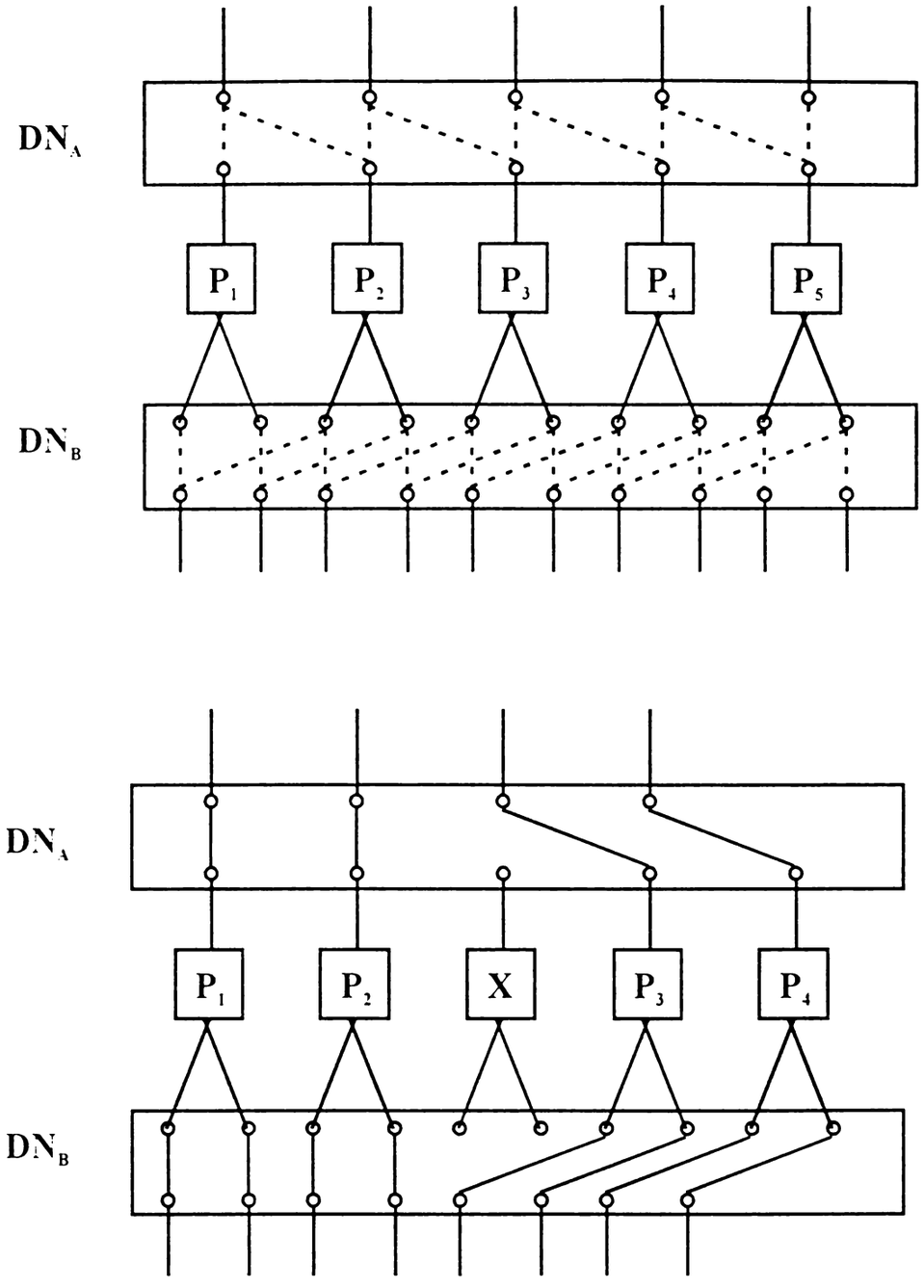
Schema poate fi ameliorată prin *adăugarea de conexiuni redundante*, suficiente pentru a tolera defecte multiple.

Un nod suplimentar este utilizat pe nivel în partea dreaptă a arborelui. Fiecare nod al arborelui include extensiile; se conectează la cei doi "copii" ai săi şi de asemenea la cei doi "copii" ai vecinului său din stânga şi la unul din "copii" vecinului din dreapta. Figura 4.33a prezintă schema de bază pentru reconfigurare. Când un procesor se defectează, toate conexiunile unui procesor spre dreapta sa sunt reajustate spre vecinii din dreapta. Pentru a reduce complexitatea conexiunilor redundante, pot fi utilizate două rețele de decuplare, situație prezentată în fig. 4.33b.

Schemele au fost extinse cu noduri de extensie multiple per nivel, unde un nod de extensie este poziționat pentru 2^i noduri.



a. Reconfigurarea în RIN *arbore*.



b. Rețea de decuplare în RIN *arbore*

Fig. 4.33 Reconfigurare în RIN *arbore*

4.5.6 Reconfigurarea RIN *multinivel*

Terminologie.

In acest paragraf vom detalia terminologia de bază utilizată în descrierea RIN-MN *tolerante la defect*.

In fig. 4.34 se prezintă structura hardware a unei RIN-MN.

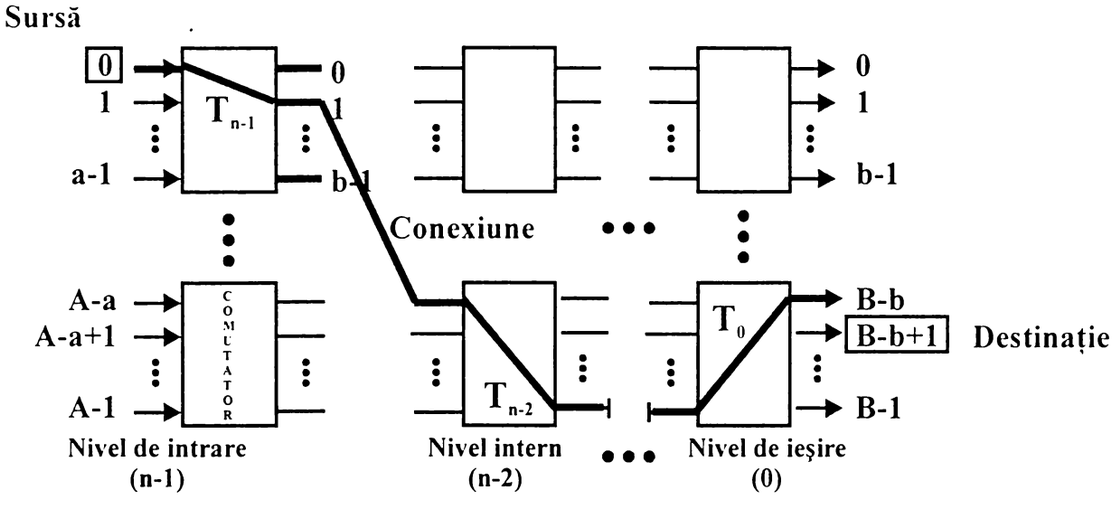


Fig. 4.34 O diagramă generică a RIN-MN evidențiind o cale

O RIN-MN este o colecție de *comutatoare* şi *conexiuni* între comutatoare. Un semnal intră sau părăseşte o rețea printr-un *port*. O rețea cu A porturi de intrare şi B porturi de ieşire este o rețea $A \times B$.

Comutatoarele sunt dispozitive *multiport*.. In RIN-MN sunt utilizate comutatoare de tip *selector* care la un moment dat conectează doar una din intrările sale la una din ieşirile sale.

O RIN-MN *tolerantă la defect* este una ce oferă servicii chiar când conține una sau mai multe componente defecte. Un defect poate fi fie *permanent* fie *tranzitoriu*. Vom considera în continuare doar *defecte permanente*. Toleranța la defect este definită doar în raport cu un *model ales de toleranță la defect* (*fault-tolerance model*) care are două părți. *Modelul de defect* (*fault-model*) caracterizează toate defectele ce se presupune că pot apare, declarând modurile de defectare. Pentru fiecare componentă a rețelei *criteriul de toleranță la defect* (*fault-tolerance criterion*) reprezintă condiția ce trebuie întâlnită pentru ca rețeaua să poată fi considerată că a tolerat un defect(e) dat. Un model de defect poate fi ales cu caracteristici ce simplifică analiza de fiabilitate, chiar

dacă acele caracteristici se depărtează semnificativ de realitate. In timp ce criteriile de toleranță la defect reflectă capabilitățile operaționale normale (fault-free) ale rețelei, în realitate situațiile sunt mai complexe.

O rețea este *tolerantă la un singur defect* (*single fault tolerance*) dacă ea poate funcționa așa cum a fost specificată de criteriul de toleranță la defect. In general, dacă orice set de *i* defecte poate fi tolerat, atunci rețeaua este *tolerantă la i defecte* (*i-fault tolerant*).

RIN-MN pot fi descrise atât *topologic* (*grafic*) cât şi *algebric*. Topologia unei rețele reprezintă configurația de conexiuni din structura sa. Configurația poate fi reprezentată printr-un graf. Topologia este determinată de proiectarea comutatoarelor şi de configurația conexiunilor. Nodurile din graful unei RIN-MN pot fi numerotate şi atunci rețeaua este descrisă în termeni de relații algebrice între noduri. Modelul algebric este util în discutarea strategiilor de control şi comunicare.

Reamintim că există trei forme de bază pentru conexiuni într-o rețea.

O *conexiune unu-la-unu* (*one-to-one connection*) trece informația de la un port al RIN, *sursa*, la alt port al RIN, *destinația*. Ruta exactă parcursă de informație constituie *calea* sa. Conexiuni multiple unu-la-unu pot fi active simultan.

O *conexiune de permutare* este un set de conexiuni unu-la-unu astfel încât două conexiuni unu-la-unu să nu aibă aceeaşi sursă sau destinație. Această clasă de conexiuni sunt utile doar în contextul rețelelor cu un număr egal de surse şi de destinații. Fluxul de informație ce circulă simultan de la o sursă spre două sau mai multe destinații este suportată de o *conexiune de difuzare* şi calea urmată se numeşte *cale de difuzare* (*broadcast path*).

Etichetele de rutare (*routing tags*) reprezintă un mod de a descrie o cale printr-o rețea şi de a fi oferite pentru un control distribuit al rețelei. Controlul este *distribuit* dacă dispozitivele ce utilizează rețeaua generează propriile etichete de direcționare şi comutatoarele rețelei se pot poziționa bazându-se pe informația conținută în etichetă. În fig. 4.34 se prezintă un comutator în etajele n-1, n-2, şi 0, ce este poziționat de biții T_{n-1}, T_{n-2} şi T_0 din etichetă. Etichetele de rutare sunt importante pentru RIN-MN *tolerante la defect* deoarece ele sunt capabile să specifice o cale funcțională, dacă ea există; limitările în numărul şi dimensiunile etichetelor se reflectă în limitări ale toleranței la defect.Există trei metode utilizate de surse pentru a genera etichete de rutare ce specifică o cale liberă de defect. Cu o *rutare nonadaptivă* o sursă realizează existența unui defect doar când calea ce trebuie stabilită traversează componenta

defectă din RIN. Elemente de identificare al defectului sunt transmise spre sursă, care încearcă următoarea cale alternativă. Această abordare necesită un număr redus de circuite dar şi performanţele oferite sunt reduse.

Există două variante de *rutare adaptivă*. Cu *modificarea la solicitare* (*modification on demand*) o sursă menţine un tabel cu defectele pe care le-a întâlnit în încercarea de-a stabili cai şi utilizează această informaţie pentru a ghida o nouă rutare.

Cu *notificarea difuzării* (*broadcast notification*) unei erori, toate sursele sunt notificate asupra componentelor defecte pe măsură ce ele sunt diagnosticate.

O cale liberă de defect nu trebuie să fie specificată de o sursă, dacă etichetele de rutare pot fi modificate ca răspuns la defectele întâlnite pe măsură ce o cale este urmată sau stabilită. Această *rutare dinamică* poate fi realizată în RIN-MN construite cu comutatoare capabile de realizarea revizuirilor necesare ale etichetelor de rutare.

În literatura [AA887], [AR92], [MI096], [HA96], [MI098] se face o analiză sistematică şi o comparaţie a RIN-MN *tolerante la defect*. Inițial, se acordă atenţie câtorva căi de comunicaţie conducând la o lipsă a toleranţei la defect şi la o fiabilitate scăzută.

- adăugarea unor niveluri suplimentare de comutatoare
- adăugarea unor conexiuni suplimentare
- adăugarea unor porturi suplimentare.

Figura evidenţiază toate căile care utilizează conexiunea *l*. O defecţiune la nivel de *l* deconectează aceste căi. Situaţia devine dramatică atunci când dimensiunea RIN creşte deoarece numărul de căi ce traversează o conexiune dată, creşte liniar cu *N*.

Tehnicile pentru obţinerea toleranţei la defect în RIN-MN pot fi clasificate după modul în care ele implică *modificarea topologiei sistemului*.

Există metode cunoscute de creare a toleranţei la defect în RIN-MN care nu modifică topologia sistemului. Aceste tehnici utilizează coduri detectoare de erori, *bit-slice* etc. Ele sunt însă limitate la o singură defecţiune.

Concluziile desprinse demonstrează că domeniul RIN-MN *tolerante la defect* este încă în stadiul incipient, având largi posibilităţi de dezvoltare.

Aplicarea unor metrici fundamentale ca şi *modelul de defect* şi *criteriul de toleranţă la defect* vor permite arhitecţilor de calculatoare să evalueze în continuare noi tipuri de RIN-MN.

RIN-MN au fost descrise în §2.4.4. Ele se caracterizează prin următorii parametri constructivi:

- $N=m^n$ intrări
- n niveluri de comutare
- N/m elemente de comutare tip *crossbar* de talie $m \times m$.

O particularizare constă în faptul că uzual $m = 2$ şi atunci vor exista:

- $\log_2 N$ niveluri a câte $N/2$ comutatoare 2×2
- $O(N \log_2 N)$ costul hardware.

RIN cu $\log_2 N$ niveluri are două proprietăţi importante:

- existenţa unei conexiuni unice de la orice intrare la orice ieşire
- orice cale de comunicaţie de la o intrare la o ieşire poate avea conexiuni comune.

Apar două dezavantaje datorită acestei topologii:

- conexiune I/E poate fi blocată de către o conexiune stabilită anterior
- defectarea chiar a unei singure conexiuni sau a unui comutator deconectează câteva căi de comunicaţie conducând la o lipsă a toleranţei la defect şi la o fiabilitate scăzută.

În fig. 4.35 se ilustrează această situaţie pentru o RIN *omega* având 16 intrări şi 16 ieşiri construită cu comutatoare 2×2.

Reţelele studiate din punct de vedere al performanţelor sunt:

- *delta* şi *cub cu nivel suplimentar* (*extra stage cube*)
- *omega cu cale multiplă* (*multipath omega*)
- RIN-*F*
- RIN cu *redundanţă dinamică*
- RIN-β
- RIN-*C augmentat* etc.

Orice schemă de toleranţă la defect care utilizează noţiunea de modificare a topologiei RIN implică detectarea şi localizarea defectelor în reţea, urmată de o reconfigurare corespunzătoare. In literatură, [Prad 96] se menţionează mai multe metode de detectarea defectului în RIN-MN. De exemplu se utilizează tehnici de testare *off-line* sau *on-line* în care se aplică un număr redus de teste independent de talia reţelei. Se poziţionează toate comutatoarele aplicând la intrări configuraţii 01 sau 10. Dacă nu există defecte se poate demonstra că porturile de ieşire vor furniza configuraţii 01 sau

10. Apoi se poziționează toate comutatoarele în poziția "*încrucișat*" (*cross-over*) și se aplică aceleași configurații 01,10 iar la ieșiri se testează dacă apar 10, 01.

Dacă apare o defecțiune, câteva dintre ieșiri vor furniza o configurație ilegală adică 00 sau 11. Configurațiile pot fi utilizate pentru localizarea defectului.

Odată ce un defect a fost detectat în RIN-MN există numeroase strategii pentru reconfigurarea rețelei. Acestea includ replicarea întregii rețele, adăugarea unui nivel suplimentar de comutatoare, varierea dimensiunii comutatorului, adăugarea de conexiuni suplimentare între niveluri, adăugarea de porturi suplimentare, etc.

În continuare se va prezenta o metodă de transformare a unui RIN-MN cu o *singură cale* într-un RIN-MN cu *căi multiple* între fiecare pereche de I/E a rețelei.

Factorii care contribuie la *complexitatea comutatorului* (*switch complexity*) într-un RIN-MN sunt: numărul de nivele de comutare, numărul de comutatoare/nivel și talia comutatoarelor (numărul de contacte).

O altă măsură a costului unui RIN-MN este *complexitatea conexiunii* (*link complexity*) care depinde de numărul de conexiuni în cadrul unui nivel și de numărul de niveluri. Dintre cele două tipuri de conexiuni în RIN-MN, cele între niveluri sunt mai scumpe decât cele în cadrul unui nivel.

Un *model de defect* (*fault-model*) are în vedere efectele defectelor fizice în operarea unui sistem. Se utilizează trei modele: *stuck-at fault*, *modelul defectului de conexiune* și *modelul defectului de comutator*.

In modelul *stuck-at*, o defecțiune cauzează blocarea comutatorului într-o stare particulară indiferent de starea intrării de comandă. Astfel se afectează capacitatea de a stabili conexiunile necesare. Comutatorul poate fi folosit pentru stabilirea căii de comunicație dacă starea blocată a comutatorului este de asemenea starea solicitată. In modelul *defectului de conexiune*, o defecțiune afectează o conexiune individuală a unui comutator, lasând partea care rămâne a comutatorului, operațională.

In modelul de *defect de comutator* – cel mai consistent dintre modelele prezentate – o defecțiune face comutatorul total inutilizabil. Exemplul prezentat a fost proiectat să tolereze astfel de defecte de comutator.

Topologia RIN-MN *cu căi multiple* poate permite ca rerutarea să fie făcută doar la sursă sau în anumite puncte fixe din rețea. In acest caz o linie ocupată, o linie defectă, sau un comutator defect, întâlnite în timp ce trebuie stabilită o cale, pot necesita o *parcurgere inversă a traseului* (*back tracking*) până la un nivel la care există o bifurcație și de acolo să se încerce stabilirea altei căi de comunicație. Dacă căile între

fiecare pereche I/E într-o RIN-MN *cu căi multiple* are o bifurcație la fiecare nivel, atunci redirecționarea poate fi făcută din mers fără procedura de parcurgere inversă.

Un *graf de redundanță* oferă o cale convenabilă pentru a studia proprietățile RIN-MN *cu căi multiple* ca şi numărul de defecte tolerate sau tipul posibil de redirecționare. Acest graf descrie toate căile posibile între o sursă şi o destinație în cadrul unei rețele. El constă din două noduri distincte – sursa S şi destinația D – şi restul nodurilor corespunde cu comutatoarele ce apar pe căile între S şi D.

Condițiile de proiectare pentru un RIN-MN *omega* tolerant la defect sunt:

- *toleranță la defect de comutator* în toate nivelurile, inclusiv primul şi ultimul
- posibilitatea de *rerutare* fără parcurgerea inversă a traseului
- *reparabilitatea on-line* a componentelor defecte
- *complexitatea* scăzută a comutatorelor şi a conexiunilor.

RIN-MN *omega cu nivel suplimentar*

O RIN-MN *omega* căreia i se adaugă conexiuni şi comutatoare suplimentare pentru a o face tolerantă la defect se numeşte o RIN-MN *omega cu nivel suplimentar* (*extra-stage shuffle-exchange network*).

În literatură [BT88] ,[BT89],RIN-MN *omega* şi RIN-MN *omega mărită* se notează,din rațiuni de simplitate,cu SEN (Shuffle Exchange Network),respectiv SEN+

Vom utiliza următoarele convenții de reprezentare:

- nivelurile sunt numărate de la 0 la $n-1$
- comutatoarele în fiecare nivel sunt numărate de la 0 la $N/2-1$
- comutatorul i de pe nivelul j este referit ca şi comutatorul (i,j)
- conexiunile superioare (denumite conexiuni 0) de comutatoare i sunt numărate $2i$
- conexiunile inferioare (denumite conexiuni 1) sunt numărate $2i+1$.

Rerutarea se face într-o manieră distribuită utilizând eticheta binară a destinației ca şi *etichetă de rutare* (*routing tag*).

Fie $d_0, d_1, \ldots, d_{n-1}$ adresa binară a destinației dorite.

Comutatoarele de-a lungul căii din nivelul i utilizează bitul d_i şi fac o conectare spre ieşirea superioară dacă bitul este 0 şi spre ieşirea inferioară dacă bitul este 1. Generarea etichetei de direcționare este sarcina nodului sursă.

Pentru a obține toleranța la defect se exploatează faptul că există subseturi de comutatoare în fiecare nivel care se găsesc pe căile ce conduc la același subset de destinații. De exemplu, orice comutator de pe nivelul 0 conduce la orice destinație; toate comutatoarele *pare* de pe nivelul 1 conduc la destinații de la 0 la $N/2-1$; toate comutatoarele *impare* de pe nivelul 1 conduc la destinații numerotate de la $N/2$ la $N-1$. Toate comutatoarele de pe un nivel dat care conduc la același subset de destinații cuprind un *subset conjugat* de comutatoare. Comutatoarele de pe fiecare nivel sunt partiționate în câteva subseturi conjugate.

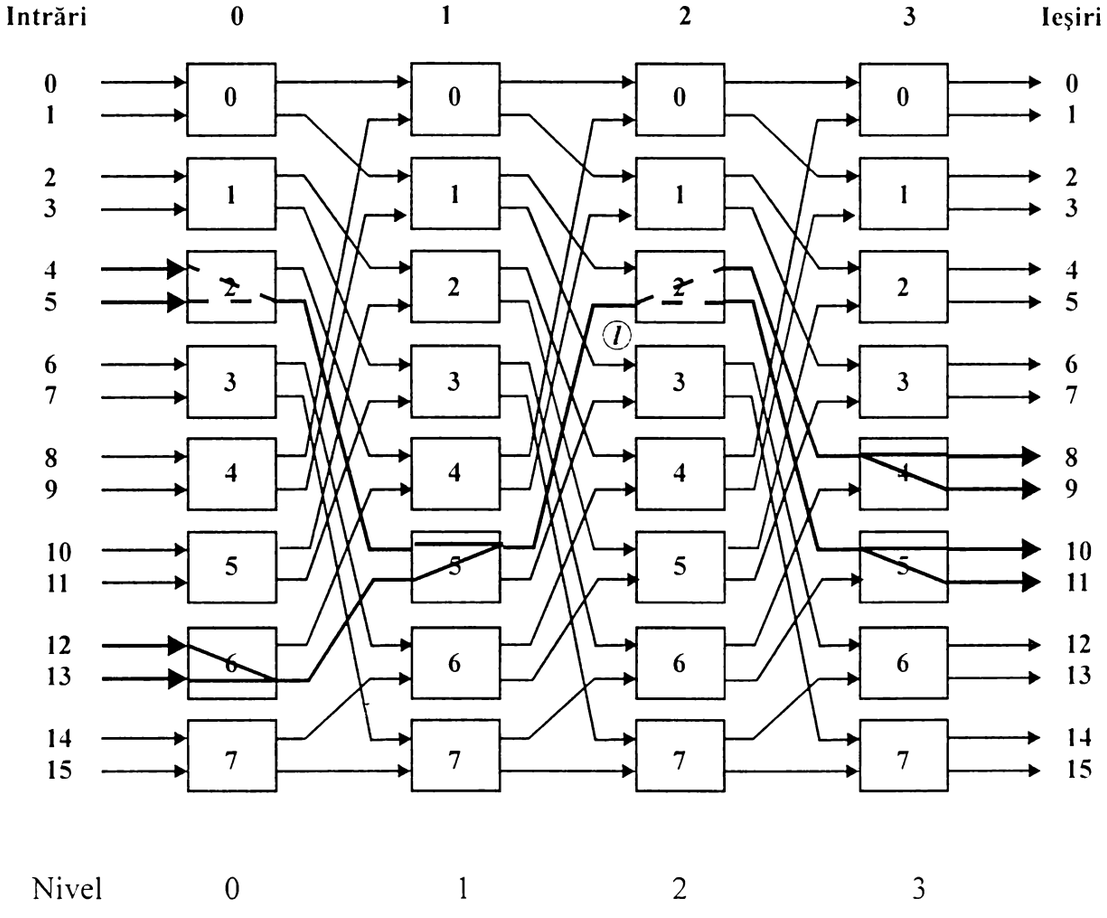


Fig. 4.35 RIN *omega* 16 × 16. Există două căi ce utilizează conexiunea *l*

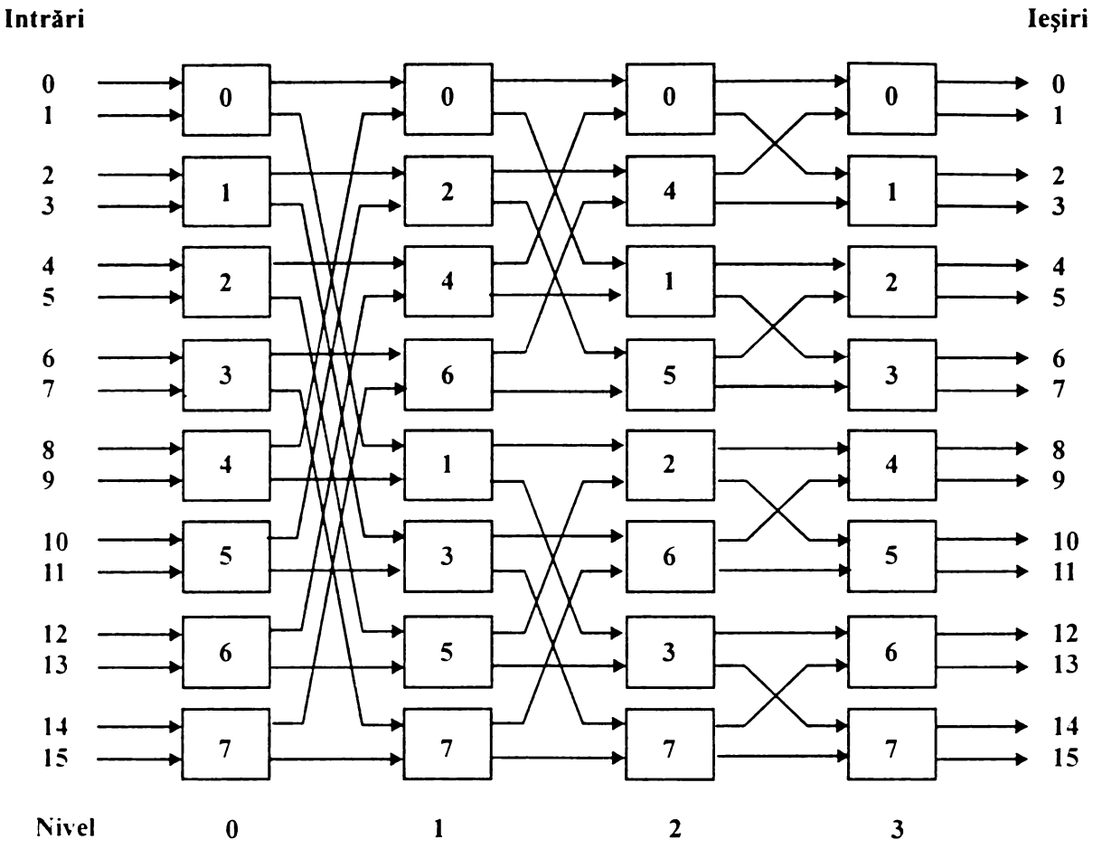
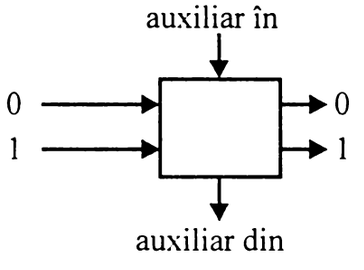
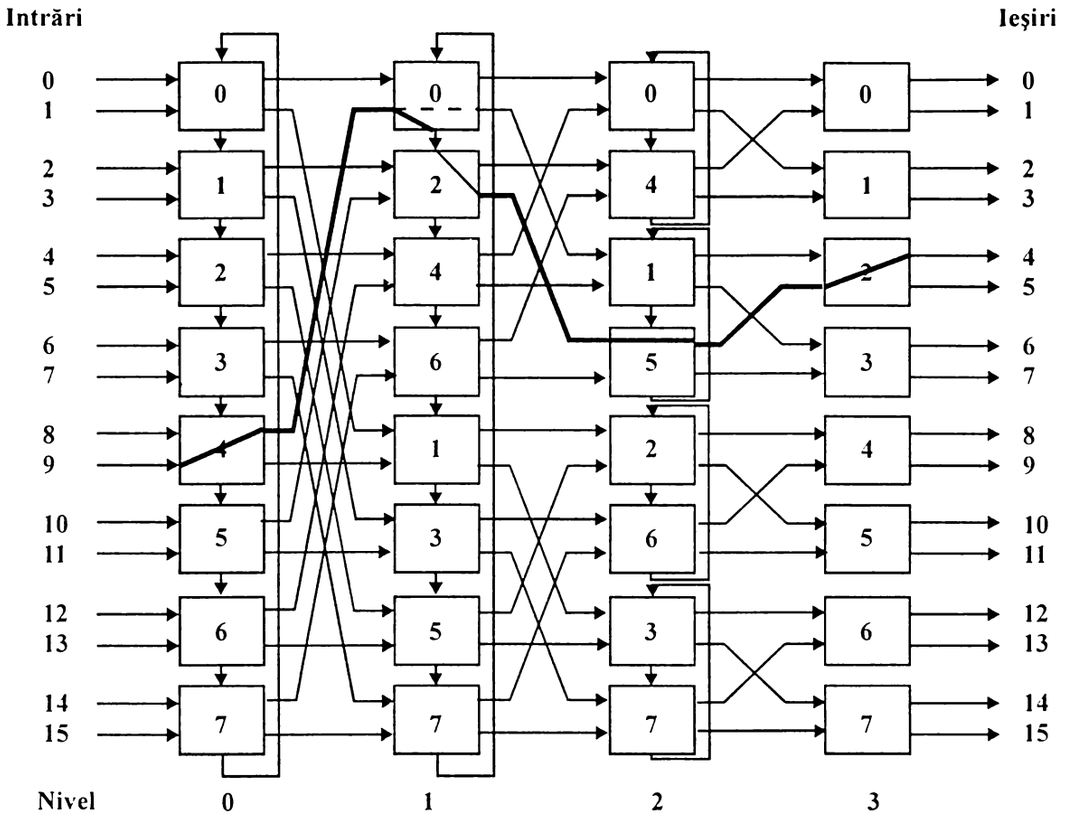


Fig. 4.36 RIN-MN redesenată, cu identificarea subseturilor conjugate de comutatoare.



a) Comutator necesar pentru formarea buclelor de comutare

Fig. 4.37 Comutatorul defect (1, 2) este tolerat cînd intrarea 9 este conectată la ieşirea 4



b) RIN_MN *omega augmentat*

Fig. 4.37 Comutatorul defect (1, 2) este tolerat cînd intrarea 9 este conectată la ieşirea 4

In fig. 4.36 se prezintă RIN-MN din fig. 4.35 redesenată pentru facilitarea identificării subseturilor conjugate de comutatoare. De exemplu pe nivelul i, două comutatoare k şi l aparţin aceluiaşi subset conjugat dacă $k \equiv l \bmod 2^i$. Eticheta de rutare parţială necesară pentru stabilirea unei conectări pentru o destinaţie, fie de la comutatorul k fie de la l, este aceeaşi.

In fiecare subset conjugat de comutatoare există câteva perechi de comutatoare denumite *perechi conjugate de comutatoare* (*conjugate pairs of switches*). Comutatoarele într-o astfel de pereche sunt conectate la aceleaşi comutatoare din nivelul următor. De exemplu în fig. 4.35, comutatoarele (0,0) şi (4,0) formează o pereche conjugată deoarece ambele sunt conectate la comutatoarele (0,1) şi (1,1). In general, comutatoarele (i,j) şi $(i+N/4,j)$ unde $0 \le i < N/4$, $0 \le j < n-1$, sunt ambele conectate la comutatoarele $(2i,j+1)$ şi $(2i+1,j+1)$. Subseturile conjugate şi perechile conjugate de comutatoare joacă un rol fundamental în acest tip de scheme.

Schema pentru *crearea de căi multiple* se bazează pe conectarea comutatoarelor aparținând unui subset conjugat prin utilizarea conexiunilor adiționale pentru a forma bucle. Pentru aceasta este necesară înlocuirea comutatoarelor 2×2 cu comutatoare 3×3 de tipul prezentat în fig. 4.37a. Conexiunile verticale utilizate pentru conectarea comutatoarelor pe același nivel sunt denumite *conexiuni auxiliare* (*auxiliary links*). RIN-MN din fig. 4.35, augmentată în această manieră, este prezentată în fig. 4.37b. Această figură prezintă de asemenea cum poate fi obținută *rutarea tolerantă la defect*. Dacă un comutator nu este capabil să proceseze o solicitare de conectare datorită unui comutator defect dispus pe nivelul următor sau datorită unei conexiuni ocupate, el poate ruta solicitarea prin conexiunea auxiliară de ieşire spre următorul comutator din buclă. Următorul comutator poate apoi să facă o conectare la un comutator diferit (corect) dispus pe nivelul următor.

Dacă se presupune că trebuie făcută o conectare între conexiunea de intrare 9 spre conexiunea de ieşire 4, calea de date în condiții normale ar fi prin comutatoarele (4,0), (0,1), (1,2) şi (2,3). In ipoteza defectării comutatorului (1,2), traseul inițial va fi schimbat utilizându-se calea (4,0), (0,1), (2,1), (5,2) şi (2,3).

De fapt se observă că utilizând o conexiune auxiliară ori de câte ori se întâlneşte un defect, se permite oricărei surse să fie conectată la orice destinație în timp ce se tolerează orice comutator singular defect în orice nivel cu excepția celui inițial şi a celui final.

O defectare a unui comutator în nivelul inițial deconectează sursele ataşate acelui comutator de restul rețelei. Similar orice defectare a unui comutator de pe nivelul final deconectează destinațiile ataşate comutatorului. Pentru a extinde toleranța la defect în aceste niveluri, fiecare sursă şi fiecare destinație trebuie să fie prevăzută cu cel puțin un port adițional de I/E, astfel ca ele să poată fi conectate la cel puțin două comutatoare distincte. In fig. 4.38 se prezintă schema pentru conectarea surselor şi destinațiilor echipate cu două porturi fiecare. Un multiplexor 2×1 este plasat la fiecare conexiune de intrare a nivelului 0 şi fiecare sursă este conectată la două MUX-uri distincte. Pentru a prevedea coexiuni adiționale spre destinații, cele $N/2$ comutatoare din nivelul final al RIN-MN sunt înlocuite cu N 1×2 DMUX-uri. Fiecare destinație este direct conectată la două DMUX.

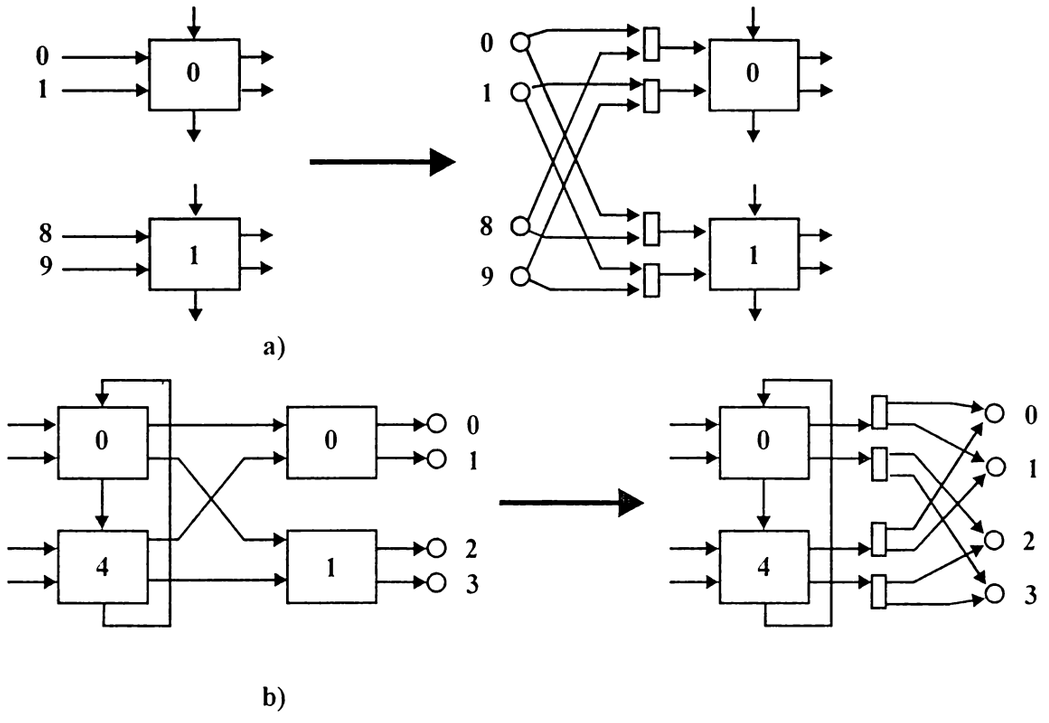


Fig. 4.38 Scheme de realizare unor conexiuni multiple
pentru surse şi destinaţii - adăugarea a MUX-uri 2 ×1 la nivelul de intrare (a);
înlocuirea comutatoarelor din nivelul final cu DMUX-uri 1 × 2 (b)

Dacă fiecare buclă din fig. 4.38 este divizată în două bucle, cu condiţia ca să nu existe două comutatoare într-o buclă dată ce să formeze o pereche conjugată, atunci se obţine o reţea care are nu numai toleranţă la defect singular dar şi câteva proprietăţi ce să faciliteze reparaţia on-line şi întreţinerea.

O astfel de reţea este denumită ASEN (Augmented Shuffle-Exchange Network). In fig. 4.39a se reprezintă un ASEN pentru N=16 iar graful său de redundanţă este dat în fig. 4.39b. Această reţea se numeşte ASEN-Max deoarece ea are numărul maxim de comutatoare pre buclă cu restricţiile că comutatoarele dintr-o buclă aparţin aceluiaşi subset conjugat şi că nu există două comutatoare într-o buclă ce să formeze o pereche conjugată. Numărul de comutatoare dintr-o buclă de pe nivelul i a lui ASEN-Max este 2^{n-2-i}. De notat că în nivelul final (nivelul n-2) există doar un comutator/buclă.

Este posibil să se proiecteze ASEN-Max şi să se reţină proprietatea toleranţei la un singur defect. De exemplu ASEN din fig. 4.40 are o buclă de talia a două comutatoare în fiecare nivel cu excepţia ultimului. In general, un ASEN în care numărul de comutatoare dintr-o buclă pe nivel i este egal cu Min $(2^{n-2-i}, k)$ este denumit ASEN-k.

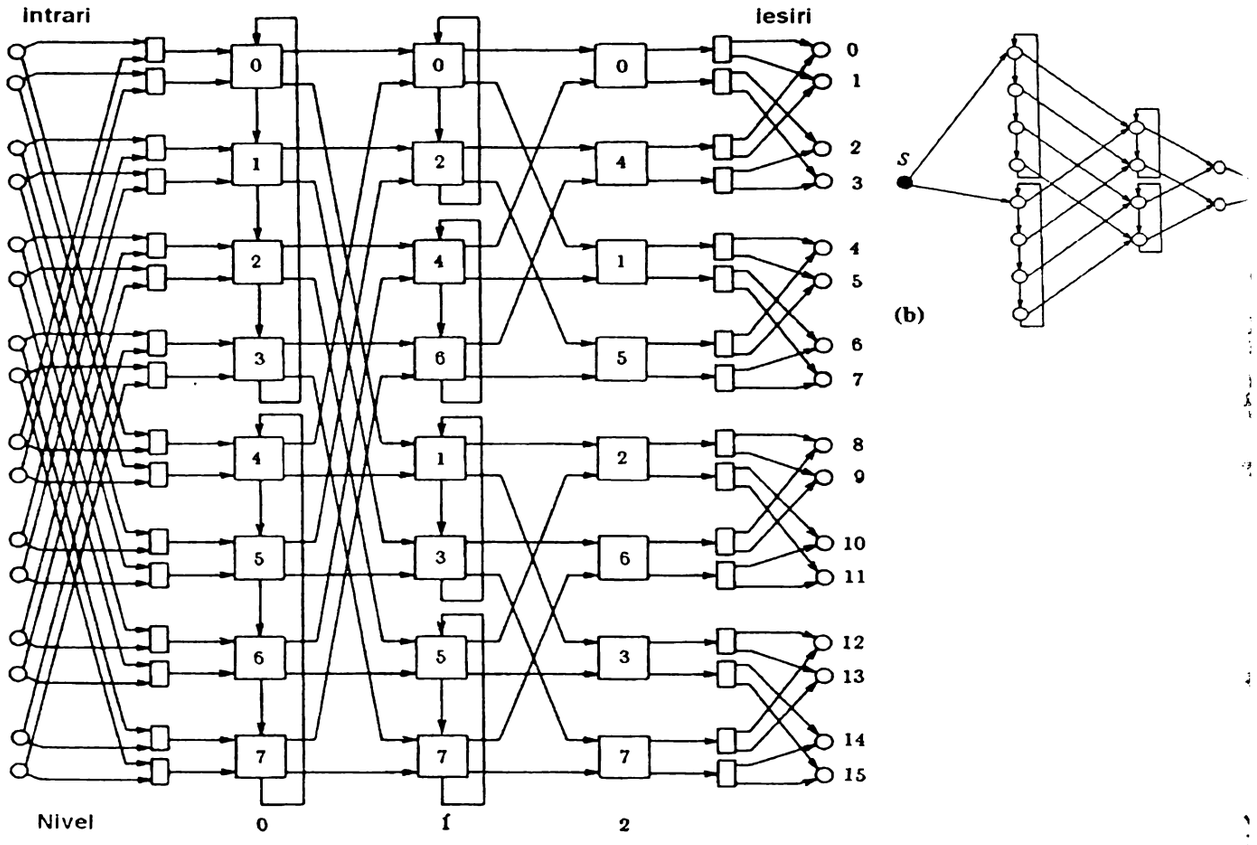


Fig. 4.39 Un ASEN – Max pentru N=16(a); graful de redundanță al acestui circuit(b)

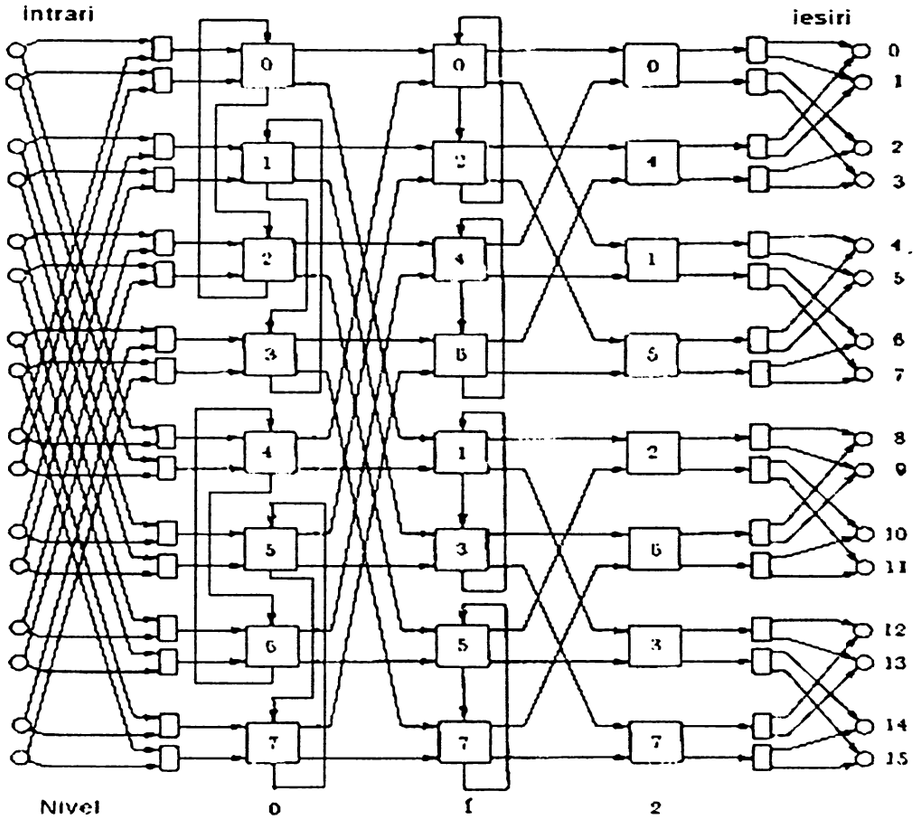


Fig. 4.40 Un ASEN-2 pentru N=16

Toate buclele într-un nivel dat trebuie să conțină un număr egal de comutatoare, ceea ce restrânge pe k la o putere a lui 2.

Grafurile de redundanță pentru ASEN-2 şi ASEN-4 sunt repezentate în fig. 4.41. De notat că buclele formate în toate nivelurile, cu excepția nivelului final, sunt astfel formate încât pentru fiecare buclă există altă buclă care este conectată la acelaşi set de comutatoare pe următorul nivel. Astfel de perechi de bucle sunt denumite *bucle conjugate*.

Rețelele ASEN au log N–1 niveluri şi $N/2$ comutatoare de talie 2×3/nivel, plus N MUX-uri 2×1 şi N DMUX-uri 1×2.

A. Controlul adaptiv al rutării la RIN – MN ASEN.

Siguranța în funcționare şi ameliorarea performanței ce se obțin de la un RIN-MN, depind de cât de efectiv căile alternative sunt utilizate de către algoritmul de rutare. Algoritmii bazați pe *parcurgerea inversă a traseului* (*back-tracking*) sunt costisitori în termeni de resurse hardware şi de timp afectat pentru stabilirea conexiunilor. Un algoritm ce nu se bazează pe parcurgerea inversă este uşor de implementat. El presupune să sursele şi comutatoarele au abilitatea de a detecta defecte în comutatoarele la care sunt conectate.

Algoritmul pentru rutare în ASEN este descris în literatură [KR87]. El este următorul:

- **Pentru fiecare sursă:**

Transmite cererea de conectare comutatorului primar, împreună cu eticheta binară a destinației solicitate. Dacă comutatorul primar sau multiplexorul la intrarea comutatorului primar este defect, atunci trasmite cererea comutatorului secundar. Dacă comutatorul secundar este de asemenea defect, abandonează cererea.

- **Pentru fiecare comutator de pe nivelul** i $(i < n-2)$**:**

Cererile pentru conectare pot sosi la oricare din cele trei conexiuni de intrare.

Pentru fiecare cerere, utilizează bitul i al etichetei destinație asociată şi conectează la conexiunea de ieşire etichetează corespunzător. Dacă conexiunea de ieşire solicitată este ocupată sau nu poate fi utilizată pentru că există un defect în nivelul următor,

rutează cererea prin conexiunea auxiliară de ieşire spre următorul comutator din buclă. Dacă conexiunea auxiliară este de asemenea inutilizabilă deoarece este ocupată sau deoarece este defectă, atunci abandonează cererea. Un defect în DMUX la ieşirea unui comutator de pe nivelul n – 2 este privit ca şi un defect în acel comutator.

- **Pentru fiecare comutator de pe nivelul** $n-2$:

Cererile pot sosi la oricare din cele două conexiuni de intrare.

Pentru fiecare cerere, utilizează bitul $n-2$ a etichetei de rutare şi rutează cererea prin conexiunea de ieşire. Dacă ieşirea cerută este ocupată, abandonează cererea.

- **Pentru fiecare DMUX de la ieşirea nivelului** $n-2$:

Poate primi maximum o cerere.

Dacă soseşte o cerere, atunci utilizează bitul $n-1$ a rutinei de rutare şi fă o conectare spre conexiunea de ieşire superioară sau inferioară în concordanță cu bitul de direcționare 0 sau 1.

- **Pentru fiecare destinație**:

Pot sosi maxim două solicitări.

Acceptă cererile.

Acest algoritm este uşor încorporat în proiectare hardware a comutatoarelor, care este uniformă pentru toate nivelurile.

Algoritmul de rutare este independent de numărul de comutatoare dintr-o buclă.

B. Toleranța la defect şi fiabilitatea rețelelor ASEN.

Un criteriu general de evaluare a robusteții RIN-MN se referă la faptul că fiecare membru a unui subset de surse trebuie să aibă căi către fiecare membru a unui subset de destinații. Probabilitatea ca şi criteriul menționat anterior să fie satisfăcut se numeşte *fiabilitate de tip multiterminal* (*multiterminal reliability*). Ne interesează două cazuri speciale ale acestui criteriu.

Primul caz este acela în care subseturile de surse şi destinații conțin exact un element fiecare. Acest caz special conduce la o măsură denumită *fiabilitate de tip terminal*

(*terminal reliability*) care este probabilitatea ca o pereche sursă/destinație dată să aibă cel puțin o cale fără defect între ele, fiind cunoscut că fiecare comutator are o oarecare fiabilitate.

Fiabilitatea unui comutator este probabilitatea ca el să fie *fără defect (fault – free)*. Fiabilitatea de tip terminal a unei RIN –MN poate fi evaluată din grupul său de redundanță.

Al doilea caz special al criteriului de fiabilitate de tip multiterminal este acela al subseturilor de surse şi destinații conținând toate sursele şi toate destinațiile. Acest caz conduce la supoziția că RIN – MN este defect (sau deconenctat) ori de câte ori toate căile sunt deconectate între unele perechi sursă / destinație şi oferă *fiabilitatea R (t)* a RIN – MN. Funcția *R (t)* poate conduce la calcul *MTTF – timpul mediu până la defectare*.

Deoarece algoritmul de rutare prezentat anterior pentru ASEN nu explorează toate căile disponibile, se introduce funcția ce reprezintă probabilitatea de succes a algoritmului de rutare – *fiabilitatea de tip terminal efectivă – ETR* (*effective terminal reliability*). Ea este aceeaşi pentru toate perechile dacă defectele apar uniform printre comutatoare, deoarece rețelele ASEN sunt simetrice.

În fig. 4.42 se prezintă funcțiile *ETR* pentru fiecare din rețelele ASEN – 2, ASEN – 4 şi ASEN – MAX, obținute analitic şi verificate prin simulare Monte Carlo. Se presupune că probabilitatea unui element de comutare este $p = 0.9$

C. Performanțe ale rețelelor ASEN.

În literatură [KR 87] şi [Prad 96] se prezintă câteva performanțe ale ASEN într-un mediu tipic comutării de circuite şi se arată că ameliorarea față de RIN – MN *cu cale unică* este semnificativă. Ameliorarea se datorează utilizării de conexiuni în interiorul nivelului, care rezultă în lungimi de cale mai mari decât acelea din RIN – MN *cu cale unică*. Oricum creşterea lungimii nu este semnificativă.

Probabilitatea de acceptanță P_A, în acest caz, se referă la faptul că o cerere transmisă de o sursă este acceptată de o destinație fără să fie blocată de alte cereri sau conectări în RIN. P_A se evaluează presupunând că toate celelalte surse generează simultan cererile lor pentru conectare cu o probabilitate p şi țintesc destinații alese aleator, la

începutul unui ciclu. Când două sau mai multe cereri sosesc la comutator solicitând aceeaşi conexiune de ieşire, cereile ce sunt prelucrate sunt alese pe o bază aleatoare iar celelalte sunt blocate sau abandonate.

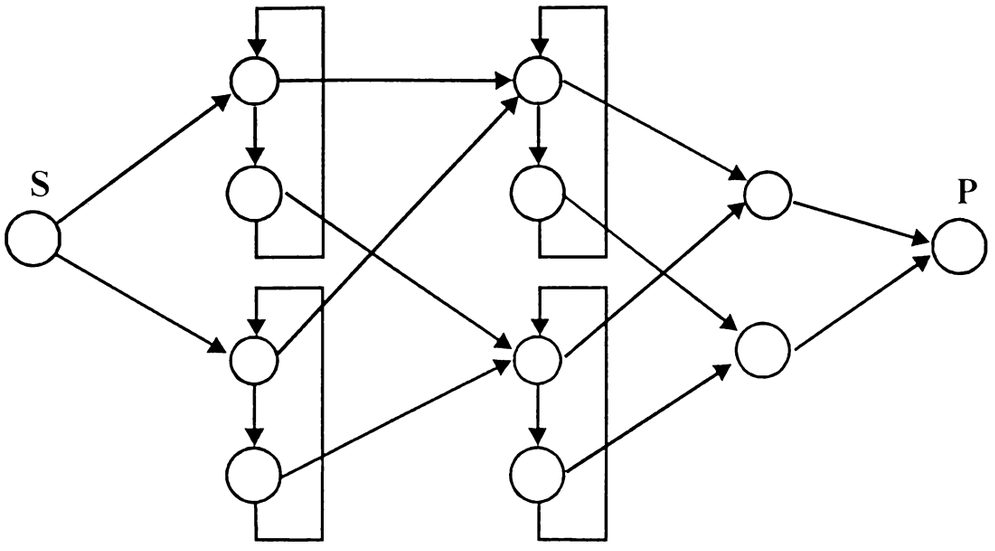
În cazul ASEN-urilor, un număr de maximun trei cereri pot să sosească la un comutator. Se pot procesa cu acelaşi bit al etichetei de rutare până la două cereri, prin conectarea uneia la conexiunea de ieşire cerută spre nivelul următor şi alta la conexiunea auxiliară de ieşire ce conduce la următorul comutator din buclă.

Destinaţiile acceptă până la două cereri / ciclu, deoarece ele au două conexiuni la reţea. P_A este definită ca şi raportul dintre numărul aşteptat de cereri ce au reuşit la numărul aşteptat de cereri emise de către surse.

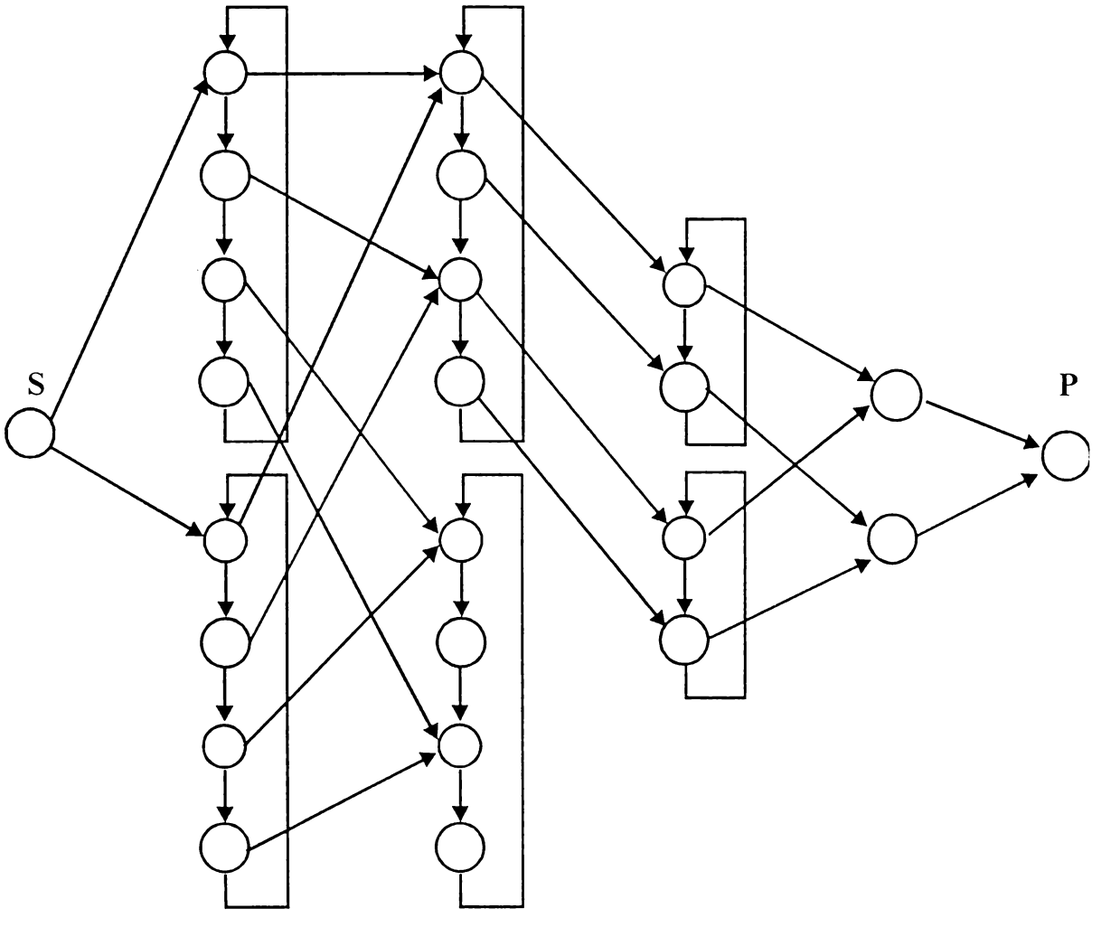
Rezultatul analizei P_A în reţelele ASEN sunt prezentate în fig. 4.42. S-a considerat probabilitatea de generare a cererilor ca fiind 1,0 respectiv 0,5. Se observă că la rata maximă de cereri generate, ASEN – 4 se comportă mai prost decât un *crossbar* dar mult mai bine decât RIN – MN *cu cale unică*. Interesant este faptul că, pentru valori mici ale lui N, dacă rata de generare a cereilor este în limite rezonabile scăzută, P_A este mai ridicată pentru ASEN decât pentru *crossbar*. Există două explicaţii: în primul rând, destinaţiile în ASEN sunt echipate cu două porturi I/E şi pot accepta până la două cereri în loc de una cât poate accepta un comutator *crossbar*; în al doilea rând, gradul de blocare, care este întotdeauna coborât pentru un N mic şi un trafic scăzut, scade prin utilizarea conexiunilor auxiliare.

Preţul plătit în ASEN pentru un P_A mai mare este o mică creştere în lungimea căilor de comunicaţie. În fig. 4.44 se prezintă distribuţia lui P_A în funcţie de lungimea căii. Masa probabilităţii este concentrată în jurul valorilor mici ale lungimii de cale.

Prin combinarea avantajelor oferite de RIN – MN *omega cu o singură cale* cu facilităţile prezentate mai sus, reţelele ASEN devin foarte atractive pentru multiprocesoare şi vor intra în circuitul industrial.



a) Graful de redundanță pentru *ASEN-2*



b) Graful de redundanță pentru *ASEN-4*

Fig. 4.41 Grafurile de redundanță pentru RIN -MN *ASEN*.

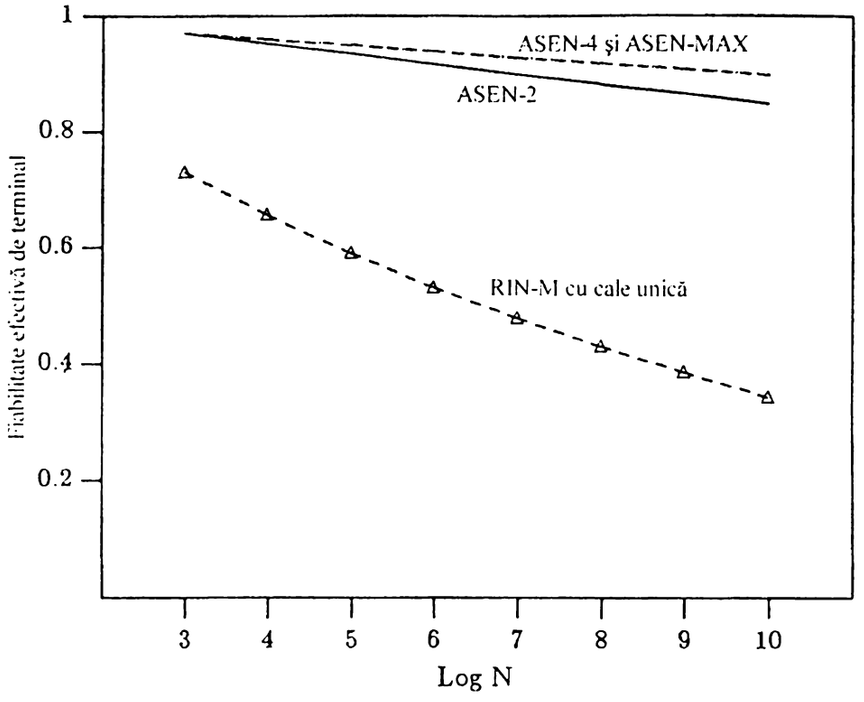


Fig. 4.42 Probabilitatea de terminal efectivă pentru ASEN şi RIN-MN

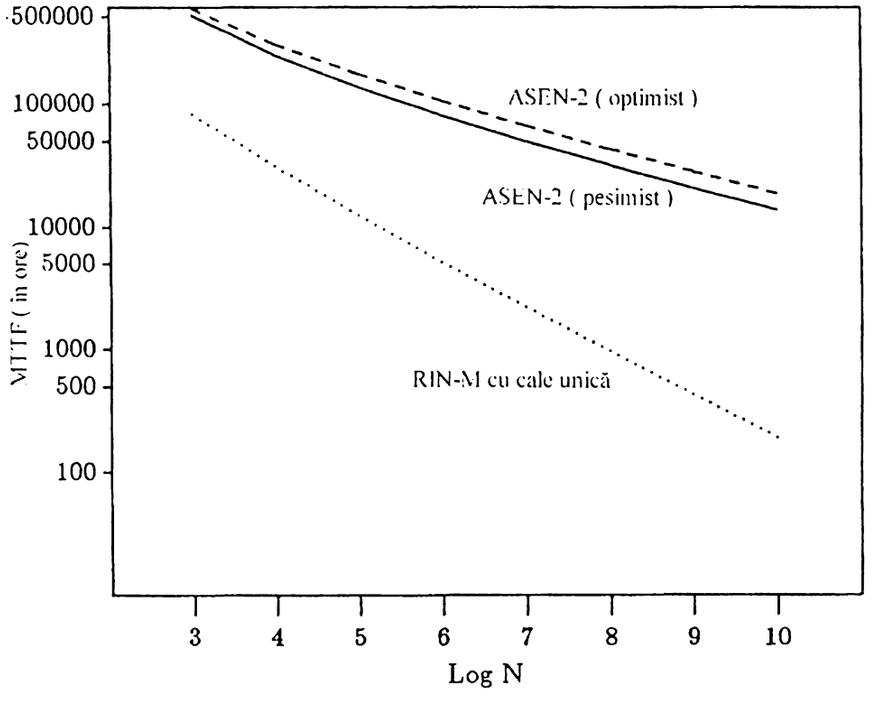


Fig. 4.43 Parametrul *MTTF* pentru ASEN-2 şi pentru RIN-MN cu *cale unică*, la o rată de defectare a comutatorului de 10^{-6} pe oră

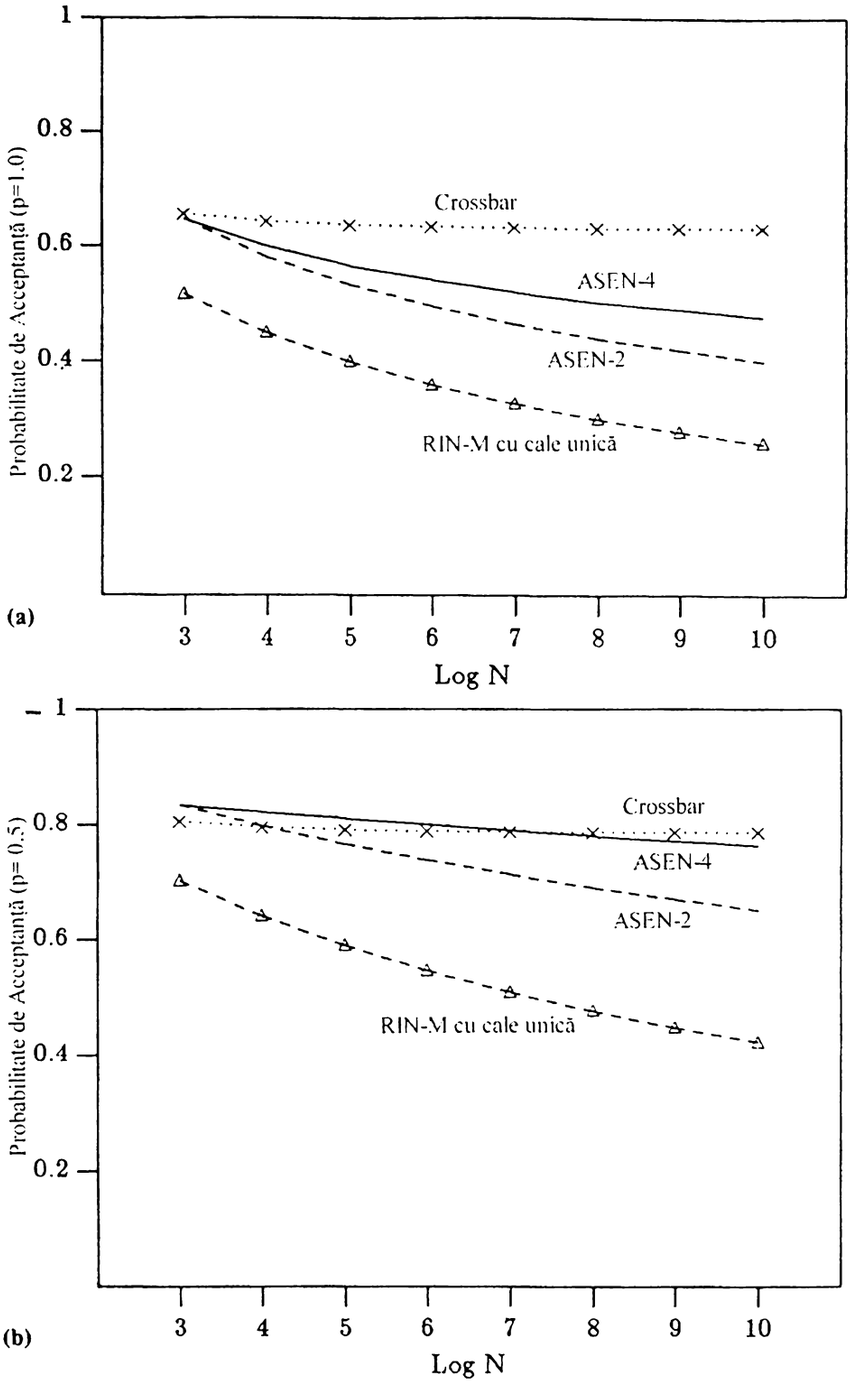


Fig. 4.44 Probabilitatea de acceptanță pentru rețele ASEN, pentru RIN-MN *cu cale unică* și pentru comutator *crossbar*. Probabilitatea de generare a unei erori este de 1.0 pentru (a) și de 0.5 pentru (b)

4.6 CONCLUZII

Capitolul 4 este dedicat analizei unor arhitecturi MIMD *tolerante la defect*. Se analizează mai multe abordări propuse pentru parcurgerea celor trei faze ale toleranței la defect dinamice: *detectarea* defectului şi *localizarea* lui, *recuperarea* din defect şi *reconfigurarea* sistemului în jurul procesorului defect. Dacă defectul este tranzitoriu, recuperarea se face fără reconfigurare. În cazul unui defect permanent, reconfigurarea sistemului devine esențială, evident, acesta suferind o degradare în performanță.

Referitor la detectarea defectului în SMP, se pot concluziona următoarele:

- detectarea prin *duplicare* şi *comparare*, utilizând redundanța spațială, reprezintă o tehnică costisitoare deoarece un SMP cu P procesoare este configurat ca având $P/2$ procesoare. Acest lucru este valabil şi pentru cele M module de memorie configurate în $M/2$ perechi. Un defect detectat într-o pereche de procesoare, dezactivează două procesoare, rămânând active $P-2$ procesoare configurate ca şi $(P-2)/2$.
- *compararea* prin tehnici software este o operațiune consumatoare de timp dar se poate eficientiza prin utilizarea PV-urilor
- *detectarea* defectelor tranzitorii şi a celor intermitente se produce utilizând tehnici de autodiagnoză a căror grad de complexitate creşte odată cu finețea şi acuratețea diagnosticării
- proiectarea şi implementarea circuitelor de codificare şi decodificare în cadrul circuitelor VLSI trebuie să asigure existența unor capacități interne de detectare a propriilor defecte.

Asigurarea toleranței la defect în cazul defectelor tranzitorii sau a celor intermitente se efectuează prin proceduri de recuperare ce asigură reexecutarea calculului. Există penalizări în performanță şi de aceea procedurile de recuperare trebuie reduse atât ca număr cât şi ca durată. De asemenea recuperarea din defect reprezintă pasul final după ce defectul din cadrul unui sistem a fost detectat şi localizat iar sistemul reconfigurat în jurul procesorului defect.

Câteva concluzii se impun:

- tehnicile de recuperare sunt diferite pentru SMP–MP, SMP–TM sau SMP–MPD
- atât *rularea înapoi* cât şi *rularea înainte*, utilizează tehnici de creare şi de exploatare a PV-urilor. Ierarhia de memorie oferă soluții de amplasare atât pentru *datele active* cât şi pentru *datele punctelor de verificare*.
- pentru a evita apariția unei dependențe între frecvența de verificare a PV-urilor şi talia memoriei cache, se pot utiliza PV-uri *virtuale*. Ambele tipuri

de date – active şi PV-uri – sunt conținute într-un sistem de memorii virtuale

- recuperarea din defect prin *rulare înapoi* în cazul multiprocesoarelor ce comunică între ele reprezintă un proces complex deoarece este necesară luarea în considerare a tuturor comunicațiilor şi efectul lor până la cel din urmă PV
- crearea şi verificarea PV-urilor în mod *sincron* evită apariția efectului *domino* şi calculul poate fi repornit după ultimul PV global. Recuperarea este foarte simplă şi timpul de recuperare este redus. Dezavantajul major îl constituie o suprasarcină semnificativă atât la crearea şi verificarea PV-urilor cât şi în procesul de comunicare
- crearea şi verificarea PV-urilor în mod *asincron* are cea mai redusă sarcină dar pentru a putea fi efectiv utilizată în practică trebuie să fie combinată cu tehnica de rulare înapoi bazată pe evidența operațiunilor de rulare (log – based rollback). În schemele recente se preferă implementarea variantei sincrone în loc de cea asincronă. Sarcina de coordonare a creeri şi exploatării PV-urilor în cazul acestor sisteme este neglijabilă în comparație cu sarcina de salvare a stărilor. Astfel sarcina în cazul variantei sincrone este aproximativ aceeaşi ca şi în cazul variantei asincrone dar beneficiul în cazul sincron îl reprezintă o schemă mult mai simplă
- crearea şi verificarea PV-urilor în mod *quasi-sincron* combină uşurința şi sarcina redusă a variantei asincrone cu avantajele timpului redus de recuperare de la varianta sincronă. Reprezintă o abordare atractivă în sistemele moderne ca de exemplu în sistemele de calcul mobile.

Referitor la proiectarea şi utilizarea RIN-MN *tolerante la defect* sunt necesare câteva concluzii:

- RIN-MN *tolerante la defect* au caracteristici ce includ *modelul de defect*, *criteriul de toleranță la defect*, *metoda de creare a toleranței la defect*, *toleranța la defecte singulare* sau *multiple*, *complexitatea rutării* şi *complexitatea hardware*. Modelul de defect şi criteriul de toleranță la defect sunt metrici esențiale pentru orice comparație a capabilităților de toleranță la defect conținute de rețelele studiate.
- codurile corectoare de erori, ce conectează subsisteme redundante în paralel, şi ce împart un subsistem în părți identice conectate în paralel, reprezintă tehnici consacrate pentru obținerea toleranței la defect

- tehnicile de obținere a toleranței la defect, independente de topologia rețelei, sunt:
  - transmisia multiplă cu aceeaşi informație pe căi distincte
  - parcurgerea de mai multe ori a RIN-MN
- tehnicile ce modifică topologia unei RIN-MN pentru a obține o mai bună toleranță la defect prevăzute în faza de proiectare, sunt:
  - adăugarea unui nivel de comutatoare suplimentar cu sau fără conexiuni by-pass
  - adăugarea unor legături suplimentare
  - creşterea numărului de porturi
  - modificarea dinamică a dimensiunii comutatorului şi a numărului de niveluri de comutatoare
  - augmentarea dimensiunii comutatorului prin adăugarea unor conexiuni corespunzătoare
  - replicarea *in integrum* a unei RIN-MN

  În literatură [AAS 87] se descriu pe larg şi alte tehnici. În ultimă instanță importantă este stabilirea unui raport corect performanță/preț în ceea ce priveşte gradul de acoperire la defect.
- RIN-MN se compară alegând un model de defect, valabil pentru toate rețelele, şi un criteriu de toleranță la defect bazat pe o reprezentare realistă a defectelor hardware ce pot apare. Modelul comun de defect presupune că orice componentă a rețelei ar putea să se defecteze şi ca acest defect să facă rețeaua inutilizabilă. Criteriul de toleranță la defect, specifică că o RIN-MN tolerează un defect doar dacă mai poate încă conecta orice intrare la oricare ieşire.

Doar câteva dintre RIN-MN sunt tolerante la defect singular în condițiile respectării tuturor cerințelor impuse anterior. Dacă se relaxează modelul de defect comun şi se presupune o perfectă siguranță în funcționare a nivelurilor de comutatoare, de la intrare şi de la ieşire, atunci vor exista mai multe tipuri de RIN-MN ce pot fi considerate tolerante la defect singular. Sunt necesare însă resurse hardware suplimentare ca şi circuite de evitare a comutatorului defect sau includerea unor conexiuni multiple între fiecare intrare a rețelei şi portul de ieşire, respectiv comutatoarele din rețea. Aria de cercetare a RIN-MN *tolerante la defect* este departe de a fi epuizată iar abordarea noilor tehnologii hardware şi software în contextul mai larg al dependabilității vor asigura ameliorarea semnificativă a performanței acestora.

5. MODELAREA DEPENDABILITĂȚII PENTRU SISTEMELE MULTIPROCESOR CU MEMORIE PARTAJATĂ

SMP-MP trebuie să ofere un nivel ridicat de performanță și, în plus, să fie *dependabile* în prezența defectelor. Proiectarea SMP-MP în condițiile obținerii unei puteri de calcul ridicate și a unei dependabilități superioare, este foarte complexă. Prin modelare se pot identifica cerințele unei bune dependabilități, identifica sursele de limitare a performanțelor și, prin urmare, selecta o arhitectură optimă.

În cap. 4 s-au prezentat atributele unui SMP tolerant la defect.

Reamintim că *fiabilitatea* unui sistem este o funcție *R(t)* definită ca fiind probabilitatea condițională ca sistemul să funționeze corect în intervalul $[t_0, t]$, dat fiind faptul că a fost operațional la t_0.

Disponibilitatea la momentul *t* este o funție *A(t)* ce reprezintă probabilitatea ca sistemul să fie operațional la momentul *t*. Evaluarea lui *A(t)*, comparativ cu cea a lui *R(t)*, nu necesită o operare continuă a sistemului în intervalul $[t_0, t]$, deoarece un sistem poate traversa un ciclu *defectare-reparare (fail-repair cycle)* de un număr de ori în acest ciclu.

În literatură [DKT90], [BRL94], [BT93] se indică patru modele operaționale pentru analiza dependabilității. Fiecare model definește cerințe minimale pentru ca un sistem să fie considerat operațional.

La proiectarea SMP trebuiesc luate în considerare măsuri de asigurare a *dependabilității raportată la performanță (performance-related dependability)* în plus față de cele de asigurare doar a dependabilității [DKT90], [RA90], [RG87].

Modelarea cu acuratețe a RIN este extrem de complexă și, de aceea, diferite topologii de RIN pot necesita tehnici de modelare diferite.

5.1 ETAPELE PRELIMINARE ÎN DESCRIEREA MODELULUI

Un model este o reprezentare abstractă a sistemului și crearea lui presupune trei faze: *definirea*, *parametrizarea* și *evaluarea*.

Definirea

Se definește sistemul și obiectivul modelării. Acesta poate fi: *fiabilitatea*, *disponibilitatea* sau ambele în funcție de *performanță*. Identificarea sistemului ca fiind operațional se face în termeni de conectivitate disponibilă între noduri, conectate prin RIN.

Conectivitatea minimă pentru un sistem operațional se poate baza pe:

- o conexiune între o pereche anume de noduri de intrare şi noduri de ieşire (dependabilitate de tip terminal)
- o conexiune între un set cunoscut de noduri de intrare şi de ieşire (dependabilitate de tip multiterminal)
- conexiuni între un procentaj dat de noduri într-un sistem (dependabilitate bazată pe activitate)
- conexiuni între toate nodurile de intrare şi de ieşire (dependabilitate de tip rețea)

Parametrizarea

Parametrii de intrare includ *tipuri de componente*, *caracteristici de defectare* ca şi *distribuția defectului*, *rata de defectare* şi *de recuperare* şi *caracteristici de reparare* ca şi *distribuția timpului de reparare*, *rata de reparare* şi *probabilitatea unei reparări cu succes*.

Rata de defectare este λ_i, iar cea *de reparare* μ_i.

Au o distribuție exponențială cu rate constante.

Parametri de ieşire sunt: *fiabilitatea, disponibilitatea stării stabile sau tranzitorii, MTTF, factorul de ameliorare a siguranței în funcționare*. De asemenea, se pot obține măsuri ale dependabilității în raport cu performanța ca şi *performanța aşteptată la timpul t* sau *performanța acumulată într-un interval de timp*.

Evaluarea

Evaluarea modelului se face utilizând fie tehnici analitice, fie simularea.

Tehnicile analitice sunt:

- diagrame bloc ale fiabilității
- arbori de defect
- modele Markov
- rețele petri

În literatură [DKT90] sunt descrise în amănunt aceste tehnici, dar cea mai puternică tehnică pentru analiza de dependabilitate este *modelarea Markov*.

În cadrul *modelelor Markov* comportarea sistemului este modelată ca şi un lanț Markov cu stări finite şi în spațiul timpului continuu.

5.2 MODELE DE FIABILITATE

Există patru tipuri de tehnici de evaluare a fiabilității: *de tip terminal*, *de tip multiterminal*, *bazată pe sarcină* şi *de tip rețea* [DKT90].

5.2.1 Fiabilitatea de tip *terminal*

Reprezintă probabilitatea ce există cel puțin o cale între două terminale specifice. Căderea apare când un terminal se defectează sau când nu există nici o cale între două terminale.

Un terminal poate fi oricare dintre o gamă de entități.

De exemplu, fiecare terminal ar putea fi o pereche procesor-memorie, în care caz fiabilitatea de tip terminal ar putea determina fiabilitatea a două procese ce comunică. În mod alternativ, un terminal ar putea fi un procesor şi celălalt un modul de memorie.

Pentru evaluarea fiabilității de tip *terminal*, în primul rând se stabileşte siguranța componentelor individuale şi se enumeră conectivitatea lor relativă. Defectele de legătură între noduri sunt ignorate. Ele pot fi înglobate în defecțiunile de comutator sau de nod.

5.2.1.a Sisteme orientate pe *crossbar*

Fie SMP-MP orientat pe *crossbar* de dimensiuni $M \times N$.

Elementul de procesare PE_i se conectează la modulul de memorie MM_j prin intermediul comutatorului SW_{ij}.

Fiabilitatea terminalului este:

$$TR(t) = R_p(t) \cdot R_m(t)\ R_{sw}(t)$$

unde:

$R_p(t)$ – *fiabilitatea procesorului*

$R_m(t)$ – *fiabilitatea memoriei*

$R_{sw}(t)$ – *fiabilitatea comutatorului*

În cazul unei distribuții exponențiale cu ratele de defectare λ_p, respectiv λ_m se poate scrie:

$$R_p(t) = e^{-\lambda_p t};\ R_m(t) = e^{-\lambda_m t};\ R_{SW}(t) = e^{-\lambda_{SW} t}$$

5.2.1.b Sisteme bazate pe magistrală

Un sistem cu B magistrale asigură comunicarea dintre orice element de procesare şi orice modul de memorie atât timp cât cel puțin o magistrală este disponibilă.

Dacă *fiabilitatea fiecărei magistrale* este:

$$R_b(t) = e^{-\lambda_b t}$$

atunci

$$TR(t) = R_p(t) \cdot R_m(t) \cdot [1 - (1 - R_b(t))^B]$$

Ultimul termen reprezintă probabilitatea ca cel puțin o magistrală să fie operațională la timpul t.

5.2.1.c Sisteme orientate pe RIN-MN

Fie o RIN-MN cu o *singură cale* cu $n=\log_a N$ etaje de comutatoare.

Atunci fiabilitatea de terminal este:

$$TR(t) = R_p(t) \cdot R_m(t) \cdot R_{SW}^n(t)$$

În cazul RIN-MN cu *cale multiplă* pot exista una sau mai multe căi ce conectează un procesor la o memorie.

Analiza unui astfel de sistem se face utilizând o diagramă bloc a fiabilității pentru a reprezenta sistemul ca și o structură serie-paralel.

În ipoteza că într-un RIN-MN *cu cale multiplă* analizăm starea a două comutatoare A și B în raport cu o anumită cale, atunci pot exista două posibilități:

Ambele A și B sunt necesare pentru o conectare validă.

Comutatoarele sunt modelate ca și un sistem serie cu:

$$R_S(t) = R_A(t) \cdot R_B(t)$$

Dacă fiecare comutator poate asigura o conectare validă atunci ele constituie un sistem în paralel cu:

$$R_P(t) = [1 - (1 - R_A(t)) \cdot (1 - R_B(t))]$$

Metoda se complică în cazul sistemelor mari.

5.2.1.d Sisteme *hipercub*

Analiza sistemelor *hipercub* este similară cu cea a sistemelor orientate pe RIN-MN. Analiza unui n-cub cu 2^n noduri se face enumerând căile și prezentându-le ca și o diagramă bloc. Există $n!$ căi alternative non disjuncte de la fiecare sursă la fiecare destinație.

În fig. 5.1. se prezintă diagrama bloc a fiabilității pentru un *hipercub* unde se analizează fiabilitatea de tip terminal între nodurile 0 și 7.

Există șase căi posibile non disjuncte în nodurile 0 și 7.

În ipoteza că toate nodurile sunt la fel de sigure, $R_n(t) = R_i(t)$ pentru $0 \le i \le 7$ atunci

$$TR(t) = 6 \cdot R_n^4(t) - 6 \cdot R_n^5(t) - 3 \cdot R_n^6(t) + 6 \cdot R_n^7(t) - 2 \cdot R_n^8(t)$$

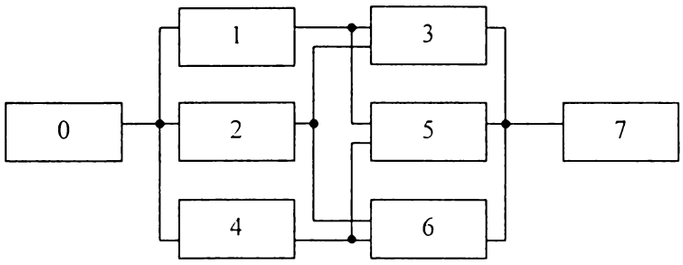


Fig. 5.1. O diagramă bloc a fiabilității reprezentând diverse căi între nodul 0 şi nodul 7

5.2.2 Fiabilitatea de tip *multiterminal*

Reprezintă o extensie logică a fiabilității de tip terminal.

În locul unei perechi specifice de terminale se ia în considerare un subset specific de terminale. Prin urmare, ea reprezintă probabilitatea ca un subset dat de terminale să fie complet conectat pentru comunicații simultane. De exemplu, fiabilitatea de tip *k–terminal*, reprezintă probabilitatea ca *k* noduri ($k>1$) să poată comunica prin RIN.

5.2.2.a Sisteme orientate pe *crossbar*

Un SMP *crossbar* de talie $M \times N$ necesită $x \cdot y$ comutatoare *crossbar* operaționale pentru a conecta *x* procesoare la *y* memorii. Fiabilitatea este:

$$MTR(t) = R_p^x(t) \cdot R_m^y(t) \cdot R_{SH}^{x \cdot y}(t) \text{ pentru } 1 \le x \le M \text{ şi } 1 \le y \le N$$

5.2.2.b Sisteme orientate pe magistrală

Pentru un SMP M X N X B fiabilitatea de tip *multiterminal* este aceeaşi ca şi cea de tip *monoterminal*, deoarece la un moment dat este nevoie de cel puțin o magistrală. Dacă pentru o conectare validă sunt necesare un set de *x* procesoare şi *y* memorii, atunci

$$MTR(t) = R_p^x(t) \cdot R_m^y(t) \cdot [1 - (1 - R_b(t))^B]$$

5.2.2.c Sisteme orientate pe RIN-MN

Fiabilitatea de tip *multiterminal* a unui sistem orientat pe RIN-MN *cu o singură cale* se raportează la *S*, numărul de comutatoare necesar pentru conectarea a *x* elemente de procesare la *y* memorii. Acest *S* depinde de localizarea procesoarelor solicitate, cât şi a memoriilor în cadrul RIN-MN. O dată ce *S* este cunoscut, fiabilitatea multiterminal devine:

$$MTR(t) = R_p^x(t) \cdot R_m^y(t) \cdot R_{SH}^S(t)$$

Analiza lui *MTR* în cazul RIN-MN *cu cale multiplă* este complexă şi necesită un studiu mai aprofundat.

5.2.2.d Sisteme *hipercub*

Un sistem este reprezentat ca şi un graf probabilistic $G(N,E)$ unde N şi E sunt seturile de noduri, respectiv de conexiuni.

Fiecărei componente i îi este asigurată o fiabilitate. Algoritmul de stabilire a fiabilității de tip *multiterminal* utilizează un *identificator de cale* de n bit pentru a reprezenta fiecare cale dintre o sursă şi destinație, unde n este numărul de componente ce se pot defecta.

Prin urmare, selecționând $n=N$ se consideră doar defecțiunile de nod, iar $n=E$ doar cele de conexiune. În situația $n=N+E$ se consideră ambele.

Un bit din identificatorul de cale este 1 dacă el este solicitat pentru conexiune şi x în cealaltă. Toți identificatorii de cale pentru conexiunile de intrare şi de ieşire sunt apoi prelucrați, utilizând tehnici din algebra booleană pentru a găsi o expresie simbolică finală, care este convertită într-o formă probabilistică pentru a determina fiabilitatea de tip *multiterminal*.

5.2.3 Fiabilitate *bazată pe sarcină*

Modelele de fiabilitate bazată pe sarcină (task-based) sunt mai generale decât fiabilitatea de terminal sau multiterminal, deoarece nu se specifică explicit nodurile sursă şi destinație. Un sistem rămâne operațional atât de mult timp cât sarcina poate fi executată cu resursele disponibile. Prin urmare, dacă o sarcină necesită cel puțin I elemente de procesare şi J module de memorie într-un SMP-MP, atunci sistemul este considerat operațional atâta timp cât sunt disponibile aceste resurse. Fiabilitatea bazată pe sarcină permite să se amelioreze toleranța la defect prin încorporarea *degradării grațioase (graceful degradation)* în sistem.

5.2.3.a Sisteme orientate pe *crossbar*

Probabilitatea ca i din x componente să funcționeze la momentul t este:

$$ {}_xC_i(t) = \binom{n}{i} R^i(t) \cdot (1 - R(t))^{n-i} \cdot C^{x-1} $$

unde:

$R(t)$ – fiabilitatea componentei $M \times N$

C – factorul de acoperire

Utilizând această notație, fiabilitatea unui SMP-MP *crossbar* de talie $M \times N$, de exemplu C.mmp ce necesită cel puțin I elemente de procesare şi J module de memorie este:

$$ TBR(t) = \sum_{i=I}^{M} \sum_{j=J}^{N} {}_MC_i(t) \, {}_NC_j(t) \cdot R_{x-bar}(t) \qquad (5.1.) $$

$R_{x-bar}(t)$ – fiabilitatea unei matrici *crossbar* de 16 × 16 din sistemul C.mmp.

$M = N = 16$ pentru sistemul C.mmp

În acest model nu este luată în calcul degradarea performanței comutatorului. Pentru a lua în calcul acest deziderat se construieşte un nou model astfel: se presupune că se doreşte un grup conectat de i elemente de procesare şi j module de memorie într-un sistem *crossbar* de $M \times N$. Deoarece un comutator este necesar pentru a conecta fiecare memorie, sunt necesare $i \cdot j$ elemente de comutare pentru conectarea cerută.

Fie $\gamma = i \cdot j$ şi $\varepsilon = M \cdot N - \gamma$, comutatoarele neutilizate. Există două modalităţi [DKT90] prin care un sistem *crossbar* $M \times N$ ar putea asigura conexiunea $i \times j$ solicitată:

- când exact i elemente de procesare şi j memorii sunt funcționale, şi elementele de comutare γ solicitate sunt disponibile să asigure conexiunile cerute. În acest prim caz (1) probabilitatea este $P_{i,j(1)}(t)$.
- Când mai mult decât i procesoare sau j memorii sunt funcționale, dar doar γ comutatoare sunt disponibile pentru a realiza o conexiune $i \times j$. În acest caz (2) probabilitatea este $P_{i,j(2)}(t)$

Se poate scrie:

$$P_{i,j(1)}(t) = {}_{M}C_{i}(t) \cdot {}_{N}C_{j}(t) \cdot R_{SW}^{\gamma}(t) \qquad (5.2.a.)$$

De notat că ε – condiția de comutatoare suplimentare – este irelevantă când acoperirea C este perfectă în ceea ce priveşte comutatoarele. Acoperirea imperfectă în comutatoare se face aproximând al doilea termen $P_{i,j(2)}(t)$ ca fiind:

$$P_{i,j(2)}(t) = \sum_{x}^{M} \sum_{y}^{N} {}_{M}C_{x}(t) \cdot {}_{N}C_{y}(t) \cdot R_{SW}^{\gamma}(t) \cdot (1 - R_{SW}(t))^{\varepsilon} \qquad (5.2.b.)$$

unde:

$x > i \qquad y > j$

Combinând cei doi termeni se defineşte *TBR(t)* astfel:

$$TBR(t) = \sum_{i=1}^{M} \sum_{j=1}^{N} P_{i,j(1)}(t) + P_{i,j(2)}(t) \qquad (5.2.c.)$$

5.2.3.b Sisteme orientate pe magistrală

La aceste sisteme, analiza este simplă deoarece există întotdeauna o cale între orice procesor şi orice memorie, atâta timp cât cel puțin o magistrală este disponibilă. Contribuția structurii de

magistrală este $1-(1-R_b(t))^B]$. Intruducând acest termen în ecuația (5.1) în locul lui $R_{x-bar}(t)$ rezultă:

$$TBR(t)=\sum_{i=1}^{M}\sum_{j=1}^{N} {}_M C_i(t)\cdot {}_N C_j(t)\cdot[1-(1-R_b(t))^B] \qquad (5.2.d.)$$

ce reprezintă fiabilitatea bazată pe sarcină pentru SMP-MP cu magistrală multiplă.

5.2.3.c Sisteme orientate pe RIN-MN

Analiza fiabilității *bazată pe sarcină* pentru această clasă de sisteme este extrem de dificilă, în principal datorită complexității RIN-MN. Un model Markov cu un singur nivel (incluzând procesoare, memorii, comutatoare) este aproape imposibil de construit datorită numărului mare de stări. De aceea, se utilizează tehnici ca și *simularea* și *descompunerea de sistem* în analiza fiabilității bazată pe sarcină.

Descompunerea ierarhică este o tehnică specială pentru analiza sistemelor orientate pe RIN-MN. Este descrisă în amănunt în [DKT90], [DTB90] și [DTB93]. Această schemă presupune descompunerea unui sistem mai mare în subsisteme mai mici pentru o analiză mai ușoară.

În literatură [DKT90] se citează exemplul unui sistem bazat pe o rețea *Butterfly* ce utilizează comutatoare *crossbar* 4 × 4 ca și în fig. 5.2.

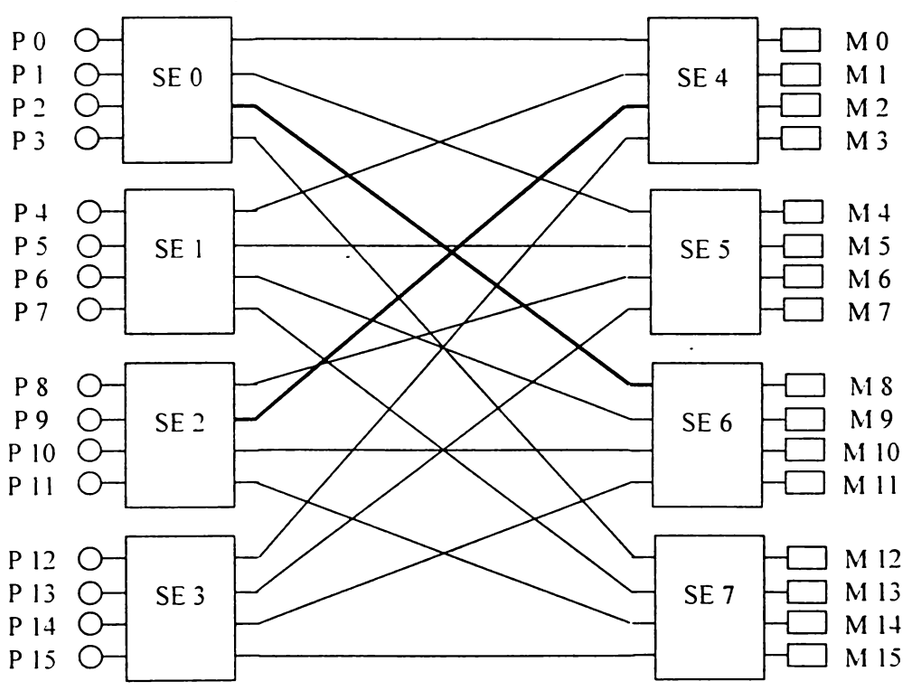


Fig. 5.2. Un SMP-MP cu RIN-MP *butterfly* de 16 × 16

Numărul de noduri din sistem este 4^n și fiecare nod constă dintr-un procesor și o memorie. Este un sistem $4^n \times 4^n$. Configurația $4^n \times 4^n$ poate fi partiționată pe 4 subsisteme $4^{n-1} \times 4^{n-1}$, fără a

perturba conexiunile. Prin urmare, fiabilitatea unui SMP-MP de rang 4^n depinde de 4 subsisteme 4^{n-1} şi de configuraţia de conectare dintre ele. Analiza porneşte cu expresiile probabilităţilor pentru o configuraţie de patru elemente de procesare, patru memorii şi două comutatoare.

Fie $P_4(i,j)^{(t)}$ probabilitatea ca i elemente de procesare şi j module de memorie ($0 \le i, j \le 4$) să lucreze la momentul t. Analiza are drept scop să găsească probabilitatea ca I elemente de procesare şi J module de memorie să lucreze într-un SMP-MP de talie 16 × 16. Aceasta se exprimă folosind patru grupe de subsisteme 4 × 4 astfel:

$$P_{16(I,J)}(t) = P_{4(i_0,j_0)}(t) \cdot P_{4(i_1,j_1)}(t) \cdot P_{4(i_2,j_2)}(t) \cdot P_{4(i_3,j_3)}(t)$$

Astfel ca: $(i_0 + i_1 + i_2 + i_3) = I$ şi $(j_0 + j_1 + j_2 + j_3) = J$

Toate combinaţiile posibile ale celor I procesoare şi ale celor J memorii sunt generate pentru expresiile de mai sus.

În final, fiabilitatea sistemului de 16 noduri cu cel puţin I procesoare şi J memorii active este:

$$TBR(t) = \sum_{i=I}^{16} \sum_{j=J}^{16} P_{16(i,j)}(t) \tag{5.3}$$

Fiabilitatea unui sistem cu 64 de noduri se obţine din configuraţii de 16 noduri, utilizând tehnica de mai sus. Un sistem cu 64 noduri are trei niveluri cu câte 16 comutatoare pe nivel.

Primul şi ultimul nivel sunt incluse în analiza de bază 4 × 4. Prin urmare, se poate determina numărul cerut de comutatoare din nivelul mediu, pentru orice grup activ utilizând un calcul de aproximare. Fiabilitatea acestor comutatoare este multiplicată cu $P_{64(i,j)}(t)$ pentru a garanta că comutatoarele din nivelul median asigură $i \times j$ conexiuni pentru cele două cazuri posibile discutate la analiza sistemelor crossbar.

5.2.3.d Sisteme *hipercub*

Modelul de fiabilitate bazat pe sarcină dezvoltat este utilizat pentru a găsi probabilitatea ca un n-cub să aibă cel puţin I noduri active conectate. Se utilizează tehnica descompunerii şi se poate lucra cu orice nivel de degradare a sistemului.

Deoarece un n-cub poate fi divizat în două $(n-1)$ – cuburi, această schemă descompune recursiv un cub mai mare în cuburi mai mici, până când dimensiunea cubului obţinut este suficient de simplă pentru model. Cel mai mic cub este denumit „modelul de bază" al analizei.

Un 2-cub (patru noduri) sau un 3-cub (opt noduri) pot constitui modelul de bază. Apoi, în mod recursiv, se obţine fiabilitatea unor cuburi de dimensiuni mai mari.

De exemplu, pentru a calcula probabilitatea a j noduri conectate într-un n-cub, îl partajăm în două $(n-1)$-cuburi. Există două cazuri în care un n-cub va avea j noduri conectate.

În primul caz, exact j noduri sunt active în n-cub.

În al doilea caz, mai mult de j noduri, de exemplu x, sunt active, dar conectivitatea este doar j.

Se poate diviza j sau n noduri între două $(n-1)$-cuburi ca şi k şi $j-k$ sau $x-k$, respectiv. Cinci distribuţii ale nodurilor active pot fi analizate pornind de la aceste două cazuri pentru a obţine $P_j(t)$, probabilitatea ca să existe j noduri active conectate în două $(n-1)$-cuburi.

Fiabilitatea n-cubului este:

$$TBR(t) = \sum_{j=1}^{2^n} P_j(t) \tag{5.4}$$

Această abordare poate fi extinsă pentru a găsi un subcub de dimensiune m, unde $m \leq n$, într-un n-cub, presupunând că un program în sistem necesită o alocare cubică în loc de I noduri conectate.

5.2.4 Fiabilitatea de tip *reţea*

Fiabilitatea de tip reţea (network reliability) este probabilitatea de-a menţine o reţea cu acces total. Dacă se consideră două reţele cu o singură cale, probabilitatea reţelei reprezintă probabilitatea ca toate procesoarele şi memoriile sistemului să fie conectate. O reţea *cu cale multiplă* poate avea acces total chiar dacă componentele reţelei se defectează.

Analiza RIN *cu cale multiplă* este extrem de dificilă. În ultimă instanţă, acest tip de analiză reprezintă un caz special a fiabilităţii bazate pe sarcină, unde sistemul nu permite degradarea resurselor de calcul.

5.3 MODELE DE DISPONIBILITATE

În cadrul evaluării fiabilităţii nu se iau în considerare activităţile de reparare a sistemului. În sistemele cu degradare graţioasă, repararea componentelor ce s-au defectat contribuie la creşterea duratei operaţionale a sistemului şi oferă o disponibilitate mai mare.

Evaluarea disponibilităţii unui sistem complex cu o singură facilitate de reparare este mult mai complexă decât modelarea de fiabilitate. Această complexitate derivă din luarea în considerare a doi factori.

În primul rând, trebuie generat un lanţ Markov al sistemului pentru a analiza disponibilitatea sa. Acest proces poate fi complex pentru un sistem mare. Numărul de stări din model poate, de asemenea, să fie prea mare pentru a fi uşor manipulat.

În al doilea rând, găsirea unei soluţii pentru disponibilitatea în cazul stărilor tranzitorii sau a celor stabile este dificilă. Din această cauză, se utilizează tehnici numerice pentru a calcula disponibilitatea.

5.3.1 Sisteme orientate pe *crossbar*

În literatură [DKT90] se prezintă un lanț Markov dezvoltat pentru un sistem *crossbar* (C.mmp) de dimensiuni 16 ×16. Întreaga rețea este tratată ca şi un singur element. Modelul este simplu deoarece o defecțiune în *crossbar* se transformă într-o defecțiune de sistem. Modelul este un lanț Markov bidimensional, unde o dimensiune reprezintă defecțiunea procesorului, iar a doua reprezintă defecțiunea de memorie.

5.3.2 Sisteme orientate pe magistrală

SMP-MP cu magistrală multiplă se modelează ca şi un lanț Markov tri-dimensional, unde fiecare dimensiune reprezintă degradarea unui tip de componentă - procesor, memorie şi magistrală. Presupunând că procesoarele, memoriile şi magistralelel au fiecare facilități de reparare, lanțul Markov poate include rețele de reparare pentru a ilustra un sistem reparabil.

5.3.3 Sisteme orientate pe RIN-MN

Modelul de disponibilitate pentru SMP-MP orientate pe RIN-MN propus în [DTB90] poate manipula sisteme mari şi orice nivel de degradare. Această schemă utilizează o abordare a descompunerii sistemului pentru a evita generarea unui lanț Markov mononivel detaliat. Acest model presupune o comportare independentă la defecțiune şi la reparare a procesoarelor, memoriilor şi comutatoarelor şi, din punct de vedere conceptual, împarte sistemul în trei subsisteme conectate în serie. Se pot dezvolta lanțuri Markov independente pentru cele trei subsisteme.

Fiecare diagramă Markov a subsistemului este un simplu lanț unidimensional. În fig. 5.3. se prezintă un lanț Markov pentru un subsistem procesor ce necesită cel puțin *I* elemente de procesare, λ_p şi μ_p reprezentând *rata de cădere*, respectiv *rata de reparare ale procesorului*.

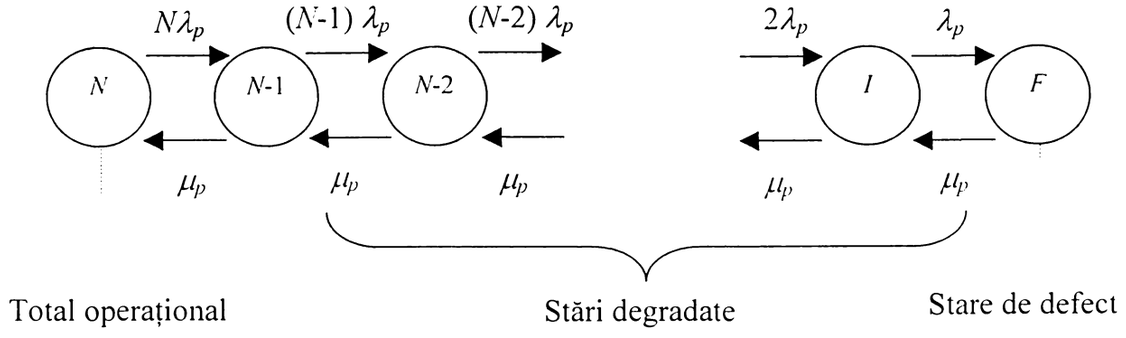


Fig. 5.3. Un lanț Markov pentru un subsistem procesor

Sistemul evoluează de la starea i la starea vecină i-1 cu o rată de cădere $i \cdot \lambda_p$ sau la starea i+1 cu o rată de reparare μ_p. Din raţiuni de simplificare a analizei se omit acoperirea imperfectă şi parametrii de reparare. Un astfel de model poate fi rezolvat prin tehnici de analiză numerică pentru a găsi probabilitatea a i procesoare active $P_i(t)$ pentru $I \leq i \leq N$.

Modelul Markov pentru subsistemul de memorie este aproape identic cu excepţia că sistemul tolerează până la J defectări. *Ratele de defectare ale memoriei* şi *ratele de reparare* sunt λ_m şi μ_m. Rezolvarea acestui model oferă $P_j(t)$, probabilitatea ca să existe j module de memorie active la orice timp t.

Lanţul Markov pentru elementele de comutare se bazează pe identificarea numărului de comutatoare necesare pentru o conexiune $i \times j$. De exemplu, calculul probabilităţii ca opt procesoare şi opt memorii să fie conectate într-un sistem 16 × 16 ca în fig. 5.2, se obţine parţionând sisteme în patru grupe, fiecare grup cuprinzând patru procesoare şi patru memorii. Se poate distribui cele opt procesoare şi opt memorii în două, trei sau patru grupuri.

Există două cazuri ce ar putea apare într-o conexiune 8 × 8. În primul caz, exact opt procesoare şi opt memorii sunt active şi sunt conectate printr-un număr necesar de comutatoare. Celelalte comutatoare nu afectează conectivitatea. De exemplu, dacă componentele active sunt distribuite în două grupuri, trebuie să funcţioneze patru comutatoare pentru ca această conexiune să fie validă. Aceste două comutatoare aparţin la două grupuri de lucru şi pot fi separate de celelalte patru comutatoare. Conectivitatea este 8 × 8, indiferent de care alte patru comutatoare sunt active.

Lanţul Markov din fig. 5.4 prezintă acest caz, cu λ_{SE} şi μ_{SE} reprezentând *ratele de defectare*, respectiv *ratele de reparare ale comutatoarelor*.

În fig. 5.4, *(x, y)* reprezintă comutatoarele *(x)* cerute şi comutatoarele suplimentare *(y)*, astfel că $x+y$ este numărul total de comutatoare din RIN-MN. Tranziţiile orizontale arată că sistemul cade (starea F_i , unde $0 \leq i \leq y$) când oricare dintre cele x comutatoare solicitate cade. În mod consecvent, fiabilitatea RIN-MN este:

$$P_{SE(1)}(t) = \sum_{i=0}^{x} P_{x,i}(t)$$

Unde *(i)* indică primul caz, adică cazul în care exact i procesoare şi j memorii sunt active. În fig. 5.4 cu $x=y=4$, $P_{SE(1)}(t) = (P_{4,4}(t) + P_{4,3}(t) + P_{4,2}(t) + P_{4,1}(t) + P_{4,0}(t))$.

Probabilitatea conexiunii $i \times j$ pentru crossbar este:

$$P_{i,j(1)}(t) = P_i(t) \cdot P_j(t) \cdot P_{SE(1)}(t)$$

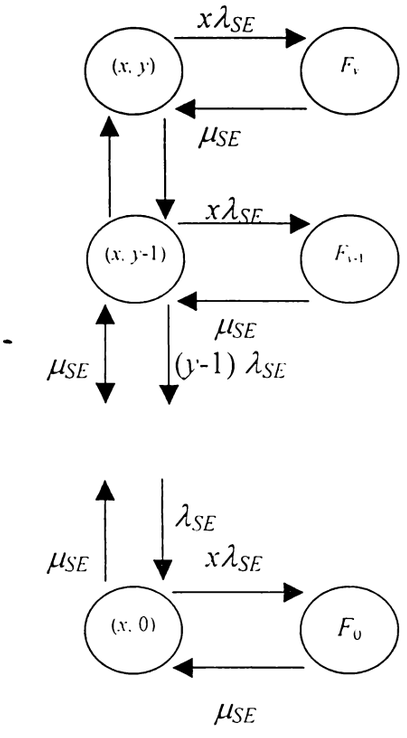


Fig. 5.4. Un lanț Markov reprezentând o RIN-MN

În al doilea caz, mai mult decât i procesoare, fie ele α, sau mai mult decât j memorii, fie ele β, pot lucra în timp ce conectivitatea rămâne $i \times j$. De exemplu, mai mult decât opt procesoare şi de opt memorii pot lucra în fig. 5.2, cu conectivitatea rămânând 8 × 8, atâta timp cât doar comutatoarele cerute pentru opt procesoare şi opt memorii sunt active. Dacă conexiunea 8 × 8 este distribuită în două grupuri, sunt necesare patru comutatoare. Celelalte patru comutatoare ar putea cădea (făcând y=0) astfel că procesoarele suplimentare şi memoriile din restul sistemului să nu contribuie la fiabilitate. Prin urmare, $P_{SE(2)}(t) = P_{x,0}(t)$ reprezintă probabilitatea ca doar x comutatoare să fie active. Se obține $P_{i,j(2)}(t)$ prin multiplicarea lui $P_{x,0}(t)$ cu probabilitățile ca α procesoare şi β memorii să fie active. Aceste probabilități sunt $P_\alpha(t)$ şi $P_\beta(t)$. Valorile pentru $P_\alpha(t)$ şi $P_\beta(t)$ se obțin rezolvând lanțurile Markov pentru procesor şi memorie. Se determină disponibilitatea sistemului adăugând toate probabilitățile configurației active exprimate de $TBR(t)$.

5.4 MODELE DE DEPENDABILITATE RAPORTATĂ LA PERFORMANȚĂ

Măsura dependabilității raportată la performanță combină aspecte ale performanței şi ale dependabilității. Este utilă analiza din această perspectivă, mai ales pentru sistemele cu degradare grațioasă.

Utilizând notația unui model lanț Markov în spațiu continuu, un SMP generic se poate reprezenta ca și un proces cu stări finite $Z(t)$, cu $t \geq 0$ și cu spațiul de stări 0, 1, ..., n. Starea 0 reprezintă starea defectă a sistemului, iar stările 1 până la n, configurațiile de lucru. Lanțul Markov ar putea fi *ciclic* sau *aciclic*, depinzând de faptul că sistemul este *reparabil* sau *nu*.

Rezolvând lanțul Markov se poate obține un vector coloană $P(t) = p_1(t); p_2(t), ..., p_n(t)$ de unde se obține probabilitatea ca sistemul să fie în diferite stări de lucru la momentul t. De exemplu, $p_i(t)$ este probabilitatea ca sistemul să fie în starea i la momentul t.

Se asociază stării i o *rată de recompensă* r_i pentru $1 \leq i \leq n$. Dacă sistemul rămâne în starea i pentru o durată τ, atunci recompensa acumulată în starea i este $\tau\, r_i$. *Funcția de recompensă acumulată* $Y(t)$ în timpul t este:

$$Y(t) = \int_0^t x(\tau) \cdot d\tau$$

unde $x(\tau)$ este rata de recompensă la momentul τ.

La rândul său, $x(\tau) = r\, Z(\tau)$ unde r este un *vector de recompensă* asociat. Se poate defini r cu parametrii ca și: *lățime de bandă, rată de execuție a instrucției* sau alte metrici de performanță relevante.

Studiile asupra acestui tip de dependabilitate au produs un număr interesant de măsuri. Astfel, pentru SMP degradabile, în literatură [DKT90] se citează: *fiabilitate a calculului* și *disponibilitate a calculului*. Utilizând conceptul de *probabilitate a unei stări de lucru* $p_i(t)$ cât și a *ratei de recompensă asociate* r_i se poate exprima *puterea de calcul la timpul t* ca fiind:

$$E[x(t)] = \sum_{i=1}^{n} p_i(t) \cdot r_i \qquad (5.5)$$

În analiza dependabilității raportată la performanța SMP-MP se introduce un parametru denumit *disponibilitatea lățimii de bandă (band-width availability), BA(t)*. Ea este definită ca fiind lățimea de bandă așteptată a unui SMP la momentul t. Lățimea de bandă este o măsură a performanței pentru sistemele sincrone, definit ca și numărul mediu de perechi procesor-memorie conectate într-un ciclu.

Utilizând o expresie de fiabilitate bazată pe sarcină, *disponibilitatea lățimii de bandă* ese:

$$BA(t) = \sum_{i=1}^{M} \sum_{j=J}^{N} P_{i,j}(t) \cdot BW_{i,j} \qquad (5.6)$$

unde :

$P_{i,j}(t)$ – probabilitatea ca sistemul să lucreze ca i procesoare și j memorii

$BW_{i,j}$ – lățimea de bandă cu factorul de recompensă r_i

Performanța acumulată la momentul t este $E[Y(t)]$ care reprezintă cantitatea totală de activitate utilă, exprimată ca şi valoare medie, pe care un SMP o poate efectua integrând ecuația 5.5 pe intervalul de timp observat:

$$E[Y(t)] = \sum_{i=1}^{n} r_i \cdot \int_0 p_i(\tau) \cdot d\tau \tag{5.7}$$

Un alt parametru introdus este *performabilitatea* ce reprezintă funcția de distribuție a probabilității performanței de sistem acumulată. Ea oferă probabilitatea ca un sistem să efectueze o cantitate dată de activități x în timpul t. Utilizând funcția $Y(t)$, performabilitatea este:

$$y(x,t) = P[Y(t) \geq x] \tag{5.8}$$

Calculul expresiei este complex. Pentru un sistem cu n stări active, fiecare distribuție a recompensei stării implică o integrare.

Evaluarea performabilității la SMP a fost limitată deoarece modelele de dependabilitate nu au fost suficient dezvoltate pentru unele circuite şi determinarea distribuției este complexă pentru un număr mare de stări.

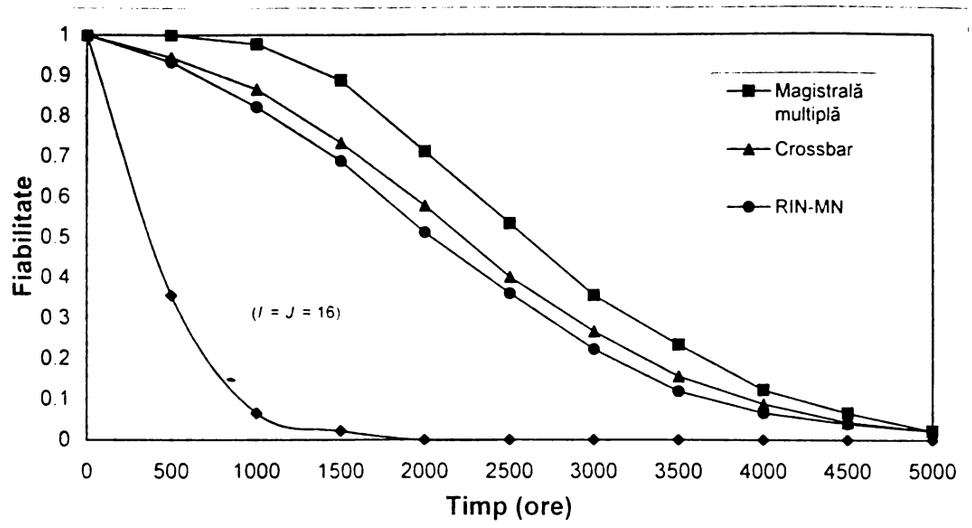
Cele mai multe dintre SMP-MP *crossbar*, *magistrală multiplă*, *RIN-MN* utilizează *lățimea de bandă BW* ca şi indicator de performanță.

Compararea dependabilității la SMP de talie mică se face luând în considerare conexiunile *crossbar*, *magistrala partajată* şi RIN-MN.

Fig. 5.5 prezintă fiabilitatea sistemului pentru un SMP-MP de dimensiuni 16 × 16 în trei ipostaze derivate din tipurile de conexiuni. Se presupune că au aceeaşi rată de defectare. Toate cele trei sisteme operează apróximativ cu aceeaşi fiabilitate când nu se permite degradarea.

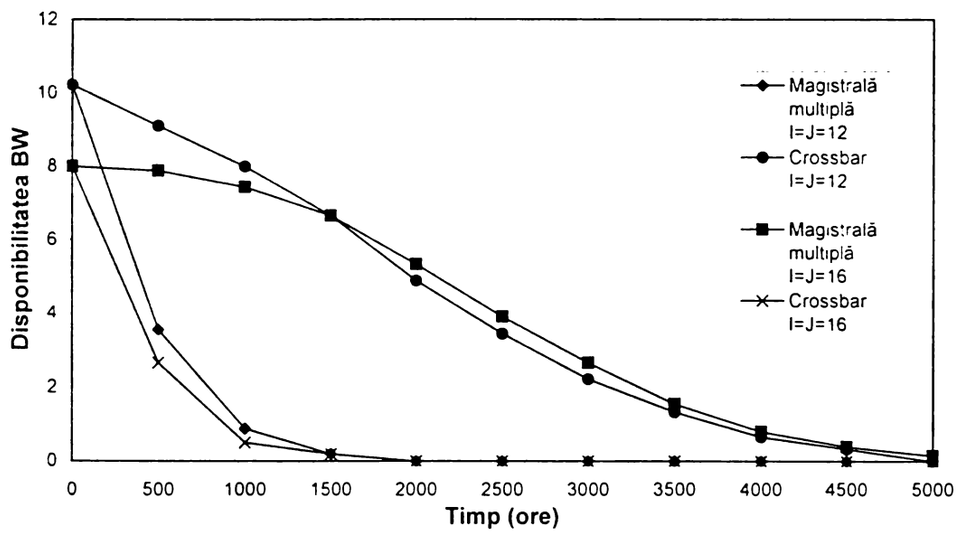
Oricum, ele au în mod distinct diferite fiabilități, dacă se permite o degradare de 25% *(I=J=12)*. SMP-MP *cu magistrală partajată* operează cel mai bine deoarece are opt căi alternative, cele mai multe, decât celelalte sisteme. Diferența în fiabilitate între *crossbar* şi RIN-MN devine notabilă când există degradare de sistem datorită diferitelor capabilități de conectare. Un defect de comutator în RIN *Butterfly* deconectează patru perechi procesor-memorie, în timp ce în *crossbar* doar o pereche este deconectată. Pe de altă parte, complexitatea RIN-MN este mai mică deoarece există mai puține comutatoare. Prin urmare, se poate ameliora fiabilitatea RIN-MN adăugând niveluri suplimentare de comutatoare.

Cuantificarea *dependabilității raportată la performanță* este prezentată în fig. 5.6 iar în fig. 5.7 se prezintă *distribuția mediată în timp a lățimii de bandă* pentru un sistem de tip C.mmp.



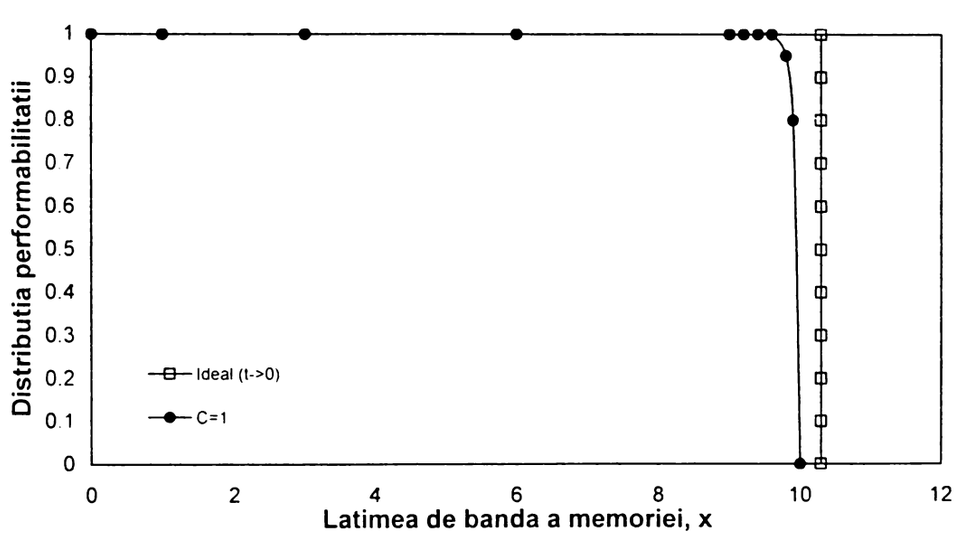
λ_p=0.0001(1· 10^{-4})
λ_m=0.0001(1· 10^{-4})
λ_{SE}=625· 10^{-9}
λ_b=0.0002(2· 10^{-5})

Fig. 5.5 Fiabilitatea unui SMP - MP în funcție de natura RIN



λ_p=1· 10^{-4}
λ_m=1· 10^{-4}
λ_{SE}=625· 10^{-9}
λ_b=2· 10^{-5}
c=1.0

Fig. 5.6 Disponibilitatea unui SMP - MP în funcție de natura RIN



λ_p=689· 10^{-7}
λ_m=2241· 10^{-7}
λ_{cb}=2034· 10^{-9}
c=1.0

Fig. 5.7 Distribuția mediată a performabilității

5.5 MODELAREA DISPONIBILITĂȚII UTILIZÎND DESCOMPUNEREA DE SISTEM PENTRU SMP-MP, ORIENTATE PE RIN-MN

În paragraful § 5.3.3. s-a prezentat succint modul de abordare a construcției unui model de dependabilitate pentru un SMP-MP orientat pe RIN-MN.

În acest paragraf vom detalia *descompunere de sistem* ca si tehnică de modelare a dependabilității unui sistem complex fără a construi un *lanț Markov mononivel* (*single-level Markov Chain* - MK).

In literatură [DTB 93] se prezintă metoda *descompunerii de sistem* în cadrul clasei de modele de disponibilitate bazate pe sarcină. În cadrul unor astfel de modele, un sistem este considerat operațional atâta timp cât cerințele sarcinii sunt îndeplinite. Modelele celor două simple MK ce pot fi dezvoltate - subsistemul procesor-memorie şi subsistemul RIN-MN - se rezolvă apoi prin pachete de programe specializate. In literatură [DKT 90], [DTB 93] sunt citate pachete de programe **H**ybrid **A**utomated **R**eliability **P**redictor- **HARP**, **C**omputer-**A**ided **R**eliability **E**stimation III- **Care III** şi **S**ymbolic **H**ierarchical **A**utomated **R**eliability and **P**erformance **E**valuation- **SHARPE**.

În cadrul modelului se calculează probabilitățile ca i elemente de procesare PE şi j module de memorie MM, $P_i(t)$ respectiv $P_j(t)$, să fie operaționale în orice moment t prin găsirea numărului de comutatoare cerute pentru conectarea celor i procesoare în cele j memorii.

5.5.1 Descrierea sistemului

Sistemul analizat este un SMP-MP de talie $N \times N$ în care N elemente de procesare PE sunt conectate cu N module de memorie MM, prin intermediul unei RIN-MN de tip *Butterfly* având $n(n=\log_4 N)$ niveluri de comutatoare-SE- de dimensiuni 4×4. Pe un nivel sunt plasate $N/4$ SE.

Particularizarea unui astfel de sistem pentru $N=16$ este prezentată în fig. 5.2.

Se presupune că un element de procesare PE_i şi modulul său de memorie corespunzător MM_i sunt localizate pe o singură placă denumită *nodul i*. Prin urmare, sistemul are o configurație de N noduri. O referire la memoria locală nu trebuie să parcurgă RIN-MN. O referire la o memorie externă modulului necesită două treceri prin RIN-MN. În prima trecere, procesorul solicitant transmite cererea către memoria externă modulului prin RIN-MN. Răspunsul este transmis de memorie către procesor la a doua trecere. De exemplu, o parcurgere completă a circuitului de la PE2 la MM8 este reprezentată prin linie groasă în fig. 5.2. În acest tip de schemă de comunicare

sunt necesare patru SE intr-un sistem (16×16) în raport cu două SE necesare în cadrul unui protocol cu o singură trecere.

În construcția modelului se fac următoarele ipoteze simplificatoare:

1. Componentele subsistemului sunt toate independente, omogene şi au distribuții de defectare exponențiale.

   Acestea sunt $\lambda_p, \lambda_m, \lambda_{sc}$ şi reprezintă *ratele de defectare* ale PE, MM şi SE.

2. Există o singură facilitate de reparare pentru fiecare tip de componentă cu o distribuție exponențială a timpului de reparare.

   Ratele de reparare sunt μ_p, μ_m, μ_{sc}.

Este interesant de remarcat că se admite repararea on-line şi a comutatoarelor SE. Această abordare presupune existența unor algoritmi de detectare a defectului pentru RIN-MN care pot localiza defectul în comutatoare. Izolarea şi remedierea defectului sunt posibile fără oprirea activității. De asemenea, se admite că toleranța la defect se manifestă şi prin degradarea grațioasă. Datorită complexității RIN-MN, pentru a se putea gestiona defectele din SE, acestea sunt prevăzute cu memorii cache incluse.

5.5.2 Tehnica de modelare

Analiza se bazează pe *metoda descompunerii* în care SMP este divizat în trei subsisteme ca şi în fig. 5.8. Fiecare sistem este analizat independent. La un moment dat pot fi conectate *I* procesoare şi *J* memorii pentru satisfacerea cerințelor impuse de sarcina la nivel de sistem.

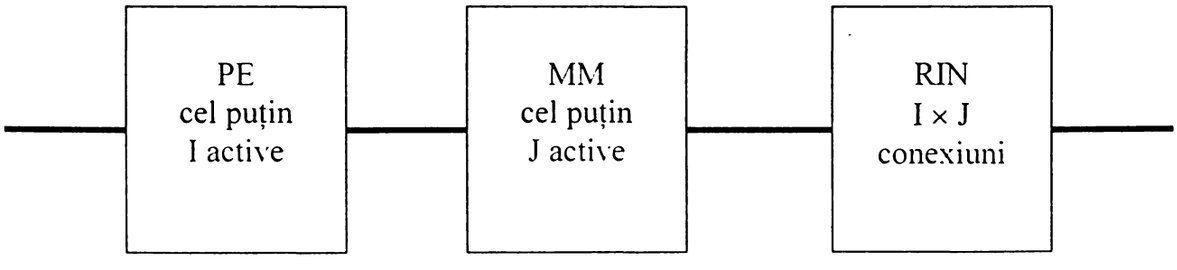


Fig. 5.8. Descompunerea unui sistem

A. Subsistemul Procesor/Memorie (SPM)

Configurația inițială a SMP are N procesoare şi cel puțin I dintre acestea trebuie să fie active pentru ca sistemul să fie operațional. MK-ul pentru SPM este prezentat în fig. 5.9.

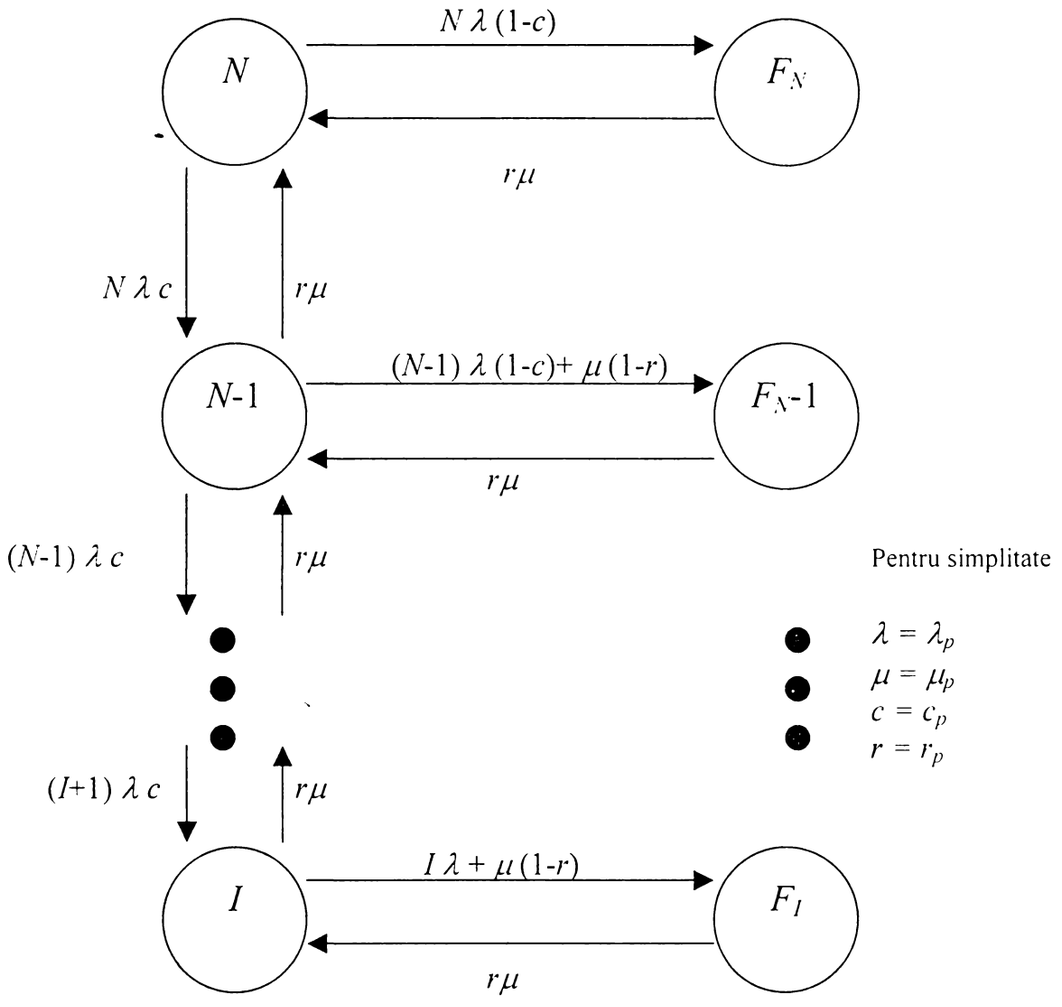


Fig. 5.9. Lanț Markov pentru subsistemul SPM

Modelul include o acoperire imperfectă şi o reparare imprecisă presupuse ca putând apare.

Notând:

C_p –acoperirea unui procesor

R_p –factorul de reparare cu succes

se observă că SPM evoluează de la o stare de lucru i, pentru $I \le i \le N$ spre o stare defectă F_i fie unei acoperiri imperfecte ($1-c_p$) sau datorită unei reparări imprecise ($1-r_p$). Dacă acoperirea este integrală şi procesul de reparare este perfect, atunci MK se reduce la un ciclu de reparare.

Soluția pentru MK prezentat se obține utilizând un pachet HARP ce furnizează probabilitatea ca *i* procesoare să fie active în sistem la momentul *t*. Aceasta este $P_i(t)$ pentru $I \le i \le N$.

MK pentru subsistemul memorie este asemănător cu cel procesor, cu mențiunea că sistemul tolerează până la N - J defecte. *Rata de defectare, rata de reparare, factorul de acoperire* și *factorul de reparare cu succes* sunt $\lambda_m, \mu_m, c_m, r_m$. Rezolvând MK-ul memoriei, se obține $P_j(t)$, probabilitatea ca să existe *j* memorii active la momentul *t*, pentru $J \le j \le N$.

B. Subsistemul RIN-MN

O RIN-MN degradabilă poate asigura conectarea dintre *i* PE și *j* MM. Astfel, modelarea unui RIN-MN poate garanta că numărul solicitat de SE trebuie să funcționeze în scopul satisfacerii conexiunii. Găsirea SE potrivite pentru conectare este dificilă deoarece PE și MM ce se conectează sunt distribuite de o manieră arbitrară. Modelul exact pentru RIN-MN este cunoscut ca fiind o problemă cu NP-complet. Aici se dezvoltă însa un model aproximativ.

Un grup conectat de *i* procesoare și *j* memorii, adică $i \times j$ conexiuni, poate fi obținut în două moduri.

În primul caz, *i* procesoare și *j* memorii sunt active și cel puțin numărul cerut de SE este disponibil pentru asigurarea conectării.

În al doilea caz, mai mult de *i* procesoare și *j* memorii sunt active, dar conexiunea este încă ($i \times j$). Acest caz apare când SE active sunt suficiente să conecteze doar *i* procesoare și *j* memorii.

O RIN-MN de dimensiuni ($N \times N$) cu 4 × 4 SE are un număr total de $N/4 \cdot \log_4 N$ comutatoare. Pentru o distribuție dată ($i \times j$), fie *x* numărul de SE-uri cerut pentru conectare și $y = N/4 \cdot \log_4 N - x$ comutatoarele SE adiționale. Starea acestor SE adiționale nu afectează conectivitatea. Un MK a unei RIN-MN cu până la *y* defectări poate încă furniza conectarea cerută pentru un sistem ($i \times j$). În al doilea caz, când mai mult decât ($i \times j$) elemente sunt active, dacă se utilizează probabilitatea ca doar *x* SE să fie perfecte, talia sistemului încă va fi ($i \times j$).

Un al doilea nivel de partiționare al SMP este utilizat pentru a-l găsi pe *x*, numărul cerut de SE. Talia sistemului este uzual dată de $(4^t \times 4^t)$ deoarece RIN-MN este construită din SE de talie 4 × 4. O configurație de $(4^t \times 4^t)$ poate fi partiționată în patru subsisteme $(4^{t-1} \times 4^{t-1})$ fără a perturba conexiunea.

Metodologia de modelare nu este limitată la RIN-MN cu SE 4 × 4. Tehnica de descompunere poate fi aplicată la orice nivel $\log_a N$ proiectat cu SE $a \times a$. Rețeaua în acest caz ar putea fi

divizată logic în *a* grupuri. Precizia rezultatelor va depinde de valorile *x* şi *y*. Este foarte dificil să se producă expresii generice pentru a calcula aceste valori comutatoare 4 × 4.

C. Configuraţia cu 16 noduri

Cele 16 PE sunt partiţionate în patru grupe unde fiecare grup are patru PE şi comutatoarele corespunzătoare. Aceste grupuri sunt notate (PG_0, PG_1, PG_2, PG_3). Similar, cele 16 MM sunt partiţionate în (MG_0, MG_1, MG_2, MG_3). Fie N_p numărul minim de grupuri de procesoare cerute pentru *i* PE active $1 \le i \le N$.

Atunci:

$$N_p = \left\lceil \frac{i}{N div 4} \right\rceil \tag{5.9.a}$$

unde: $\lceil\ \rceil$ reprezintă limitarea superioară

N = talia sistemului

Similar, pentru N_m, numărul minim al grupurilor de memorie necesare pentru *j* MM active ($1 \le j \le N$):

$$N_m = \left\lceil \frac{j}{N div 4} \right\rceil \tag{5.9.b}$$

Cele *i* PE pot fi distribuite în *p* grupuri de procesare pentru $N_p \le p \le 4$ şi *j* MM pot fi distribuite peste *m* grupuri de memorie, $N_m \le m \le 4$. Numărul total de căi din care se alege *p* şi *m* este $\binom{4}{p}\binom{4}{m}$. Fiecare din aceste distribuţii necesită SE specifice pentru conectare.

De exemplu, pentru $p = m = 2$, cele două grupuri de procesoare şi două grupuri de memorii pot fi distribuite în 36 de căi.

Dacă două grupuri de procesare sunt conectate la două grupuri corespunzătoare de memorii, având acelaşi număr de grup, $\{PG_0, PG_1\} \leftrightarrow \{MG_0, MG_1\}$, atunci aceasta este o conexiune $(2 \leftrightarrow 2)$. Sunt necesare două SE de intrare şi două SE de ieşire pentru această conexiune. Acest caz reprezintă activitatea lui SE 0, SE 1, SE 4 şi SE 5 din fig. 5.2.

Dacă în loc de selectarea grupurilor de memorie 0 şi 1 se selectează 0 şi 2, conexiunea devine $\{PG_0, Pg_1\} \leftrightarrow \{MG_0, MG_2\}$. Aceasta reclamă o conexiune $(3 \leftrightarrow 3)$. Sunt implicate SE 0, 1, 2, 4, 5 şi 6. Există 24 de conexiuni $(3 \leftrightarrow 3)$.

În fine, dacă conexiunea este între $\{PG_0, PG_1\} \leftrightarrow \{MG_2, MG_3\}$ toate cele patru SE de la intrare şi cele patru SE de la ieşire sunt necesare în exemplul din fig. 5.2.

O astfel de conexiune este denumită $(4 \leftrightarrow 4)$ şi există şase astfel de conexiuni. Conexiunea $(k \leftrightarrow k)$ este utilizată pentru a găsi numărul de SE de intrare şi de ieşire necesar pentru a conecta p şi m grupuri. Pentru un sistem (16×16), $x = 2k$ pentru $1 \le k \le 4$ şi $y = 8 - x$.

Pentru a exprima formal aceste combinaţii se utilizează un parametru de distribuţie $D_g(p,m,\delta)$ pentru un p şi m dat astfel:

$$D_g(p,m,\delta) = \binom{4}{\alpha}\binom{\alpha}{p-\delta}\binom{4-\alpha}{\delta} \tag{5.10}$$

unde:

$\alpha = \max(p,m)$

$\beta = \min(p,m)$

δ - numărul de grupuri de memorie (procesoare) selecţionate în exteriorul grupurilor de procesoare (memorii).

Pentru $\delta = 0$, $m \subset p$, dacă $p > m$ sau $p \subset m$, dacă $m > p$. Această situaţie produce o conexiune $(\alpha \leftrightarrow \alpha)$. Pentru $\delta = 1$, apare cazul când fie unul dintre grupurile de procesoare sau unul dintre grupurile de memorie este selectat din exteriorul numerelor corespunzătoare pentru a da o conexiune $((\alpha+1)\leftrightarrow(\alpha+1))$. În mod similar, $\delta = 2$ reprezintă două grupuri selectate în exteriorul numerelor de grup corespunzătoare pentru a da o conexiune $((\alpha+2)\leftrightarrow(\alpha+2))$. Teoretic, α poate varia între 0 şi β.

Fie $p = 3$ şi grupurile de procesoare 0, 1 şi 2. Dacă se aleg toate grupurile de memorii ce lucrează dintre 0, 1 şi 2, atunci aceasta este o selecţie corespondentă şi $m \subset p$. În exemplu prezentat, selecţia grupului de memorie 3 nu este o selecţie corespondentă şi $\delta = 1$.

Primul termen în (5.10) arată cât de multe căi avem pentru a selecta α din patru grupuri.

Al doilea termen reprezintă numărul de căi posibile pentru a selecta $(\beta - \delta)$ din α pentru o valoare dată a lui δ.

Al treilea termen arată selecţia lui δ din $(4-\alpha)$ grupuri.

Ecuaţia (5.10) generează următoarele numere pentru $p = m = 2$

$D_g(2,2,0) = 6 \Rightarrow$ conectare $2 \leftrightarrow 2$

$D_g(2,2,1) = 24 \Rightarrow$ conectare $3 \leftrightarrow 3$

$D_g(2,2,2) = 6 \Rightarrow$ conectare $4 \leftrightarrow 4$

Pentru un i şi j dat, p şi m pot varia între N_p şi 4 şi N_m şi 4, respectiv.

Se reprezintă numărul total de conexiuni $(k \leftrightarrow k)$ca fiind *sum(i, j),* unde:

$$sum\ (i,j) = \sum_{p=N_p}^{4} \sum_{m=N_m}^{4} \sum_{\delta=0}^{\beta} D_g(p,m,\delta) \tag{5.11}$$

Ecuația (5.11) generează diferite conexiuni $(k \leftrightarrow k)$ pentru fiecare p şi m posibile, care sunt funcții ale lui δ. Deoarece există patru grupuri, $k \in \{\alpha,4\}$.

Fiind dată *sum(i, j)* se pune problema câte din acestea sunt conexiuni $(k \leftrightarrow k)$. Numărul de conexiuni ce sunt de tipul $(k \leftrightarrow k)$ se notează cu *g(k, k)* pentru $\alpha \le k \le 4$ şi este necesar pentru a calcula probabilitatea ca o conexiune $(k \leftrightarrow k)$ să fie cerută.

$$g(k,k) = \sum_{p=N_p}^{k} \sum_{m=N_m}^{k} D_g(p,m,k-\alpha) \text{ pentru } \alpha \le k \le 4 \tag{5.12}$$

Utilizând (5.10) – (5.12) sunt generate mai multe conexiuni $(k \leftrightarrow k)$ cerute pentru un număr dat *i* de PE şi *j* de MM.

Rezultatele sunt prezentate în TABEL IX.

Pentru un p şi m dați, fiecare căsuță din tabel prezintă numărul diferit de grupe $(k \leftrightarrow k)$. Dacă elementele pot fi distribuite peste toate cele patru grupuri, atunci *sum(i, j)* este obținută prin adăugarea de intrări a tuturor căsuțelor între p şi 4 şi m şi 4.

Funcția *g(k,k)* este obținută prin adăugarea a $(k \leftrightarrow k)$ intrări tuturor căsuțelor utilizate în calcularea lui *sum(i,j).*

Pentru $p = m = 2$, numărul total de căsuțe ce trebuie incluse în calcul este prezentat cu linii îngroşate în TABEL IX. El dă *sum(i,j)*=121, *g(2,2)*=6, *g(3,3)*=52 şi *g(4,4)*=63. Atunci *g(k,k)/sum(i,j)* este probabilitatea că o conexiune $(k \leftrightarrow k)$ este cerută.

TABEL IX. Tabelul de căutare a conexiunilor

m \ p	1	2	3	4
1	4(1-1) 12(2-2)	12(2-2) 12(3-3)	12(3-3) 4(4-4)	4(4-4)
2	12(2-2) 12(3-3)	6(2-2) 24(3-3) 6(4-4)	12(3-3) 12(4-4)	6(4-4)
3	12(3-3) 4(4-4)	12(3-3) 12(4-4)	4(3-3) 12(4-4)	4(4-4)
4	4(4-4)	6(4-4)	4(4-4)	1(4-4)

Reamintim că o conexiune $(k \leftrightarrow k)$ determină numărul x de comutatoare solicitate ce trebuie să fie active pentru grupurile de p procesoare, în vederea conectării la grupurile de m memorii. MK pentru RIN-MN este prezentat în fig. 5.10.

Se notează cu (x, y) numărul de x SE-uri cerute şi cu y numărul de SE-uri suplimentare în RIN-MN, astfel ca $x + y = N/4 \log N/4 = S_t$ – numărul total de comutatoare din sistem.

Se notează cu λ_{sc} , μ_{sc}, C_{sc} şi r_{sc} *ratele de defectare, de reparare, factorul de acoperire* şi *factorul de reparare cu succes* pentru comutatoarele SE.

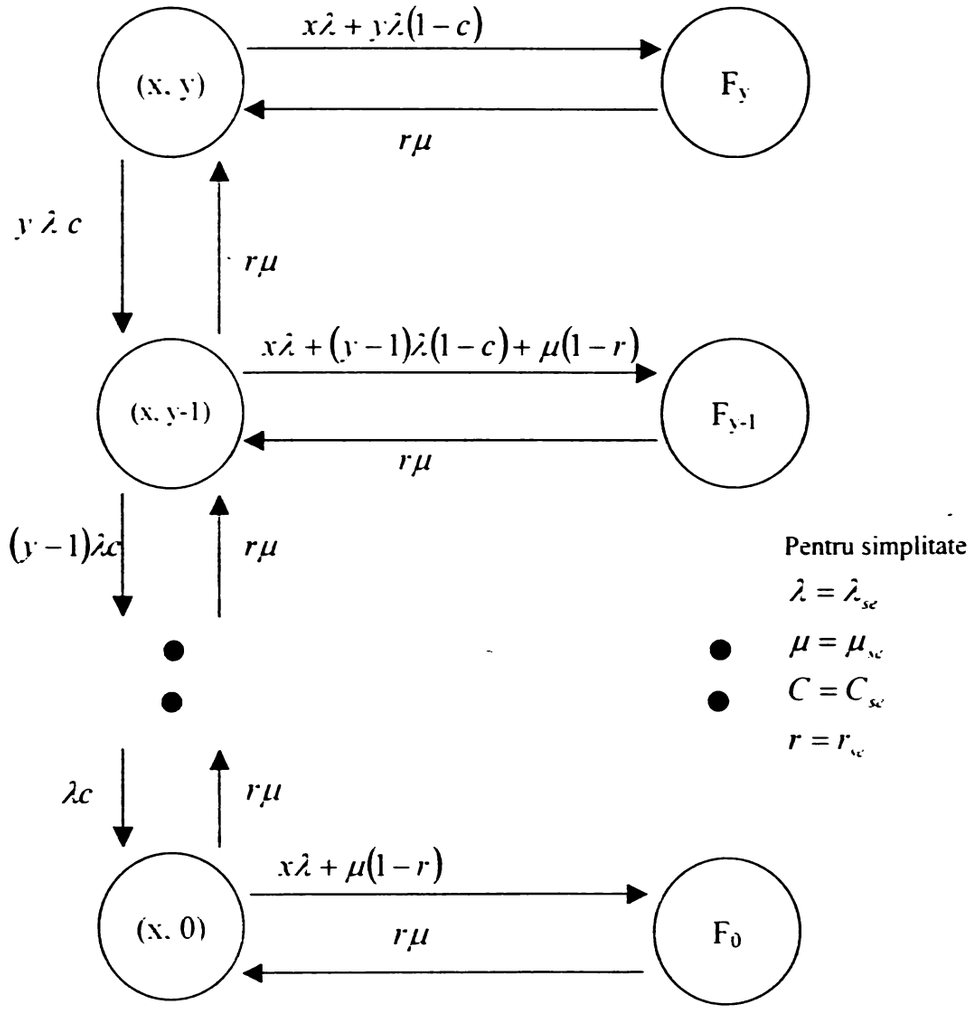


Fig. 5.10. Lanț Markov pentru RIN-MN

Tranziția orizontală cu rata de $x\lambda_{se}C_{se}$ arată că sistemul se defectează când oricare dintre comutatoarele solicitate se defectează. A doua parte a tranziției orizontale cu termenul $(1-C_{se})$ reprezintă defectarea sistemului datorită acoperirii imperfecte a SE-urilor defecte ce aparțin grupului suplimentar, de extensie.

Deşi un comutator suplimentar, un extra-SE, nu contribuie la nici o conexiune utilă, defectarea sa poate fi detectată de o astfel de manieră încât reconfigurarea sistemului este posibilă. A treia parte reprezintă repararea imprecisă a SE-urilor.

Tranzițiile verticale reflectă defectarea şi repararea a celor y extra SE-uri. Se observă că dintr-o stare (x,y), subsistemul RIN-MN ar putea evolua fie spre o stare defectă – tranziție orizontală – sau spre o stare $(x, y\text{-}1)$ – tranziție verticală – deoarece doar un singur SE ar putea să se defecteze la un moment dat. Prin urmare, rata de reparare este întotdeauna μ_{se} în MK prezentat în fig. 5.10. Repararea unui SE necesar în conexiune se consideră că este prioritară față de cea a unui extra-SE. Această strategie evită tranziția între stările defecte şi ameliorează disponibilitatea RIN-MN.

Modelul prezentat în fig. 5.10 poate modela orice degradare în performanțe cu valori adecvate ale lui x şi y. De exemplu, dacă procesoarele şi memoriile active au nevoie de cel puțin două grupuri de procesoare şi două grupuri de memorii, RIN-MN din fig. 5.10 trebuie să furnizeze o conexiune $2\leftrightarrow 2$, una $3\leftrightarrow 3$ sau una $4\leftrightarrow 4$, depinzând de distribuția procesoarelor şi memoriilor. O conexiune $2\leftrightarrow 2$ furnizează x=4, y=4, o conexiune $3\leftrightarrow 3$ dă x=6, y=2 şi o conexiune $4\leftrightarrow 4$ înseamnă x=8, y=0 (nu există degradare a RIN-MN). Astfel, pentru evaluarea unui sistem cu o degradare de 50%, MK-ul din fig. 5.10 ar putea fi rezolvat de trei ori, cu trei seturi diferite de valori (x, y).

Fie $P_{(x,l)}(t)$ probabilitatea ca sistemul să fie în starea (x,l) la momentul t. Soluția MK-ului din fig. 5.10, utilizând pachetul HARP, va furniza $P_{(x,l)}(t)$ pentru $0 \le l \le y$.

Probabilitatea ca RIN-MN să asigure o conexiune $k\leftrightarrow k$ cu x SE active este $P_{(k\leftrightarrow k)}(t)$ şi are expresia :

$$P_{(k\leftrightarrow k)}(t)=\sum_{l=0}^{\lambda} P_{(x,l)}(t) \qquad (5.13)$$

Pentru a recapitula, pentru orice subsistem $(i\times j)$, numărul p de grupuri de procesoare şi numărul m de grupuri de memorii se situează în domeniul $N_p \le p \le 4$ şi $N_m \le m \le 4$, respectiv. Valoarea lui k pentru o conexiune $k\leftrightarrow k$ se situează în domeniul $\max(p,m)\le k \le 4$. În final,

pentru o conexiune $k \leftrightarrow k$ cunoscută, valorile x şi y ale unei configuraţii (16 × 16) este obţinută ca fiind $2k$ şi respectiv (8-2k).

Probabilitatea că există un grup ($i \times j$) conectat într-un sistem (16 × 16) atunci când i procesoare şi j memorii sunt active este :

$$P_{(i\times j)}^{(1)}(t) = \sum_{p=N_p}^{4} \sum_{m=N_m}^{4} \left(\sum_{k=\max(p,m)}^{4} \frac{g(k,k)}{sum(i,j)} P_{(k\leftrightarrow k)}(t) \right) P_i(t) P_j(t) \qquad (5.14)$$

În această expresie, indicele superior 1 reprezintă primul caz, acela în care doar i PE şi j MM sunt active. Primele două sume denotă diversele combinaţii ale grupurilor de procesoare şi de memorii conţinând i PE şi j MM. Al treilea termen de însumare arată că k poate varia până la 4. De exemplu, dacă p = 2, m = 3, atunci k = 3. Aceasta înseamnă că dacă i PE şi j MM sunt distribuite în două grupuri de procesoare şi în trei grupuri de memorii, depinzând de distribuţie, atunci sunt necesare şase SE-uri de tip $\left(P_{(3\leftrightarrow 3)}(t)\right)$ sau toate cele opt SE-uri $\left(P_{(4\leftrightarrow 4)}(t)\right) \cdot g(k,k)/sum(i,j)$ ce dau probabilitatea de a solicita o conexiune $k \leftrightarrow k$.

Pentru p=m=3, g(3,3)=16, g(4,4)=39 şi sum(i,j)=55 pentru TABEL IX. Multiplicând probabilitatea subsistemului de comutatoare cu $P_i(t)$ şi $P_j(t)$, obţinute din subsistemele procesor şi de memorie, se obţine probabilitatea din primul caz care este $P_{(i\times j)}^{(1)}(t)$.

În al doilea caz, conexiunea este restricţionată la ($i \times j$) când α PE şi β MM sunt active în sistem, pentru $\alpha \geq i$, $\beta \geq j$. Aceasta este posibil doar dacă comutatoarele solicitate i şi j sunt active în grupuri separate şi $(\alpha - i)$ şi $(\beta - j)$ elemente funcţionează în grupuri separate şi nu există conexiuni între subsistemul ($i \times j$) activ şi restul componentelor active. Să considerăm un exemplu în care i PE şi j MM lucrează utilizând comutatoarele 1, 2, 3, 5, 6 şi 7 din fig. 5.2. Dacă oricare altă componentă lucrează în aceste trei grupuri de procesoare şi în aceste trei grupuri de memorii, atunci conectivitatea creşte.

Procesoarele suplimentare şi/sau memoriile trebuie să funcţioneze în grupul de procesoare 0 şi/sau în grupul de memorii 0, şi în grupul SE 0 sau 1. Aceasta implică că dacă i PE şi j MM solicitate sunt distribuite în cele patru grupuri, nu poate exista orice altă componentă suplimentară.

Probabilitatea ce este specifică cazului doi, devine nulă în această situaţie.

Această situaţie poate fi enunţată astfel : «Care este probabilitatea ca i procesoare şi j memorii să necesite o conexiune $k \leftrightarrow k$ pentru $1 \leq k \leq 3$, şi ca $(\alpha - i)$ şi $(\beta - j)$ elemente să fie distribuite în restul celor (4-k) grupuri ? »

Acest enunț poate fi exprimat astfel :

$$P_{cx}(t)=\frac{\sum_{k=\max(N_p,N_m)}^{3}\binom{4}{k}\binom{k\times 4}{i}\binom{N-4\times k}{\alpha-i}\binom{k\times 4}{j}\binom{N-4\times k}{\beta-j}}{\binom{N}{\alpha}\binom{N}{\beta}\binom{\alpha}{i}\binom{\beta}{j}} \tag{5.15}$$

În această expresie, numărătorul oferă numărul de căi pentru a alege i PE din $k \times 4$ procesoare aparținând la k grupuri (fiecare grup într-un sistem 16 × 16 are 4 PE sau 4 MM), j MM dintre $k \times 4$ memorii din k grupuri, $(\alpha-1)$ procesoare din restul a (N–4 × k) procesoare, distribuite în (4–k) grupuri, şi $(\beta-j)$ memorii din restul celor (N–4 × k) memorii. $\binom{4}{k}$ reprezintă numărul de căi prin care se poate selecta un grup $k \leftrightarrow k$ procesor-memorie ce conține un subsistem ($i \times j$). Numitorul oferă numărul total de căi prin care i PE şi j MM pot fi selectate. Limita superioară a lui k este 3 deoarece distribuția a i PE şi j MM în cele patru grupuri face ca probabilitatea cazului al doilea să fie nulă.

Starea (x,0) în MK prezentat în fig. 5.10 reprezintă situația în care doar comutatoarele solicitate lucrează şi toate celelalte SE adiționale s-au defectat.

Includerea acestei probabilități de stare ar garanta că procesoarele adiționale active şi/sau memoriile corespunzătoare nu pot să crească talia sistemului activ mai mult de ($i \times j$). De notat că $P_{(x,0)}(t)$ poate fi obținută pentru un grup $(k \leftrightarrow k)$ dat.

Probabilitatea pentru cel de al doilea caz unde se poate obține o conexiune ($i \times j$) din partea a α PE active şi β MM active se notează cu $P_{(\alpha,\beta)}(t)$.

$$P_{(\alpha,\beta)}(t)=P_{cx}(t)\cdot P_{(x,0)}(t)\cdot P_{\alpha}(t)\cdot P_{\beta}(t) \tag{5.16}$$

unde $P_{\alpha}(t)$ şi $P_{\beta}(t)$ dau probabilitatea ca α PE şi β MM să fie active la momentul t. Deoarece α şi β pot fi în domeniul $i \le \alpha \le N$ şi $j \le p \le N$ respectiv, probabilitatea cazului al doilea este :

$$P_{(i,j)}^{(2)}(t)=\sum_{\alpha=i+1}^{N}\sum_{\beta=j+1}^{N}P_{(\alpha,\beta)}(t)+\sum_{\alpha=i+1}^{N}P_{(\alpha,j)}(t)+\sum_{\beta=j+1}^{N}P_{(i,\beta)}(t) \tag{5.17}$$

Al doilea şi al treilea termen în (5.17) au o contribuție neglijabilă şi pot fi ignorate pentru a face calculul mai rapid.

Finalmente, disponibilitatea sistemului devine :

$$A(t)=\sum_{i=1}^{N}\sum_{j=1}^{N}\left(P_{(i,j)}^{(1)}(t)+P_{(i,j)}^{(2)}(t)\right) \tag{5.18}$$

5.5.3 Un model generalizat

Pentru a generaliza modelul prezentat anterior în [DTB93] sunt prezentate ca fiind necesare următoarele.

În primul rând, metodologia de modelare trebuie să fie extinsă pentru a se putea analiza sisteme mai mari cu o acuratețe rezolvabilă.

Al doilea obiectiv este de a include analiza sistemelor a căror talie nu este o putere a lui 4. Această necesitate apare datorită faptului că multe SMP pot fi configurate pentru diferite talii. De exemplu, configurația Butterfly poate fi proiectată pentru orice talie până la 256 de noduri.

Extensia modelului trebuie să includă şi alte clase de RIN-MN şi de SMP-MP.

Ideea principală este de a descompune sistemul în diferite grupuri. Distribuția a i PE şi j MM poate fi făcută într-o manieră similară cu cea de la sistemele (16 × 16).

În funcție de talia sistemului şi cu ajutorul lanțului Markov prezentat în fig. 5.9, se poate calcula $P_i(t)$ şi $P_j(t)$. Problema ce trebuie detaliată o reprezintă determinarea numărului x de SE necesare. Numărul y de SE suplimentare poate fi obținut calculând $y = S_t - x$ unde S_t este numărul total de SE din RIN-MN.

Cunoaşterea lui x şi y este esențială pentru aflarea soluției modelului MK prezentat în fig. 5.10.

A. Modelarea într-o configurație de noduri N=4^i

Se consideră un sistem ($N \times N$) unde $N = 4^t$. El are $n = \log_4 N$ niveluri conținând $N/4$ SE în fiecare nivel. Se numerotează nivelurile de la 0 la (n-1) dinspre partea procesorului.

Prin urmare, $S_t = n \cdot (N/4)$ reprezintă numărul total de comutatoare. Procesoarele sunt grupate în patru grupuri de procesare – PG – şi modulele de memorie în patru grupe de memorii – MG. Fiecare grup, indiferent dacă este PG sau MG, are 4^{t-1} elemente. Un PE întotdeauna are nevoie de un nivel specific 0 de SE în scopul de accesa oricare MM. Astfel, comutatoarele din nivelul 0 sunt incluse cu PG-urile corespunzătoare. Pentru a simplifica calculul, SE din fiecare nivel de la 2 la (n – 1) sunt, de asemenea, divizate în patru grupuri. Fiecare grup conține $N/16$ SE / nivel. Există o relație între diferitele grupuri de $N/16$ SE/nivel şi grupurile MG asociate. Accesul la grupurile MG poate fi asigurat prin includerea acestor SE cu MG-urile lor respective. Fiecare MG conține, astfel, $(n-2) \cdot (N/16)$ comutatoare. În acest mod, doar comutatoarele din nivelul 1 nu sunt incluse, nici în grupurile PG, nici în cele MG.

Comutatoarele din nivelul 1 sunt acelea ce realizează conectarea dintre grupurile PG şi cele MG. Numărul de SE din nivelul 1 depinde de distribuția grupurilor PG şi MG cerute. Bazat pe funcționalitate, se grupează comutatoarele nivelului 1 în grupuri de comutatoare – SG. Un grup

de comutatoare *SGpq* asigură conexiunea de la PG_p la MG_q. Nodurile de comutare sunt prezentate în TABEL X şi acesta este util pentru formalizarea conceptului.

TABEL X. Tabelul nodurilor de comutare

PG \ MG	0	1	2	3
0	00	01	02	03
1	10	11	12	13
2	20	21	22	23
3	30	31	32	33

Fiecare locaţie din tabel se referă la un SG şi. prin urmare, există 16 astfel de locaţii. Numărul de SE în fiecare grup este egal cu $N / 64$. Din descrierea conexiunii $(k \leftrightarrow k)$ oferită ca şi exemplu pentru analiza unui sistem cu 64 de noduri, doar un SG este cerut pentru o conexiune $(1 \leftrightarrow 1)$ în timp ce pentru conectări $(2 \leftrightarrow 2)$ sau $(3 \leftrightarrow 3)$ sunt necesare patru sau, respectiv nouă SG.

Atât SG_{pq} şi SG_{qp} sunt necesare cu scopul de a avea o conexiune între PG_p şi MG_q. Numărul de SG-uri necesar pentru o conexiune $(k \leftrightarrow k)$ este, prin urmare, egal cu k^2. De exemplu, o configuraţie cu 1024 noduri are 256 SE în nivelul 1 şi fiecare din cele 16 grupuri are 16 SE-uri. Fiecare locaţie din TABEL X în acest caz reprezintă un grup de 16 SE-uri.

Numărul total de SE-uri necesare pentru o conexiune de grup $(k \leftrightarrow k)$, pentru $1 \le k \le 4$, se obţin astfel :

x = (numărul de SE în k PG-uri) + (numărul de SE în k MG-uri) + (numărul de comutatoare de pe nivelul 1 necesare pentru a conecta k PG-uri la k MG-uri)

$$= k \cdot \frac{N}{16} + k(n-2) \cdot \frac{N}{16} + k^2 \cdot \frac{N}{64}$$

$$= \frac{Nk}{16}\left[n - 1 + \frac{k}{4}\right] \qquad (5.19)$$

Modelul MK din fig. 5.10 este rezolvat utilizând valoarea lui x şi valoarea corespunzătoare a lui y.

Ecuaţiile (5.13) şi (5.14) se utilizează, în continuare, pentru a găsi probabilitatea ca să existe un grup ($i \times j$) conectat când sunt active i PE şi j MM. $P_{ct}(t)$ este obţinut din (5.12) înlocuind $(k \times 4)$ cu $(k \times N/4)$

Probabilitatea unei conexiuni $(i \times j)$ când mai mult decât i PE şi j MM sunt active se obţine rezolvând (5.16) şi (5.17). În final, (5.18) oferă disponibilitatea sistemului.

B. Modelarea într-o configuraţie de noduri $N \neq 4^i$

Se consideră un sistem $N \times N$ unde $4^{i-1} < N < 4^i$ pentru un întreg i oarecare. Numărul n de niveluri necesar pentru a configura RIN-MIN este i. Următorul parametru de configurare este numărul de procesoare şi de grupuri de memorie. Divizarea SMP în patru grupe ar putea duce la o proiectare ineficientă şi scumpă. De exemplu, să analizăm situaţia unui sistem cu 100 de noduri. Se pot repartiza aceste noduri în toate cele patru grupuri. Multe dintre comutatoarele (4×4) ar putea fi subutilizate sau chiar neutilizate. Este, prin urmare, esenţial ca aceste noduri să fie grupate în cât mai puţine grupuri posibile. O grupare densă înseamnă a selecta nodurile de o astfel de manieră încât să se maximizeze utilizarea comutatorului. Aceasta conduce la formarea de grupuri de 4^{n-1} noduri. Astfel, numărul de grupuri va depinde de talia N a sistemului.

Fie G numărul de grupuri unde $1 \le G \le 4$.

Atunci

$$G = \left\lceil \frac{N}{4^{n-1}} \right\rceil \tag{5.20}$$

Prin urmare, exemplul sistemului cu 100 noduri necesită două grupuri. Distribuţia a N noduri în G grupuri poate fi realizată în multe moduri dacă $N \neq G \cdot 4^{i-1}$.

În exemplul anterior, distribuţia a 100 noduri în două grupuri este (50, 50), (64, 34), etc.

Se presupune că N este distribuit în mod egal în cele G grupuri, astfel că numărul de SE necesare pentru fiecare grup este acelaşi.

Numărul minim de PG-uri şi MG-uri pentru analiza a i PE şi j MM este :

$$N_p = \left\lceil \frac{i}{N div G} \right\rceil \text{ şi } N_m = \left\lceil \frac{j}{N div G} \right\rceil \tag{5.21}$$

Cele i PE pot fi distribuite peste p PG pentru $N_p \le p \le G$ şi j MM pot fi distribuite peste m MG pentru $N_m \le m \le G$. Fiecare din aceste distribuţii necesită un set specific de SE pentru

interconectare. Înlocuind valoarea 4 în G în ecuațiile (5.10), (5.11) şi (5.12), se pot calcula valorile pentru *g(k,k)* şi *sum(i,j)* unde $\alpha \le k \le G$.

Calcularea lui x este similară ca şi tehnică cu cea descrisă în 5.5.3.A. Comutatoarele de pe nivelul 0 sunt incluse în grupurile PG şi astfel, fiecare PG are N/G PE-uri şi $\lceil (1/G)/(N/4) \rceil$ comutatoare pe fiecare nivel. Toate comutatoarele situate pe nivelul 2 până la cele de pe nivelul $(n-1)$ sunt incluse în MG-urile corespunzătoare. Prin urmare, fiecare MG constă din M/G grupuri MM şi $(n-2)\cdot\lceil N/4G \rceil$ comutatoare. Pasul următor constă în determinarea numărului solicitat de comutatoare de pe nivelul 1 necesare pentru o conexiune $(k \leftrightarrow k)$. Tabelul nodurilor de comutare pentru o astfel de conectare este TABEL X, dar numărul de linii şi coloane este restrâns la G.

Deoarece există G grupe PG şi G grupe MG în TABEL X, se descompun comutatoarele de pe nivelul 1 în G^2 comutatoare. Fiecare SG conectează un PG specific la un MG. Numărul de comutatoare în fiecare SG este dat de $(N/4)\times(1/G^2) = N/4G^2 \cdot N/4G^2$. El este un număr întreg în anumite cazuri. Aceasta se datorează situației când talia sistemului nu este o putere a lui 4. Atunci, câteva dintre comutatoarele de pe nivelul 1 nu sunt utilizate pentru conectare.

Numărul total x de comutatoare pentru o conexiune $(k \leftrightarrow k)$ pentru $1 \le k \le G$ se obține din :

$$x = k \cdot \frac{N}{4G} + k(n-2)\cdot\frac{N}{4G} + k^2 \cdot \left\lfloor \frac{N}{4G^2} \right\rfloor$$

$$= \frac{Nk}{4G}(n-1) + k^2 \left\lfloor \frac{N}{4G^2} \right\rfloor \qquad (5.22)$$

Valorile x şi y astfel obținute sunt utilizate în modelul MK din fig. 5.10.

C. Extensia modelului

Tehnica de modelare propusă poate fi utilizată pentru evaluarea RIN-MN ce utilizează comutatoare de altă mărime decât 4×4. Analiza grupurilor PE şi MM rămâne aceeaşi. Calculul diferiților parametrii utilizând p grupuri de PE active şi m grupuri de MM active se pot utiliza aplicând expresiile derivate în 5.5.2.

Dacă talia comutatorului este $a \times a$ atunci numărul de grupuri trebuie să fie a în loc de 4.

O a doua considerație trebuie făcută în legătură cu x ce trebuie utilizat în modelul MK din fig. 5.10.

Se pot utiliza diferite tehnici de aproximare pentru aflarea valorii lui x. O abordare constă în utilizarea tuturor comutatoarelor unui grup, aşa cum s-a prezentat anterior. Această metodă este

rezonabilă pentru $a \geq 4$, deoarece cu un număr mai mare de comutatoare, numărul de niveluri scade. Pe de altă parte, pentru $a = 2$ sau 3, este necesară o aproximare mai exactă a lui x.

Următoarea extensie a modelului se referă la SMP-MP. Un astfel de sistem are toate procesoarele PE dispuse pe o parte, iar memoriile MM pe cealaltă parte a RIN-MN. Este necesară doar o cale de la un PE la un MM, în vederea conectării.

Utilizând notațiile din 5.5.3.B se vor obține $p \leftrightarrow m$ conexiuni în loc de $(k \leftrightarrow k)$ conexiuni, unde $N_p \leq p \leq G$ şi $N_m \leq m \leq G$. Numărul necesar de comutatoare este prezentat în TABEL X.

Toleranța la defect poate fi ameliorată prin utilizarea unor tehnici specifice. Aceste tehnici includ adăugarea unor niveluri suplimentare, crescând numărul de porturi, replicând RIN-MN, etc. etc. Deoarece redundanța şi topologia a RIN-MN tolerante la defect variază, este dificil de a formaliza o tehnică generală pentru modelare şi pentru calcularea disponibilității.

Tehnica de modelare a unui subsistem RIN-MN variază în funcție de topologie. Conceptul prezentat poate fi aplicat la RIN-MN tolerante la defect, dacă comutatoarele pot grupate în două grupuri: numărul x cerut de SE şi numărul y de comutatoare suplimentare. Este dificil de identificat x şi y într-o RIN-MN *cu căi multiple*. Validitatea modelului unei RIN-MN tolerantă la defect depinde de uşurința cu care se pot găsi valorile corespunzătoare pentru x şi y.

5.6 CONCLUZII

Analiza de dependabilitate în arhitecturile paralele se referă atât *la analiza de fiabilitate* cât şi la cea de *disponibilitate* [RA90], [RG87], [TD89], [TZ94].

Fiabilitatea este una dintre cele mai importante atribute ale sistemelor de calcul. Cele mai utilizate tehnici de analiză ale fiabilității sunt abordările analitice. Dintre acestea am analizat *modelarea combinatorie* şi *modelarea Markov*.

Modelarea combinatorie utilizează tehnici probabilistice ce enumeră modurile diferite în care un sistem poate rămâne operațional. Probabilitățile evenimentelor ce conduc la starea în care un sistem devine operațional sunt calculate ca să formeze o estimare a fiabilității sistemului. Fiabilitatea sistemului este în general exprimată în termeni de fiabilități ale componentelor individuale. În practică se utilizează două modele, serie şi paralel. În *sistemul serie*, fiecărui element al sistemului i se pretinde să funcționeze corect pentru ca sistemul în ansamblul lui să funcționeze corect. Într-un *sistem paralel*, doar câteva din elemente trebuie să fie operaționale pentru ca sistemul să funcționeze corect.

Ca şi dezavantaj major al acestei metode se remarcă faptul că sistemele complexe nu pot fi modelate convenabil deoarece expresiile fiabilității devin foarte complexe. În plus, acoperirea defectului este dificil de incorporat în expresii combinaționale.

Modelele Markov utilizează concepte de *stare* şi de *tranziție a stării*. Pentru modelele de fiabilitate fiecare stare a modelului Markov reprezintă o combinație distinctă de module ce defectează şi de module fără defect. Tranzițiile de stare sunt caracterizate de probabilități, ca şi *probabilitatea de defectare*, *de acoperire a defectului* şi *de reparare*. Principalul beneficiu al utilzării MK în modelarea dependabilității îl reprezintă abilitatea de a manipula acoperirea defectului într-o manieră sistematică. Încorporarea acoperirii C în analiza MK permite modelarea siguranței sistemului.

Complexitatea din punct de vedere al calculului al unei analize de dependabilitate a fost redusă drastic prin fragmentarea în module mai mici, utilizând un model de rețea ierarhică mai apropiat de modelul de sistem şi care favorizează comunicările locale. Proiectanții combină metode algebrice Booleene binecunoscute cu un model de rețea ierarhică pentru a evalua *fiabilitatea de terminal*, *fiabilitatea de rețea* şi *fiabilitatea bazată pe sarcină*.

S-au dezvoltat mai mulți algoritmi pentru evaluarea fiabilității SMP. În aceşti algoritmi, fiabilitatea de tip terminal este utilizată în mod comun ca şi o măsură a conectivității. A fost studiată de mulți cercetători şi există diferite metode de evaluare a acestui parametru. Amintim, cu tilu informativ, că cea mai simplă dintre aceste tehnici este de a enumera stările favorabile prin examinarea evenimentelor posibile. Evident, acest lucru este imposibil în rețelele de dimensiuni mari.

Modelarea disponibilității presupune încorporarea reparării. Lanțurile Markov au proprietatea de a fi ciclice. Prin introducerea noțiunii de *reparare* se permite reîntoarcerea de la o stare mai puțin operațională la o stare mai operațională sau chiar complet operațională. Pentru aceasta se specifică o probabilitate de întoarcere la nivel de tranziție. Evaluarea experimentală a disponibilității nu este întotdeauna posibilă din cauza constrângerilor de timp şi de cost. Din această cauză se utilizează frecvent parametri *MTTF* sau *MTTR* ceea ce conduce la disponibilitatea stării ferme, A_{SS}.

Disponibilitatea sistemului este evaluată în sistemele de dimensiuni mari şi prin alte tehnici dintre care amintim strategiile de rutare ierarhice. Aceasta înseamnă că fiabilitatea este asociată cu existența unor căi de comunicare între toate perechile de noduri. Procesul de obținere a expresiilor de fiabilitate pornind de la un set minimal este un proces consumator de timp. De exemplu calculul fiabilității de tip *k-terminal* prin considerarea tuturor combinațiilor posibile de noduri şi legături funcționale şi nefuncționale a unei rețele ierarhice este exponențial în natura sa

şi puternic restrictiv în reţelele ierarhice. În literatură [DA93] se indică mai multe metode de evitare prin furnizarea unor expresii aproximative ale fiabilităţii pentru o reţea oricât de mare, prin modelarea acestora într-o formă ierarhică şi prin calcularea diverşilor parametri ale fiecărui grup ce conţine un număr relativ redus de noduri. Ulterior se combină aceste expresii pentru a obţine o valoare generală a fiabilităţii.

Metodele de analiză şi evaluare a dependabilităţii sunt numeroase, înregistrându-se o direcţie fermă de cercetare în acest sens. Tehnici de evaluare exacte presupun însă un volum mare de calcul şi de aceea proiectanţii se bazează pe soluţii de proiectare devenite clasice, ale căror fiabilitate şi disponibilitate sunt cunoscute, concentrându-se însă asupra unor expresii statistice ale dependabilităţii întregului sistem. Tendinţa firească este însă de a utiliza simulatoare şi programe de test pentru a se obţine o privire de ansamblu cu un grad de generalitate acceptat asupra comportării unui sistem din punct de vedere al dependabilităţii.

6. MODELUL EXPERIMENTAL

6.1 ARHITECTURA SMP – MP UTILIZAT CA ŞI MODEL

Sistemul de calcul multiprocesor, utilizat ca şi model experimental, reprezintă o arhitectură clasică pentru un mediu de calcul cu memorie partajată distribuită.

Memoria este distribuită printre procesoare, dar toate memoriile partajează acelaşi spaţiu de adresă.

Nodurile N sunt conectate între ele prin intermediul unei reţele de interconectare de bazată pe *magistrală multinivel* (*Multistage Bus Network*). Vom utiliza în continuare pentru acest tip de reţea acronimul RMM.

Arhitectura sistemului multiprocesor este prezentată în fig.6.1.

Fiecare nod N_i conţine :

- procesorul P_i
- memoria M_i, locală pentru P_i, şi controlerul ei CM_i
- memoria cache C_i şi controlerul ei CC_i
- memoria cache utilizată pentru director D_i şi controlerul ei CD_i
- interfaţa cu RMM

Un procesor P_i din nodul N_i este conectat la un modul de memorie M_i atât prin intermediul RMM cât şi direct. M_i este o *memorie locală* pentru P_i.

Ierarhia de memorie este constituită din *memorii cache* situate pe stratul (layer) superior L_1, din *memorii cache* situate pe stratul L_2 utilizat de procesoarele ce aparţin aceluiaşi *grup* (*cluster processors*) şi din *memoriile externe*, locale sau situate la distanţă. Nivelul L_1 conţine memorii cache de capacitate redusă (8 – 32 Kbytes) ce servesc o cerere de acces în cadrul unui singur ciclu de procesor. Nivelul L_2 conţine memorii cache de capacitate mult mai mare (1 – 4 Mbytes) şi onorează o solicitare în mai multe cicluri de procesor. Deoarece întregul spaţiu de adresă este partajat, dar memoria este fizic distribuită, eşecurile de acces la L_1 şi la L_2 pot fi tratate atât în memoria locală cât şi în cea situată la distanţă. Accesul la memorie este neuniform şi prin urmare organizarea este denumită cu *acces neuniform la memorie* (*Non-Uniform Memory Access*) - NUMA. Accesul la memoria locală necesită transferul de date prin magistrala internă din nod şi

acces la memoria DRAM. Accesul la memoria situată la distanță necesită transferuri de date prin magistrală, transferuri de mesaje prin RIN, şi acces la DRAM. Astfel, timpul de deservire a unui acces ratat la memoria cache, în mod special atunci când data trebuie adusă de la o memorie situată la distanță, poate fi de câteva ordine de mărime mai mare ca şi timpul de acces la memoria cache.

Arhitectura sistēmului de calcul este prezentată extins, cu evidențierea ierarhiei de memorie, în fig. 6.2.

De remarcat că în timp ce latențele în accesul la memoria locală pot fi tolerate, accesele la memoria de la distanță, generate în timpul execuției, pot reduce drastic performanța.

Pentru a reduce impactul latențelor în accesul memoriilor de la distanță în literatură [SLM89], [MBL89], [ADT90], [NB93], [IB00], [IB99], [BINK98] s-au propus mai multe soluții. Amintim câteva dintre ele :

- creşterea eficienței memoriilor cache private
- ameliorare performanțelor RIN
- managementul memoriilor ierarhizate

O atenție specială trebuie acordată proiectării comutatorului în cadrul unei RIN-MN deoarece există un impact direct asupra performanței SMP-MPD.

În cadrul modelului experimental am luat în considerare un comutator 2 X 2 bazat pe o magistrală internă. Am utilizat comutarea de pachete ca şi tehnică de comutare prin rețea. În ipoteza că se utilizează tehnici de comutare mai avansate [BWK98], [BINK97], [BIK00] ca şi *wormhole*, *canale virtuale* şi *rutare adaptivă* se pot aborda arhitecturi de comutatoare mult mai sofisticate. De exemplu SGI Spider [IB00], similar cu Intel Cavallino, sunt comutatoare *crossbar* ce utilizează canale virtuale. În literatură [BWK98], [Iyer99] se prezintă diferite arhitecturi de comutatoare 2 X 2 ce utilizează canale virtuale cu sau fără tampoane. Este interesant de remarcat că prin incorporarea unei memorii cache rapide SRAM în cadrul fiecărui comutator dintr-o RIN-MN se pot colecta datele partajate aşa cum evoluează ele prin rețea şi se pot astfel asigura accesele viitoare de la procesoarele ce reutilizează aceste date.

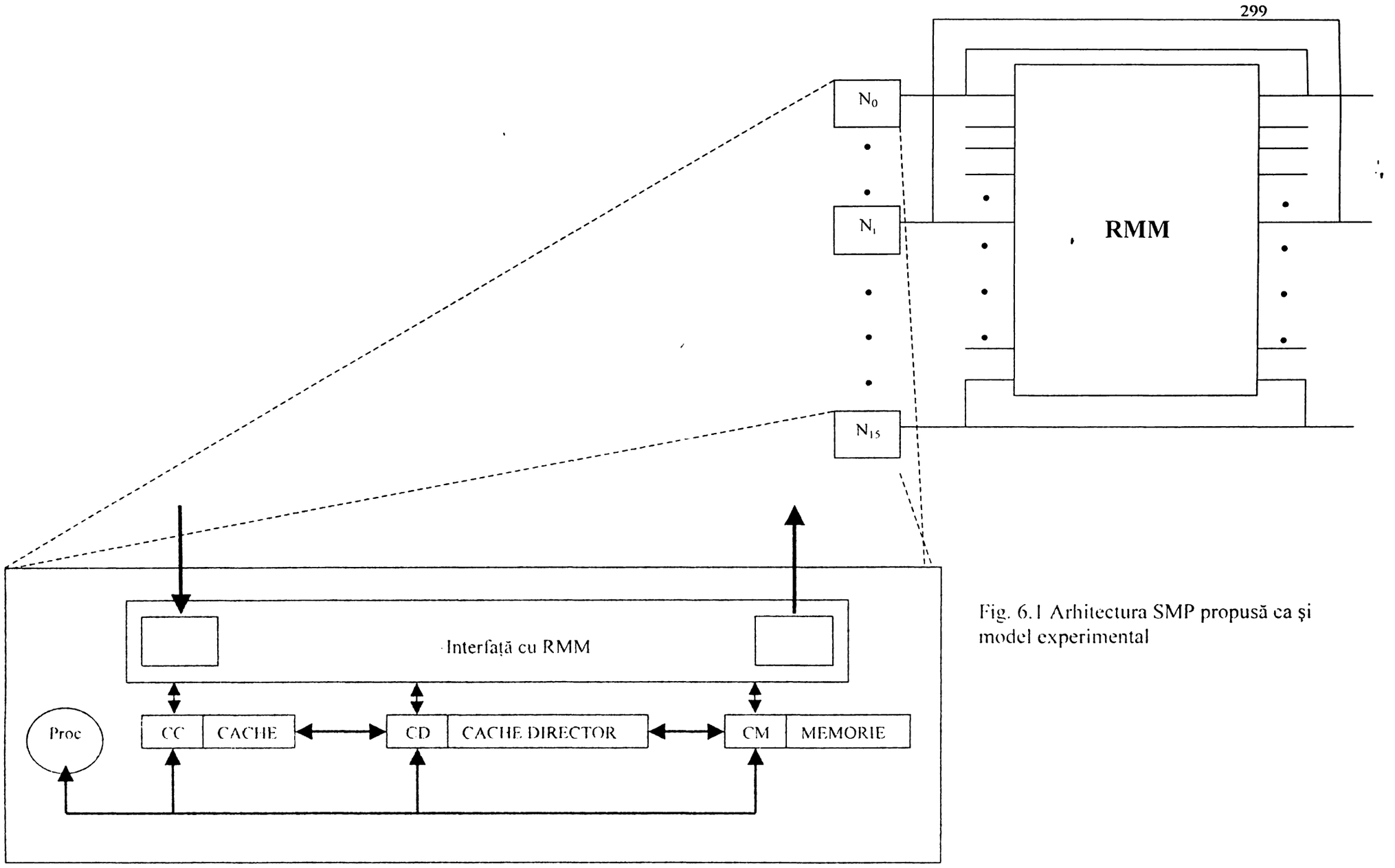


Fig. 6.1 Arhitectura SMP propusă ca și model experimental

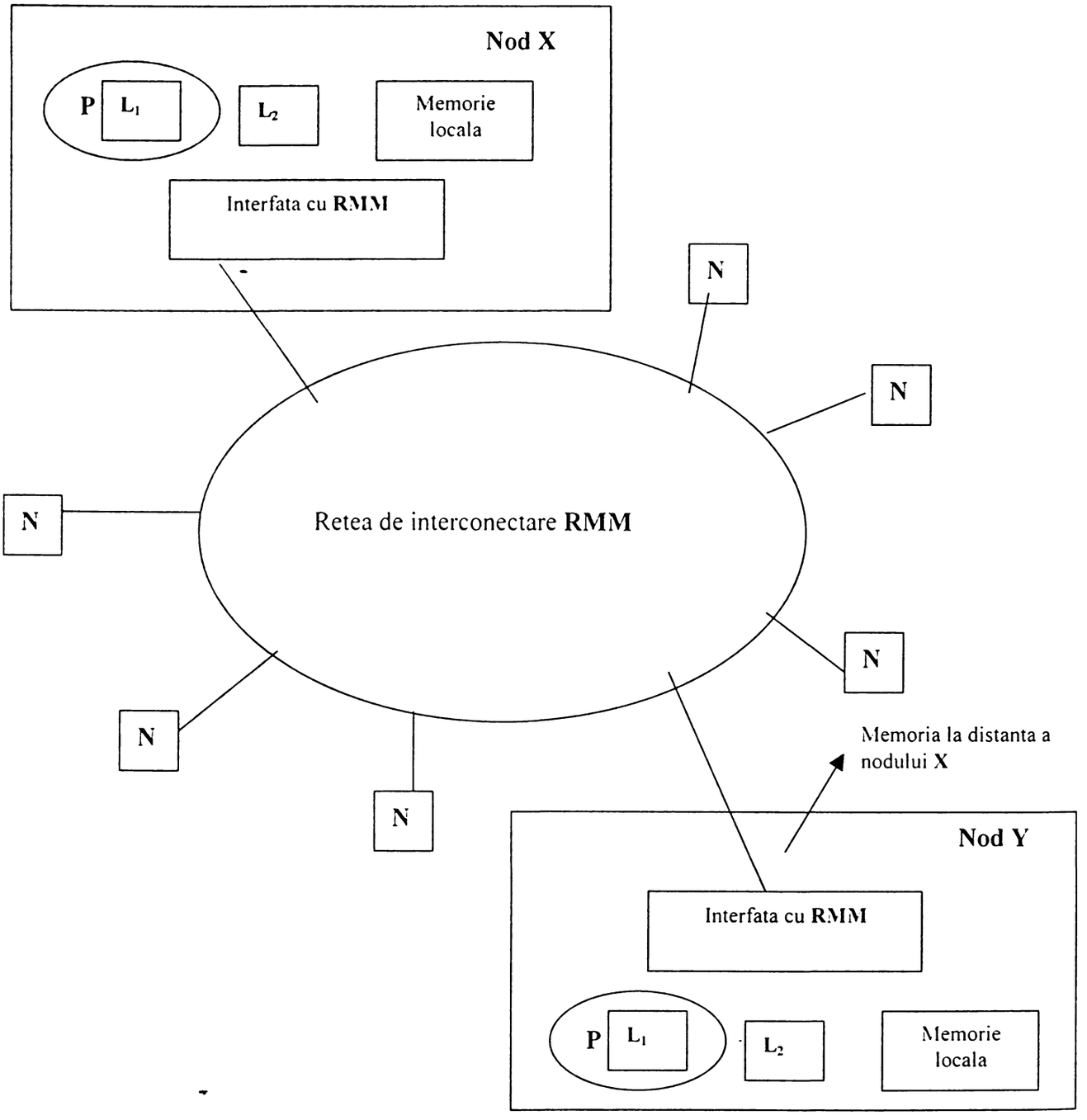


Fig. 6.2 Arhitectura SMP cu evidențierea ierarhiei de memorie

Extinderea ierarhiei memoriei se poate efectua prin plasarea unor capacități suplimentare de memorare (DRAM, SRAM) în fiecare dintre comutatoarele componente ale RIN. Se obține o rețea de interconectare *multinivel cu ierarhie de memorie*, RIN – MIM. Costul unei astfel de soluții constă în complicarea mecanismelor de păstrare a coerenței cache ceea ce impune soluții hardware suplimentare.

Din acestē rațiuni am optat pentru o soluție de compromis între o ierarhie echilibrată de memorie, în termeni de straturi, şi o rețea RMM ce permite circulația bidirecțională a mesajelor şi în acelaşi timp existența unor mecanisme simple de menținere a coerenței.

6.2 STRUCTURA REȚELEI DE INTERCONECTARE DE TIP RMM

În cadrul sistemelor cu arhitectura bazată pe *memorie partajată distribuită* (MPD) modulele de memorie sunt conectate direct la procesoarele corespondente dar spațiul de adresă este partajat. În situația în care RIN este de tip *magistrală ierarhică*, magistrala de pe nivelul superior constituie un potențial de limitare a performanțelor sistemului [BN91], [NB93], [BINK97], [BNA94], [BNT94], [Coif01].

Pentru a ameliora performanța, un număr de magistrale trebuie să fie conectate la nivelul superior utilizând tehnici de intertesere a memoriei. În literatură [BINK 97] se indică o astfel de conectare prezentată în fig. 6.3. Fiecare magistrală impreună cu controlerul asociat se plasează într-un comutator similar cu cel utilizat într-o RIN-MN clasică. Se obține o rețea de interconectare de tip *Rețea Magistrală Multinivel* (RMM).

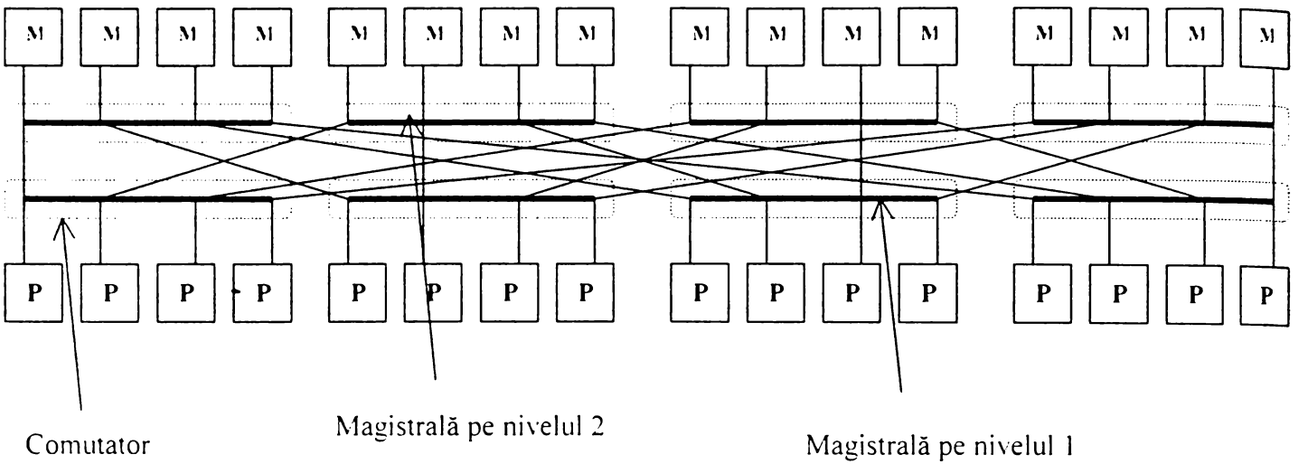


Fig.6.3 Un SMP 16× 16 bazat pe RMM utilizând comutatoare 4x4

În general o RMM de dimensiune $N \times N$, utilizând comutatoare $k \times k$, are $l=\log_k N$ niveluri de comutatoare numerotate de la 0 la l-1. Pe fiecare nivel sunt plasate N/k comutatoare. Fiecare comutator are k porturi superioare şi k porturi inferioare. Interconexiunea dintre două niveluri de magistrale implementează funcția "*k-amestec*".

O astfel de rețea pentru SMP din fig. 6.1 este prezentată în fig. 6.4

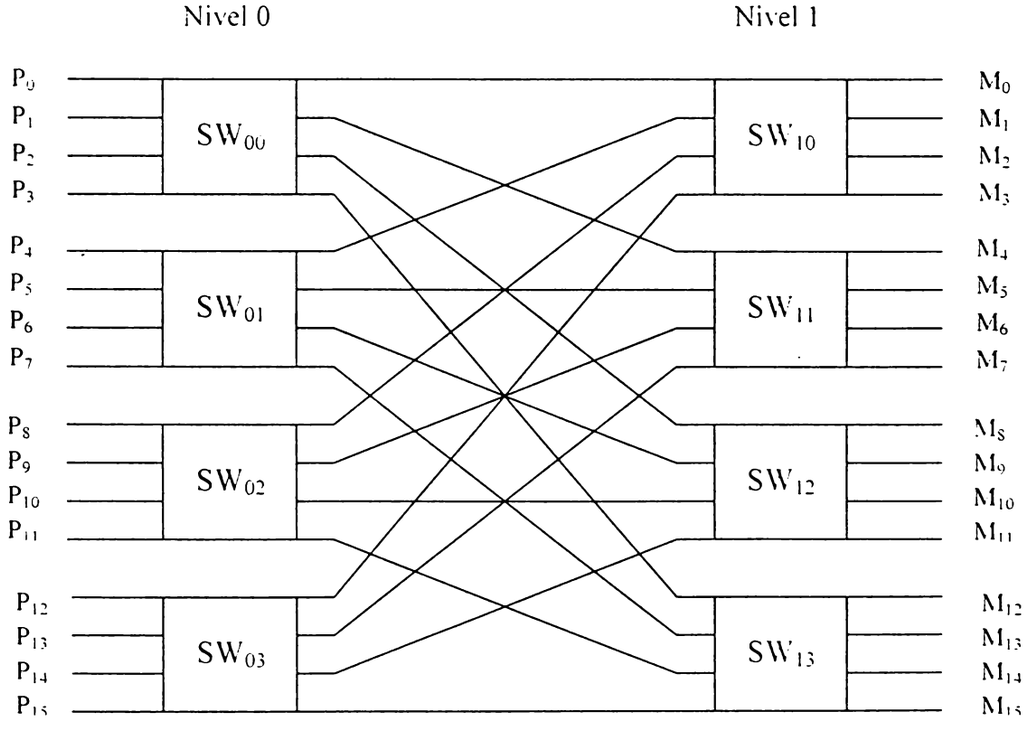


Fig.6.4 RMM 16x16, utilizând comutatoare 4x4

În acest capitol se analizează performanțele unui SMP ce utilizează o RMM 16×16, construită cu comutatoare 2×2. Am optat pentru această soluție deoarece compararea performanțelor acestei rețele față de ale altor RIN-MN este mai facilă atunci când se utilizează comutatoare 2×2. utilizate în majoritatea sistemelor comerciale.

Prin urmare RIN de tip RMM 16×16 este prezentat în fig 6.5

La acest tip de rețea poate să existe sau poate să nu existe o interconectare de tip "*amestec perfect* " înainte de primul nivel de comutatatoare. În primul caz rețeaua este de tip *omega* şi nu face obiectul analizei prezentate în acest capitol. Algoritmii de rutare sunt dezvoltați pe o rețea unde nu există implementată funcția "*amestec perfect* " înainte de primul nivel. Prin urmare, un set de procesoare cu memoriile aferente sunt conectate la un comutator din primul nivel şi la alt comutator dispus pe ultimul nivel.

Dacă există o conectare " *k - amestec perfect* " înainte de primul nivel, atunci la comutatoarele dispuse pe primul nivel şi pe ultimul vor fi conectate un set diferit de procesoare. Indiferent însă de amplasarea conecticii de interconectare sunt posibile patru tipuri de rutări ce vor fi descrise în continuare.

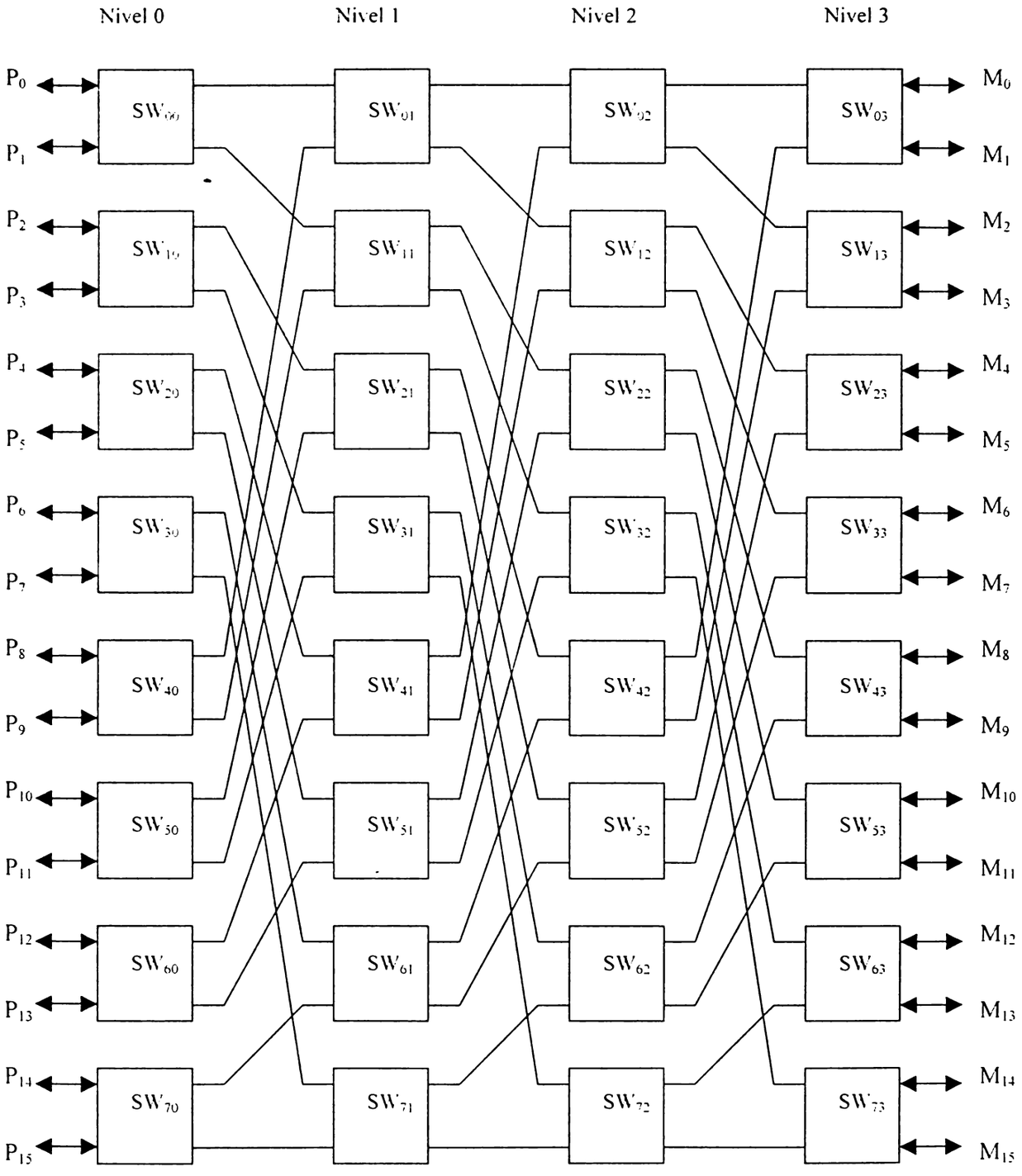


Fig.6.5 RMM 16x16, utilizând comutatoare 2x2

RMM este o rețea bidirecțională, permițind trecerea datelor atît în sens direct cât şi în sens invers. Fiecare comutator are un set de conexiuni plasate în stânga mai aproape de procesoare şi un set de conexiuni plasate în dreapta mai aproape de memorii.

Structura internă a unui comutator 2×2, utilizat în RMM,este prezentată în fig.6.6

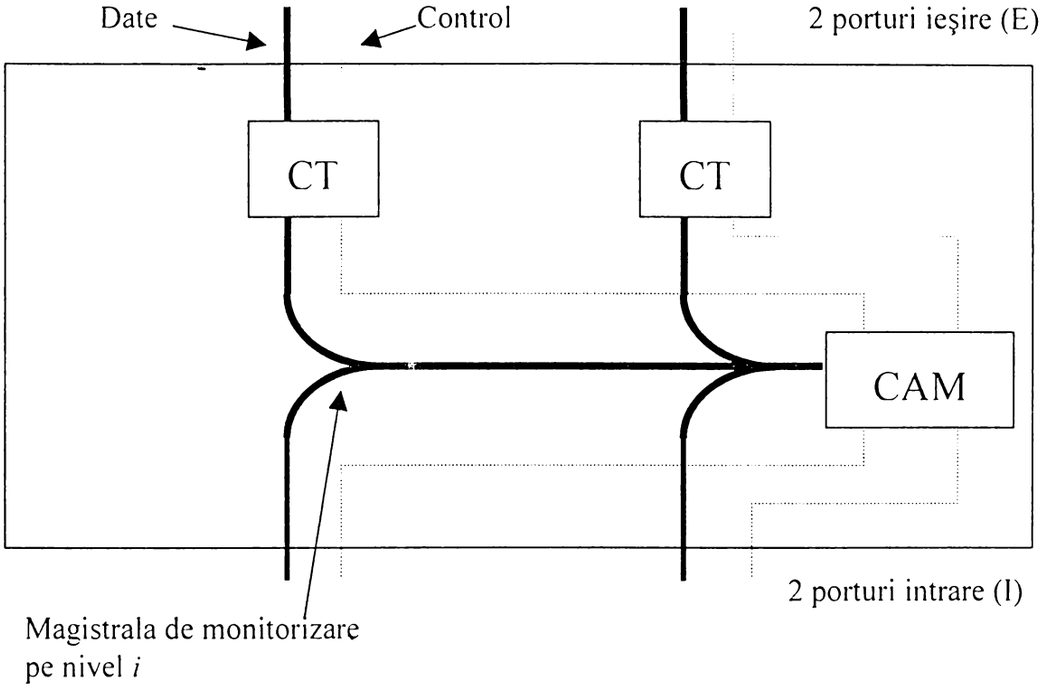


Fig.6.6 Diagrama bloc a unui comutator RMM 2x2

Comutatorul constă din următoarele:

- o magistrală de monitorizare (MM)
- un controler de acces la magistrală (CAM)
- un controler de trafic (CT)

Diagrama bloc a unui CT este prezentată in fig. 6.7. Blocurile CT din componența comutatoarelor RMM păstreaza un director al informației de stare asupra tuturor blocurilor partajate din sistem care l-ar putea parcurge în concordanță cu conexiunile comutatoarelor. Un director nu conține însă nici o informație referitoare la blocurile private.

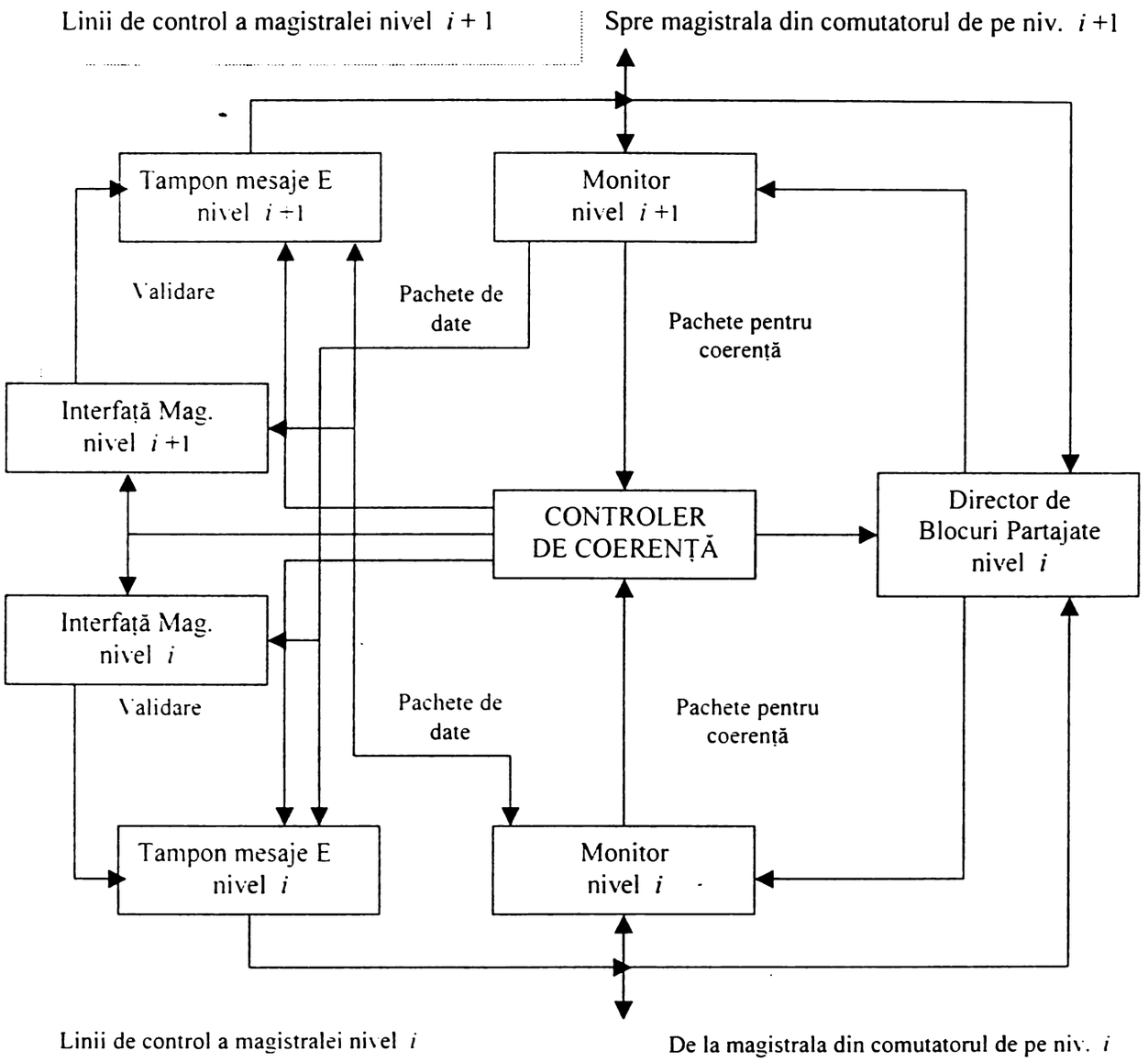


Fig.6.7 Diagrama bloc a unui controler de trafic (CT)

Blocul CT constă dintr-un comutator plasat pe nivelul *i* ce utilizează informația din director pentru a filtra cererile provenind de la magistrala plasată pe nivelul *i*+1 spre magistrala de pe nivelul *i* şi vice versa.

Ieşirile unui CT sunt conectate la magistralele de pe următorul nivel superior utilizând aceeaşi configurație de conectare ca şi o RIN-MN convențională. Accesul la magistrala de monitorizare este comandat de un *controler de acces la magistrala* (CAM). CAM

primeşte semnale reprezentând cereri de acces la magistrală din parte blocurilor CT conectate la magistrală şi validează semnale de garantare a accesului către solicitanți. Există mai multe politici de arbitrare. În general un grup de magistrale plasate pe nivelul *i* este conectat la o magistrală de pe nivelul *i*+1 prin intermediul unor CT individuale plasate pe nivelul *i*. Blocurile CT situate pe nivelul cel mai de sus sunt conectate direct la modulele de memorie corespunzătoare. Controlerele memoriilor cache din nodurile procesoare monitorizează magistralele de pe nivelul *i*. Un procesor lansează o cerere de acces la memorie spre cache-ul aparținător. Dacă este o solicitare pentru o dată privată şi dacă este un succes,atunci blocul de date este furnizat imediat. Dacă este un eşec,atunci solicitarea este avansată spre modulul de memorie corespunzător prin intermediul RMM. Blocul CT manipulează solicitările pentru date private în mod separat aşa că acestea nu fac subiectul acțiunilor de menținere a coerenței.

Pentru blocurile partajate, este permisă coexistența unor copii multiple în diferite memorii cache locale. Coerența acestora este menținută prin protocoale pentru copii multiple. Un astfel de protocol va fi descris ulterior.

Funcționarea diferitelor module dintr-un CT este următoarea. Fiecare CT conține două *module de monitorizare (Snoopers)* (MM). Cele situate pe nivelul *i* şi pe nivelul *i*+1 monitorizează magistralele corespondente. Sarcina lor este supravegherea tuturor tranzacțiilor de pe magistrală şi luarea de măsuri adecvate. Dacă cererea este pentru un bloc privat ce trebuie să parcurgă printr-un anume CT determinat de etichetă,atunci MM plasează cererea *tamponului de mesaje de ieşire* superior. Dacă însă cererea este pentru un bloc partajat atunci MM citeşte directorul pentru a determina dacă trebuie luată o acțiune de menținere a coerenței. Apoi plasează cererea *controlerului de coerență* (CCH), dacă ea este solicitată. Acțiunea specifică de menținere a coerenței depinde de starea blocului din comutatorul respectiv. Un CCH execută acțiunile necesare pentru a implementa un anumit protocol de menținere a coerenței. El modifică directorul CT-ului cum este necesar, corespunzător mesajului de intrare recepționat şi regulilor protocolului. Apoi transmite un mesaj următorului nivel, dacă este solicitat.

Interfețele cu magistrala (IM) gestionează accesele la magistralele respective. Când există un mesaj în coada de aşteptare de ieşire ce trebuie transmis unei magistrale, atunci interfața trimite un semnal de solicitare a magistralei spre CAM şi după ce cîştigă accesul, validează tamponul de ieşire pentru a plasa mesajul pe magistrală.

Din punct de vedere al modelării prin lanțuri Markov, un comutator RMM este reprezentat ca şi în fig. 6. 8.

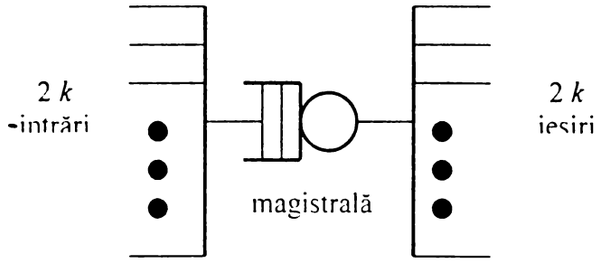


Fig.6.8 Înlănțuirea pachetelor într-un comutator RMM

Comutatoarele RMM pot efectua conectări în sens *direct (înainte), invers (înapoi)* şi cu *buclare*.

În general procesoarele $P_0.P_1,...,P_{N-1}$ sunt conectate la conexiunile din stânga comutatoarelor RMM de pe nivelul 0. Modulele de memorie $M_0,M_1,...M_{N-1}$ sunt conectate la conexiunile din dreapta ale comutatoarelor dispuse pe nivelul l-1. Un modul de memorie M_i este de asemenea conectat la un procesor P_i şi este o *memorie locală* pentru P_i.

O *sursă* este identificată printr-o etichetă sursă $S = s_0\, s_1...s_j...s_{l-1}$, iar o *destinație* printr-o etichetă de destinație $D = d_0d_1...d_j...d_{l-1}$ unde s_j şi d_j sunt cifre în sistemul k de numerație.

Cifrele s_0 şi d_0 sunt cele mai semnificative şi s_{l-1} şi d_{l-1} sunt cel mai puțin semnificative.

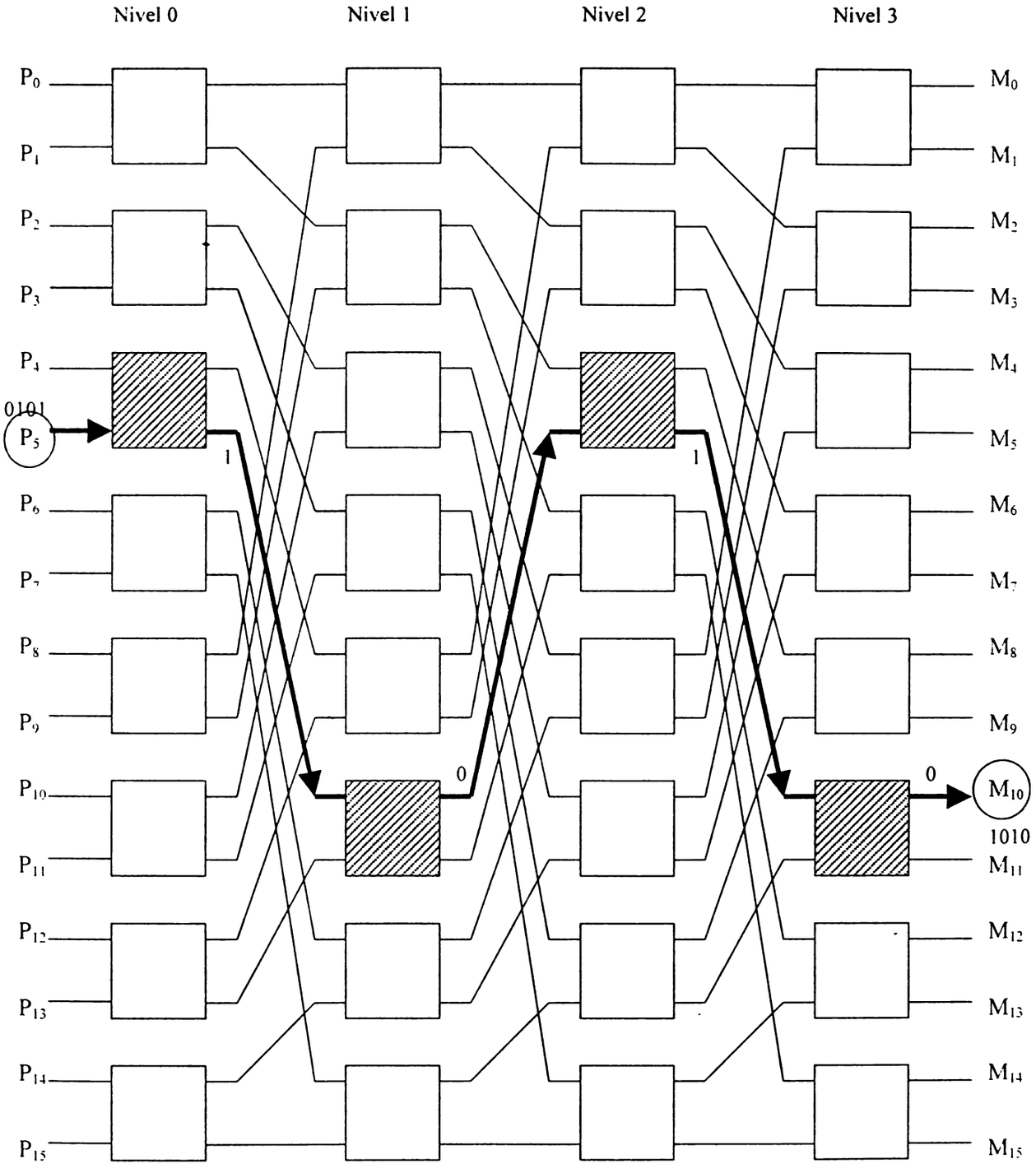
Conectica între niveluri în cadrul unei RMM implementează funcția "*k-amestec*", ceea ce înseamnă că terminalul din dreapta de la poziția $a_0a_1...a_{l-1}$ din nivelul i este conectat cu terminalul din stânga la poziția $a_i...a_{i-1}a_0$ de pe nivelul i +1 pentru i =0,1,...l-2.

O cerere de acces la memorie este satisfăcută intern de către memoria locală când eticheta sursă şi eticheta destinație a unei cereri sunt identice. Dacă etichetele sunt diferite, solicitarea parcurge rețeaua spre o memorie situată la distanță. În fig. 6.5 se prezintă ca şi exemplu o RMM 16×16 cu comutatoare 2×2. Înainte de nivelul 0 nu există implementată funcția " *k-amestec*". Astfel un set de procesoare cu memoriile aferente sunt conectate la un comutator pe primul nivel şi la alt comutator în aceeaşi poziție pe ultimul nivel.

În fig.6.9 o cerere traversează în *direcția înainte* sau *în sens direct* când ea porneşte de la un procesor şi parcurge nivelurile 0,1,...(l – 1) în această ordine.

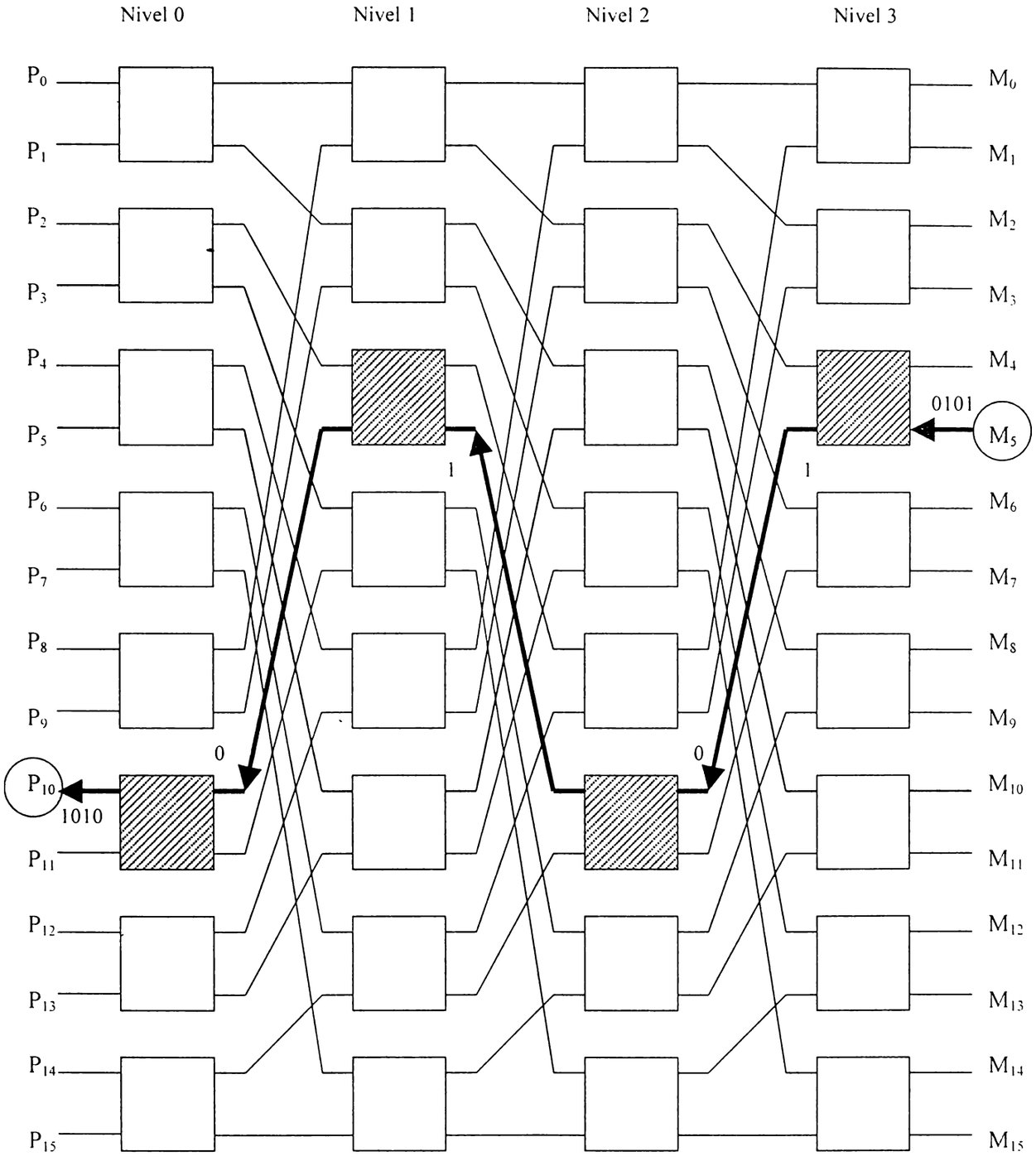
În fig. 6.10 se prezintă *rutarea înapoi* sau *sensul invers* de parcurgere. Cererea porneşte de la memorie şi parcurge nivelurile (l – 1),...1,0 în această ordine. Un pachet poate de asemenea să traverseze de la stânga la dreapta şi să facă o buclă, denumită în formă de U, într-un nivel intermediar ca şi în fig. 6.11 Aceasta este o *buclare în sens direct*. Similar în fig. 6.12 se prezintă o *buclare în sens invers*.

Cele patru tehnici de rutare prezentate anterior asigură patru căi distincte între o sursă şi o destinaţie în RMM. Ca şi rezultat, toleranţa la defect şi fiabilitatea RMM sunt mult mai bune decât în cazul unei RIN-MN convenţionale. În cazul reţelelor convenţionale ca şi Omega, Delta, etc., eticheta destinaţie este utilizată în scopul autorutării unei cereri doar în direcţia înainte - sensul direct. În cazul RMM,eticheta destinaţie poate fi de asemenea utilizată pentru autorutare în sensul direct. Deoarece conexiunile de pe nivelul 0 sunt directe în loc să implementeze funcţia "*k-amestec*", eticheta destinaţie propriu-zisă nu poate fi utilizată pentru autorutare în sensul invers. Eticheta de rutare in cazul rutării în sens invers -RI- se obţine prin aplicarea funcţiei "*amestec invers*" asupra etichetei destinaţie, pe o cifră.



S = 0101 D = 1010 ERD = 1010

Fig.6.9 Rutarea în sens Direct în RMM



S = 0101 D = 1010 ERI = 0101

Fig.6.10 Rutarea în sens Invers în RMM

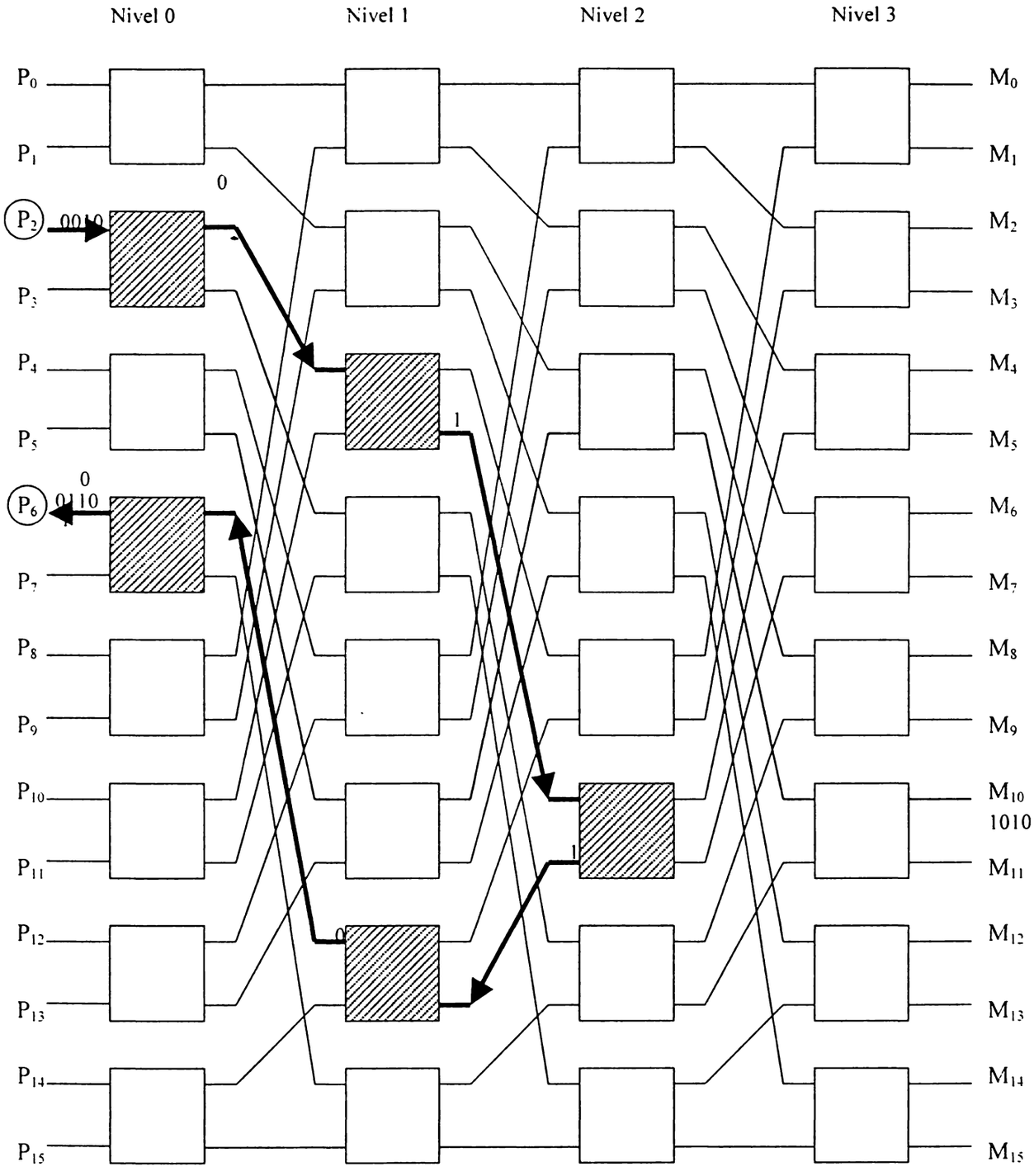
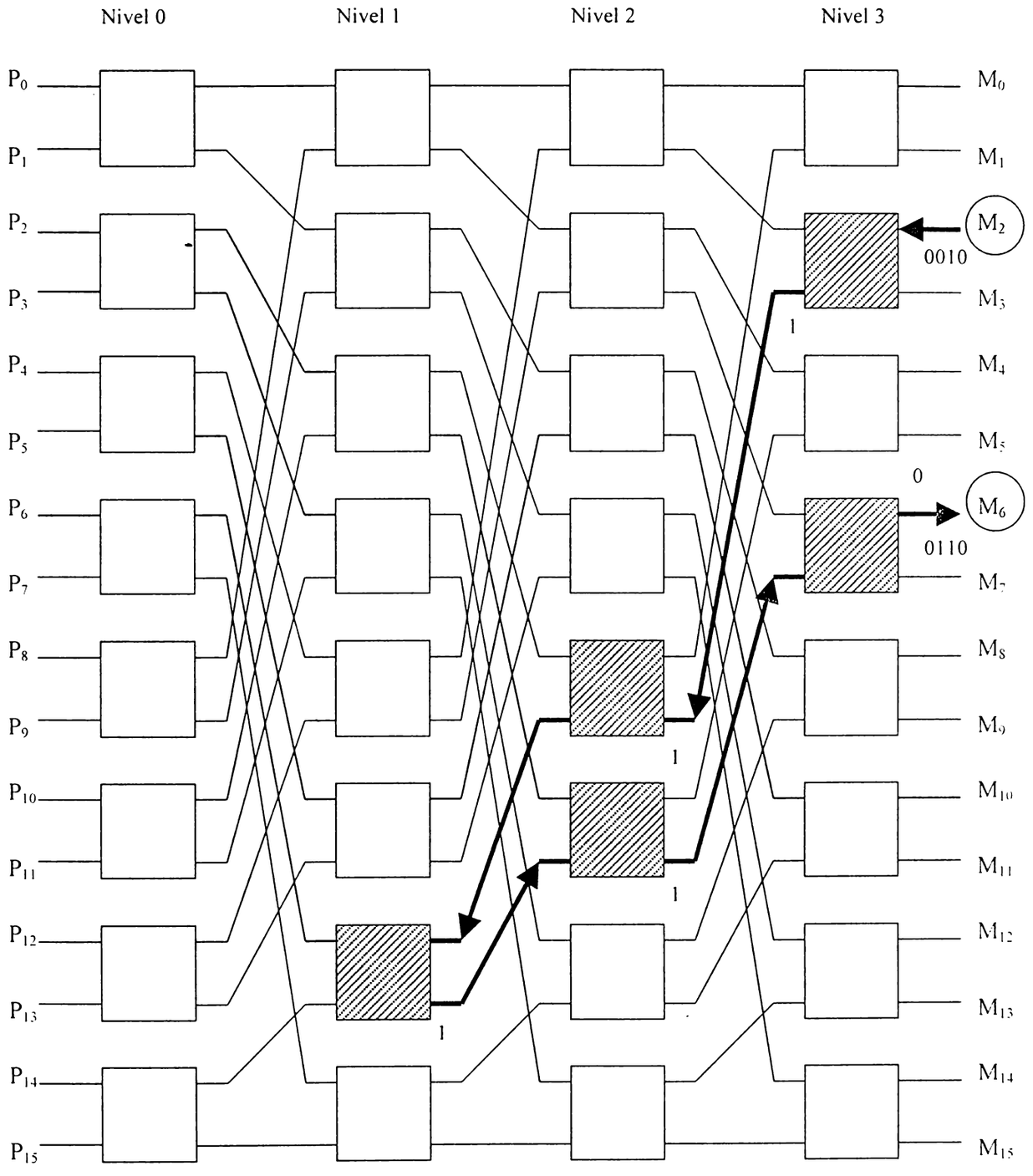


Fig.6.11 Buclare în sens direct (BD)



S = 0010 D= 0110 ERD=0110 ERI=0011
EC=0100 ECR=0010 NII=1

Fig.6.12 Buclare în sens invers (BI)

DEFINIȚIA 6.1 (ERD). *Eticheta de Rutare în sens Direct (ERD) este aceeași ca și eticheta destinație a unei cereri de acces la memorie.*

$$ERD = d_0 d_1 \ldots\ldots d_{l-1}$$

DEFINIȚIA 6.2 (ERI). *Eticheta de Rutare în sens Invers (ERI) este eticheta destinație asupra căreia a operat funcția "amestec" pe o cifră.*

$$ERI = b_0 b_1 \ldots b_{l-1} = d_{l-1} d_0 d_1 \ldots d_{l-2} \text{ unde: } b_j = d_{(j-1) \bmod (l)}$$

DEFINIȚIA 6.3 (EC). *Eticheta Combinată (EC) este eticheta obținută prin aplicarea funcției SAU-EXCLUSIV între cifrele sursă și a celei destinație*

$$EC = c_0, c_1 \ldots, c_j, \ldots, c_{l-1} \qquad \text{unde} \quad c_j = s_j \oplus d_j$$

De notat că în timp ce cifrele în etichetele sursă (S) respectiv destinație (D) sunt în baza k, cifrele din EC sunt în baza 2.

DEFINIȚIA 6.4 (ECR). *Eticheta Combinată Rotită (ECR) este eticheta combinată (EC) asupra căreia s-a aplicat funcția "amestec invers" sau rotită spre dreapta cu o cifră.*

$$ECR = r_0 r_1, \ldots, r_{l-1} = c_{l-1} c_0, c_1, \ldots, c_j, \ldots, c_{l-2} \quad \text{unde} \quad r_j = c_{(j-1) \bmod (l)}$$

DEFINIȚIA 6.5 (NID). *Nivelul de Întoarcere spre sensul Direct (NID) - înainte - este poziția non-zero.*
Aceasta este NID = m,astfel că $r_m = 1$ *și* $r_j = 0$ *pentru* $m < j \leq l-1$.

DEFINIȚIA 6.6 (NII). *Nivelul de Întoarcere spre sensul Invers (NII) -înapoi- este poziția non-zero cea mai din stânga în Eticheta Combinată (EC). Aceasta este NII=n,astfel ca* $c_n = 1$ *și* $c_j = 0$ *pentru* $0 \leq j < n$.

Etichetele de rutare ERD şi ERI sunt utilizate pentru autorutare în cazul direcţiilor *înainte*, respectiv *înapoi*. Etichetele EC şi ECR sunt utilizate pentru a identifica nivelurile U de întoarcere de tip NID sau NII. Nivelurile NID şi NII sunt utilizate pentru a determina unde se produc întoarcerile *înainte* sau *înapoi* în timpul parcurgerilor de tip U.

6.2.1 Algoritmi de rutare pentru RMM.

Există mai multe tehnici de rutare pentru RMM. În literatură [BINK97], [CWH01], [Coif01] se prezintă algoritmi de selectare a *căii cu distanţă minimă* în cadrul unei RMM.

6.2.1.1 Tehnici de rutare

A. Rutare în sens direct (RD)

În această rutare, o solicitare de acces la memorie generată de procesorul S traversează reţeaua de la nivelul 0 până la nivelul l–1 spre memoria de destinaţie D.

Un exemplu de RD de la sursa 0101 la destinaţia 1010 este prezentat în fig. 6.9.

Cifra de indice j a etichetei ERD este utilizată de către un comutator amplasat pe nivelul j, pentru autorutare. De obicei se utilizează comutatoare 2×2 şi atunci cifra are reprezentare binară.

Astfel o cerere de acces ce provine din poziţia $S = s_0s_1...s_j...s_{l-1}$ din stânga nivelului 0 este comutată spre poziţia $s_0s_1...s_j...s_{l-2}d_0$ spre dreapta nivelului 0, apoi traversează o conectică "*k-amestec*" şi ajunge la poziţia $s_1s_2...s_j...s_{l-2}d_0s_0$ la intrarea nivelului 1 şi este comutată la $s_1s_2...s_j...s_{l-2}d_0d_1$ de la ieşirea nivelului 1.

În general, când o cerere soseşte în poziţia $s_js_{j-1}...s_{l-2}d_0d_1...d_{j-1}s_{j-1}$ la stânga nivelului j, ea este comutată spre poziţia $s_j...s_{l-2}d_0d_1...d_{j-1}d_j$ din dreapta nivelului j, parcurge o conectică "*k-amestec*" (excepţie în cazul ultimului nivel, $j = l - 1$) şi soseşte în poziţia $s_{j-1}...s_{l-2}s_{j-1}...s_{l-2}d_0d_1...d_js_j$ din stânga nivelului j+1. În final, ea atinge destinaţia $d_0d_1...d_j...d_{l-1}$ la ieşirea ultimului nivel al RMM.

B. Rutare în sens Invers (RI).

În această tehnică o cerere provenind de la nodul sursă S se deplasează înapoi de la nivelul l–1 prin nivelul 0 spre nodul de destinație D. Un exemplu de RI de la sursa 0101 spre destinația 1010 este prezentat îm fig. 6.10. Cifra de indice j din eticheta ERI este utilizată de către un comutator de pe nivelul j pentru autorutare. Uzual se operează cu bitul j în cazul comutatoarelor 2×2.

O cerere ce a fost lansată din poziția $S = s_0s_1...s_j...s_{l-1}$ la dreapta nivelului l–1, este comutată spre poziția $s_0s_1...s_j...s_{l-2}d_{l-2}$ la stânga nivelului l–1, apoi parcurge o conectică ce implementează funcție "*k-amestec invers*" şi atinge poziția $d_{l-2}s_0s_1...s_j...s_{l-2}$ la dreapta nivelului l–2. În general când o cerere soseşte în poziția $d_jd_{j-1}...d_{l-2}s_0s_1...s_{j-1}s_j$ la dreapta nivelului j. ea este comutată spre poziția $d_jd_{j-1}...d_{l-2}s_0s_1...s_{j-1}d_{(j-1)mod(l)}$. Apoi parcurge o conectică "*amestec invers*", cu excepția ultimului nivel j=0, şi soseşte în poziția $d_{j-1}d_jdj_{-1}...d_{l-2}s_0s_1...s_{j-1}$ din stânga nivelului j–1.

C. Rutare cu Buclare în sens Direct (BD).

Solicitarea porneşte de la sursa S dispusă pe nivelul 0, urmează o rutare RD (utilizând ERD) până la nivelul NID–1 şi atinge partea stângă a nivelului NID în poziția $s_{NID}s_{NID-1}...s_{l-2}d_0d_1...d_{NID-1}s_{NID-1}$.

La NID, schimbă de sens formând o buclă în formă de U, în loc de-a comuta spre dreapta nivelului NID. Solicitarea este comutată spre stânga nivelului NID în poziția $s_{NID}s_{NID-1}...s_{l-2}d_0d_1...d_{NID-1}d_{(NID-1)mod(l)}$ şi urmează o rutare RI (utilizând ERI) până la nivelul 0. În final atinge poziția $d_0d_1...d_{NID-1}s_{NID}...s_{l-2}d_{l-1}$ în stânga nivelului 0. În fig. 6.11 se indică o astfel de rutare pentru 0010 la 0110.

D.Rutare cu Buclare în sens Invers (BI)

În BI, solicitarea este este lansată de sursa S plasată pe nivelul l–1, urmează RI (utilizând ERI) până la nivelul NII+1 şi atinge poziția $d_{NII}d_{NII-1}...d_{l-2}s_0s_1...s_{NII-1}s_{NII}$ la dreapta nivelului NII. La nivelul NII, formează o buclă U, în loc să comute spre stânga lui NII.

Solicitarea este comutată spre dreapta nivelului NII în poziția $d_{NII}d_{NII-1}...d_{l-2}s_0s_1...s_{NII-1}d_{NII}$ şi urmează un traseu RD (utilizând ERD) până la nivelul l–1. În final, solicitarea atinge poziția $s_0s_1...s_{NII-1}d_{NII}d_{NII-1}...\ d_{l-1}$ în dreapta nivelului l–1. Un exemplu de astfel de rutare de la 0010 la 0110 este arătat în fig.6.12.

6.2.1.2a Algoritmul de determinare a numărului de căi din fiecare tip de rutare între surse şi destinaţii

Algoritmul pentru determinarea numarul de căi din fiecare tip de rutare între orice sursă şi destinaţie este un algoritm de tip *backtracking*. Scopul este verificarea şi măsurarea unor proprietăţi şi nu realizarea unei unelte de simulare. Pentru fiecare nod sursă se determina toate căile posibile la fiecare destinaţie folosind RD, RI, BD, BI. Pentru aceasta se explorează întreg spaţiul de stări, selectându-se doar stările valide. Concomitent se determină şi lungimea minimă a fiecarei căi pentru toate perechile sursă-destinaţie. Vectorul de stare este reprezentat printr-o stivă.

```
initializare stiva cu nodul sursa
k=1
LOOP WHILE (k >= 0)
    LOOP
        are_succesor = succesor(k)
        IF (are_succesor) THEN
            este_valid = valid(k)
    UNTIL (NOT are_succesor OR (are_succesor AND este_valid))
    IF (are_succesor) THEN
        IF (solutie(k)) THEN
            inregistrare
        ELSE
            k = k + 1
    ELSE
        k = k - 1
```

succesor(k) determină succesorul nodului din poziţia *k*. Succesorul poate fi unul din cele patru noduri, corespunzătoare direcţiilor posibile de deplasare. Când un nod nu mai are succesor se coboară în stivă la nodul precedent şi se reia explorarea ($k=k-1$).

valid(k) verifică validitatea vectorului parțial de stare curent. Vectorul este valid dacă nu s-au depăşit marginile, dacă nu sunt cicluri, dacă nodul k+1 nu coincide cu nodul k–1, dacă sunt mai multe bucle.

solutie(k) verifică dacă s-a ajuns la o soluție, adică dacă s-a ajuns la un nod destinație. În caz afirmativ se va incrementa contorul de căi în funcție de tipul căii şi totodata se va reține lungimea căii minime. Dacă nu s-a ajuns încă la o soluție se contiuă algoritmul pentru următorul nod (k=k+1).

Programul de implementare al acestui algoritm şi rezultatele sunt prezentate în ANEXA 1, ANEXA 2 şi în [CWH01].

6.2.1.2 b Algoritmul de stabilire a căii optime

Distanța dintre o sursă şi o destinație în RMM se defineşte ca fiind numărul minim de comutatoare prin care un pachet de date trebuie să traverseze rețeaua.

Pentru o RIN-MN convențională, distanța este întotdeauna l. În cazul unei RMM, distanța poate fi mai mică decât l dacă se aleg tehnici de rutare BD sau BI. Acestea sunt utilizate atunci când nivelul din rețea unde se produce întoarcerea - buclarea- este mai mic decât nivelul central al rețelei. Prin urmare se obțin beneficii nete în termeni de distanță între o sursă dată şi toate destinațiile.

Un algoritm pentru a selecta rutarea optimă pentru o pereche sursă - destinație dată, este următorul:

1. $S = s_0 s_1 \ldots s_{l-1}$
2. $D = d_0 d_1 \ldots d_{l-1}$
3. $EC = S \oplus D = c_0, c_1 \ldots, c_j, \ldots, c_{l-1}$
4. $ECR = c_{l-1}, c_0, c_1 \ldots, c_j, \ldots, c_{l-2}$
5. $d_l = \lfloor l/2 \rfloor$, $d_u = \lceil l/2 \rceil$
6. **IF** (sursa = destinația)
7. **THEN** cererea de acces este pentru memoria locală
8. **ELSE**
9. Caută NID şi NII, în funcție de ECR şi EC
10. **IF** ($NID = (l - 1 - NII) = 0$)
11. **THEN** selectează rutarea BD sau rutarea BI

12. **ELSE IF** ($NID < d_l$)
13. **THEN** selectează rutarea BD
14. **ELSE IF** ($NID \geq d_u$)
15. **THEN** selectează rutarea BI
16. **ELSE**
17. selectează rutarea RD sau RI

Algoritmul de stabilire a căii optime selectează un traseu care are o lungime minimă de cale în termeni de număr de comutatoare traversate.

Fiind dată o sursă $S = s_0 s_1 \ldots s_{l-1}$ şi o destinaţie $D = d_0 d_1 \ldots d_{l-1}$ acest algoritm calculează etichetele descrise anterior. Se compară apoi etichetele pentru a decide care din cele patru tehnici de rutare va oferi lungimea minimă prin reţea. În acest algoritm d este definit ca şi nivelul central a RMM. Trebuie remarcat că rutarea optimă între două noduri este fixată într-o reţea dată. Astfel, calea optimă poate fi precalculată şi memorată într-un tabel ce poate fi citit când se generează o cerere. Nu este necesar ca acest algoritm să fie executat ori de câte ori un mesaj este extras.

Dacă sursa şi destinaţia sunt identice, atunci cererea este pentru momoria locală. În acest caz, nu se solicită nici o traversare prin RMM. Toate celelalte cereri traversează cel puţin un nivel din RMM.

Memoriile ce sunt conectate la un procesor prin primul sau ultimul nivel al RMM sunt denumite *memorii de grup (cluster memories)*. Similar,procesoarele ce au cel puţin un comutator între ele şi memorii formează *procesoare de grup (cluster processors)* al acestor memorii. Un grup de procesoare conectat la un comutator de pe nivelul 1, îndeplineşte această condiţie. Cererile de acces la memoriile unui grup necesită ca doar un comutator să fie traversat. Astfel când ($NID = (l - 1 - NII)) = 0$, se optează pentru rutări BD sau BI.

Dacă NID $< \lfloor l/2 \rfloor$, sau NII $\geq \lceil l/2 \rceil$, există niveluri de întoarcere în RMM înaintea sau după nivelul din centrul RMM.

Aceasta va reduce lungimea totală a căii la mai puţin de l şi astfel rutăriile BD sau BI sunt selectate. Dacă nici una din aceste condiţii nu sunt îndeplinite,atunci NID $\geq \lceil l/2 \rceil$, şi NII $< \lfloor l/2 \rfloor$. În acest caz rutările RD sau RI sunt singurele opţiuni posibile.

Lungimile de cale, în termeni de număr de comutatoare traversate, sunt următoarele:

- Memorie locală : 0 comutatoare (RMM nu e traversată)
- Rutare RD sau RI : l comutatoare
- Rutare BD:

  -memorii grupate : 1 comutator

  -alte memorii : $2 \times NID + 1$ comutatoare
- Rutare BI :

  -memorii grupate : 1 comutator

  -alte memorii : $2 \times (l - 1 - NII)+1$ comutatoare

În ANEXA 1 este prezentat un program scris în limbajul de programare C ce determină numărul de căi pentru fiecare tip de rutare între toate sursele şi toate destinaţiile. De asemenea acest program calculează lungimile minime ale căilor stabilite între surse şi destinaţii. S-au luat în considerare succesiv RMM 16X16, 256X256 şi, respectiv, 1024X1024.

În ANEXA 2 sunt prezentate tabele ce conţin numărul de căi între o sursă dată şi toate destinaţiile pentru o RMM 16 X 16, luând în considerare toate cele patru tipuri de rutări. Sunt prezentate de asemenea tabele ce afişează lungimile minime ale căilor pentru rutările RD, RI, BD şi BI.

În ANEXA 3 este prezentat programul de determinare a căii optime între o sursă şi o destinaţie dată în RMM.

În TABEL XI sunt prezentate aceste lungimi pentru o sursă şi o destinaţie dată. S-a considerat o RMM 1024×1024[BINK97], [CWH01].

TABEL XI

Lungimile de cale pentru fiecare din rutări, considerând s = 0 şi diferite destinaţii

Destinaţia i	Rutări RD/RI	Rutare BD	Rutare BI
514	10	19	19
256	10	5	17
2	10	19	3
72	10	19	13

6.2.2 Analiza de performanță a RMM

Se analizează comparativ comportamentul din punct de vedere al performanțelor a două rețele de interconectare, RMM şi respectiv RIN-MN *bidirecțional*, abreviat RIN-MN(B). Ele sunt plasate în cadrul aceluiaşi mediu cu memorie partajată distribuită, adică SMP din fig. 6.1.

În ambele cazuri, modulul de memorie M_i este direct conectat la procesorul P_i şi este denumit *memorie locală* a lui P_i. Cererile de la un procesor către memoria sa locală sunt denumite *cereri interne* şi sunt extrase prin magistrala internă dintre procesor şi memoria sa locală. O memorie poate deasemenea recepționa *cereri externe* ce sunt generate de alte procesoare şi traversează rețeaua RMM sau RIN-MN (B).

Un *grup de procesoare (cluster processors)* asociate unui procesor P sunt acele procesoare situate la distanță de un comutator în raport cu P. Într-un SMP-MPD, există $k-1$ procesoare ce pot fi accesate prin comutatorul de pe primul nivel sau de pe ultimul la care P este conectat.

Prin urmare o *cerere externă* destinată unui procesor dintr-un grup sau unei memorii dintr-un grup se întoarce din primul nivel sau din ultimul, fără a parcurge întreaga RMM. Cu alte cuvinte *cererile externe* lansate de procesoare parcurg RMM de la stânga spre dreapta în timp ce răspunsurile de la memorii traversează rețeaua de la dreapta spre stânga.

Ambele *cereri interne* şi *externe* sosesc la coada de aşteptare formată la memorie. Disciplina de deservire a acestor cereri este FIFO.

6.2.2.1 Funcționarea rețelei

În analiza funcționării rețelei se fac următoarele ipoteze de lucru:

- Sistemul este *sincron* şi *comutat prin pachete*
- Tamponul pentru pachete are *capacitatea infinită*
- *Timpul de serviciu al magistralei*, în cazul RMM, sau *timpul de serviciu al legăturii*, în cazul RIN-MN(B), pentru a transfera un mesaj, formează un singur ciclu sistem
- Un procesor este reprezentat printr-un *centru de întârziere*, într-un ciclu dat

- *Timpul de serviciu al modulelor de memorie* reprezintă multiplii întregi ai *timpului ciclu de sistem.*

Odată ce transmite cererea de acces la memorie, procesorul aşteaptă până când obţine pachetul de răspuns de memorie (în caz de citire) sau de confirmare (în caz de scriere)

Se definesc următorii parametrii de sistem:

$k \times k$:	talia comutatoarelor RMM sau RIN-MN
N	număr de procesoare sau memorii în SMP-MPD
L	$\log_k N$ nivele în RIN
t_s	timp de serviciu al comutatorului
t_m	timp de serviciu al memoriei
p	probabilitatea ca un procesor să plaseze o cerere de acces la memorie
m	probabilitatea ca un procesor să solicite memoria sa locală când s-a făcut o cerere de acces
m_1	probabilitatea ca un pachet să fie destinat unui procesor din grup.
p_i	probabilitatea ca o cerere să traverseze prin nivelul i
r_i	timp mediu de răspuns a unui comutator de pe nivelul i
q_l	număr mediu de cereri locale de la un procesor, în cadrul unui ciclu
q_e	număr mediu de cereri la distanţă de la un procesor, în cadrul unui ciclu
d_n	întârziere totală în reţea considerând toate nivelurile
l_m	lungime medie a firului de aşteptare în modulul de memorie
d_m	întârziere medie într-un modul de memorie
P_u	utilizare a procesorului (fracţiune din timpul în care procesorul este ocupat)

Analiza performanţei atât a RMM cât şi a RIN-MN(B) se bazează pe următoarele supoziţii :

- Pachetele sunt generate în fiecare nod sursă de către procese aleatoare independente şi identic distribuite.
- În orice moment, un procesor este fie ocupat executând calcule interne, fie este în aşteptare pentru un răspuns în urma unei cereri de acces la memorie.
- Fiecare procesor ocupat,dacă nu există cereri ce aşteaptă să fie rezolvate,generează un pachet cu probabilitatea p la începutul fiecărui ciclu.

- Probabilitatea ca această cerere să fie adresată unei memorii locale (*cerere internă*) este m şi probabilitatea ca ea să fie adresată altui modul(*cerere externă*) este $(1-m)$
- O *cerere externă* se va adresa unei memorii dintr-un grup cu probabilitatea m_1 şi unui alt modul de memorie, exterior grupului, cu probabilitatea $1-m_1$
- Un răspuns de la memorie parcurge reţeaua în direcţia opusă,pe acelaşi traseu,în cazul reţelelor RMM,respectiv RIN-MN(B).

Mesajele de la procesor spre memorie sunt generate utilizând probabilităţi specificate astfel :

- *Probabilitatea de cerere a accesului (p)* : Probabilitatea de cerere a accesului este utilizată ca şi un mijloc de estimare a comportamentului procesorului în termeni de cereri de acces la memorie. Când un procesor este ocupat în procesul de calcul,el poate genera o cerere. La fiecare ciclu procesorul decide dacă un mesaj poate fi sau nu transmis cu această probabilitate. În medie, sunt necesari $1/p$ cicluri pentru a extrage o cerere din procesor.
- *Probabilitatea de cerere a accesului la memoria locală (m)* : Fiind cunoscut că o cerere trebuie lansată către memorie, se utilizează o probabilitate m pentru a se decide dacă această cerere este pentru memoria locală sau pentru cea externă.

Deşi simple,aceste probabilităţi au un rol important fiind singurele intrări în analiza de performanţă. După fiecare cerere de acces la memorie, procesorul aşteaptă pentru o confirmare. Odata ce această confirmare a fost recepţionată, procesorul efectuează calcule pe durata unui ciclu şi apoi, bazat pe probabilităţile de mai sus, decide dacă va genera sau nu altă cerere de acces la memorie.

Utilizarea procesorului : *Utilizarea procesorului*, P_u, definită ca şi fracţiunea de timp în care un procesor este ocupat, va fi determinată de *timpul de aşteptare* şi de *timpul de serviciu* necesare satisfacerii unei cereri de acces în diverse puncte.

În cazul în care rutările de tip BD şi BI vor fi permise doar în primul şi ultimul etaj şi toate celelalte cereri vor fi rutate direct, utilizarea procesorului este :

$$P_u = 1/\{1+p(1-m)(1-m_1)(d_n+d_m)+pmd_m+p(1-m)m_1(2r_0+d_m)\} \qquad (6.1)$$

În cazul studiat,un mesaj în RMM sau în RIN-MM(B) va fi expediat de-a lungul unui traseu minim. În acest caz

$$P_u = \frac{1}{1+\alpha+\beta+\gamma+\eta} \tag{6.2}$$

unde :

α : întârzierea aşteptată pentru ca o cerere locală să fie deservită

β : întârzierea aşteptată pentru ca o cerere la grupul de memorii să fie deservită

γ : întârzierea aşteptată pentru a deservi toate cererile,cu excepţia memoriilor grupate ce urmează rutarea RD sau RI

η : întârzierea aşteptată pentru a deservi toate cererile ce urmează o rutare RD sau RI

Termenii $\alpha, \beta, \gamma, \eta$ depind de :

- Probabilităţile de rutare de-a lungul fiecărei căi
- Intensitatea traficului de reţea
- Solicitarea de prelucrare a cererii în fiecare centru de deservire

Puterea de procesare, definită ca şi numărul mediu de procesoare ce rămân ocupate, este NP_u

În scopul de a compara eficacitatea comutatorului RMM sau RIN-MN(B) se defineşte un parametru de cost. O soluţie simplă de a defini *costul* unui comutator este reprezentată de *numărul de conexiuni* în comutator. Numărul de conexiuni este k^2 pentru un *crossbar* în timp ce pentru un RMM este $2k$. Într-un SMP-MPD costul total al RIN-MN(B) este $k N \log_k N$ iar pentru RMM este $2N \log_k N$.

Eficacitatea costului reţelei este definită ca fiind raportul dintre *puterea de procesare* şi *costul* reţelei. Eficacitatea costului va indica performanţa ce poate fi atinsă pe unitatea de cost a reţelei.

6.2.2.2 Probabilităţi de rutare şi întârzierile în parcurgerea căii.

Probabilităţile de rutare şi întârzierile în parcurgerea unei căi, atât pentru RMM cât şi pentru RIN-MN(B), se obţin presupunând că toate memoriile non-locale sunt adresate în mod egal de către un procesor [BINK97]. Aceste ecuaţii pot fi modificate în cazul referirilor neuniforme la memorii situate la distanţă [IB00]. [BIK00]. Deoarece lungimea căii în rutarea RI este aceeaşi ca şi rutarea RD, se extrage η bazat pe RD şi se multiplică cu 2 pentru a include şi rutarea RI. Similar se procedează şi pentru rutările BD şi BI.

În continuare definim :

Cereri de acces la memoria locală (α) : O cerere de acces la memoria locală nu implică traversarea comutatorului. Prin urmare singura întârziere apare în deservirea cererii în modulul de memorie şi ea este d_m.

Prin urmare fiind date p şi m, anterior definite, se deduce :

$$\alpha = p \times m \times d_m \tag{6.3}$$

Rutarea de grup (β) : Cererile la procesoarele asociate într-un grup traversează reţeaua spre un comutator din primul sau din ultimul nivel şi sunt rutate BD sau BI spre procesorul de destinaţie. Toate perechile sursă-destinaţie de acest tip, unde toţi biţii cu excepţia celor mai puţin semnificativi $\log_2 k$ ai etichetei EC sunt 0, atrag acest tip de rutare. Astfel numărul memoriilor asociate în grup pentru o sursă dată este k-1 deoarece $k \times k$ este dimensiunea unui comutator din RMM sau din RIN-MN(B). Comutatorul dispus pe nivelul 0 este traversat odată pentru accesarea memoriei din grup
şi odată pentru a retransmite confirmarea. Când o memorie externă este solicitată, probabilitatea ca memoriile unui grup să fie solicitate este :

$$\beta_p = \frac{k-1}{N-1} \tag{6.4}$$

Considerând $2 \times r_0$ este întârzierea / comutator şi că d_m este întârzierea în deservirea memoriei, se obţine :

$$\beta = p \times (1-m) \times \frac{k-1}{N-1} \times (2r_0 + d_m) \tag{6.5}$$

Rutarea BD sau BI pentru memoriile negrupate (γ) : În rutarea BD şi BI, cererea parcurge reţeaua într-o direcţie până la un nivel anume şi apoi face o buclă de întoarcere pentru a ajunge la procesorul de destinaţie. Cunoscând nivelul NID unde se produce buclarea, lungimea căii este $2 \times NID + 1$.

NID este traversat doar odată în timp ce toate nivelurile din stânga lui NID sunt traversate de două ori,dar nu în mod necesar prin acelaşi comutator. Deoarece întotdeauna există memorii grupate acoperite ($NID=0$, $NII=l+1$) se va începe procedura de optimizare cu $NID \geq 1$ şi $NII \leq l+2$.

Se consideră $NID < \lfloor l/2 \rfloor$. O dezvoltare similară se poate face pentru $NII \geq \lceil l/2 \rceil$

Se cunoaşte că numărul total de destinaţii este N-1. Pentru un nivel dat de întoarcere $1 \leq i < d$, deoarece NID este definit ca şi cel mai din dreapta bit din etichetă, toţi biţii din stânga acestei poziţii pot fi 1 sau 0. Această constatare conduce la concluzia că pot exista k^i căi de parcurgere în reţea. Numărul de căi în care un bit din ECR poate fi 1 este k-1. Atunci când o memorie externă este solicitată, ecuaţia pentru probabilitatea unei rutări BD sau BI pentru memorii grupate este :

$$\gamma_p = 2 \times \left(\sum_{i=1}^{d-1} \frac{k-1}{N-1} \times k^i \right) \tag{6.6}$$

unde d = $\lfloor l/2 \rfloor$.

Întârzierea într-o astfel de rutare este dependentă de nivelul în care se produce buclarea. Întârzierea pentru toate comutatoarele de pe un traseu dat, cu excepţia celei din nivelul de buclare este :

$$\Delta_1 = 2 \times \sum_{i=0}^{i-1} 2r_i$$

Deoarece trebuie luat în considerare faptul că nivelul de întoarcere este traversat de asemenea şi de cererea de acces şi de confirmare cu întârzierea $r_i + d_m$ se obţine :

$$\Delta_2 = 2 \times \left(\sum_{i=0}^{i-1} 2r_i + r_i \right) + d_m$$

Ecuația de determinare a lui γ este :

$$\gamma = p \times (1\text{-}m) \times 2 \times \left\{ \sum_{i=1}^{d-1} \frac{k-1}{N-1} \times k \times \left[2 \times \left(\sum_{j=0}^{i-1} 2r_j + r_i \right) + d_m \right] \right\} \qquad (6.7)$$

Rutarea RD (η) : Această tehnică se utilizează în cazul tuturor acelor perechi sursă - destinație care nu cad sub incidența categoriilor descrise anterior. Deoarece atât rutarea RD cât şi cea RI reprezintă o ultimă opțiune pentru orice alt tip de perechi sursă - destinație, se poate exprima :

$$\eta = (1 - \beta - \gamma) + d_n$$

În acest tip de rutare toate comutatoarele sunt traversate, ceea ce înseamnă :

$$d_n = \sum_{i=0}^{i-1} r_i$$

Prin urmare, întârzierea aşteptată pentru toate aceste rutări este :

$$\eta = p \times (1 - m) \times \eta_p \times \left[\left(\sum_{i=0}^{i-1} r_i \right) + d_m \right] \qquad (6.8)$$

unde:

$$\eta_p = 1 - \beta_p - \gamma_p \qquad (6.9)$$

iar β_p, γ_p sunt date de (6.4) şi respectiv de (6.6)

Ecuațiile de la (6.1) la (6.9) sunt valide când memoria locală este accesată cu o probabilitate m şi toate celelalte memorii sunt adresate cu probabilități egale, adică $(1-m)/(N-1)$.

TABEL XII prezintă numărul de destinații ce pot fi atinse pornind de la P_0 cu fiecare dintre rutări ca şi o funcție de talia rețelei. Talia unui comutator este 2 ×2. Analizând rezultatele din tabel, se observă că :

- un număr semnificativ de conexiuni beneficiază de alte tipuri de rutări decât cele de tip RD sau RI, utilizate în mod curent

- aceleaşi număr de procesoare utilizează rutări BD şi BI în două reţele succesive ca şi dimensiuni.

De exemplu se consideră dimensiunile 64, respectiv 128, ale reţelei. Aceasta implică l=6 şi l=7. Reţelele, deşi au dimensiuni diferite, au acelaşi număr de destinaţie pentru rutare BD deoarece adăugarea unui nivel introduce un nivel central autentic în cazul $l = 7$. Acesta este nivelul 3. În situaţia $l = 6$ nu există un centru autentic al reţelei. Adăugarea unui nivel central nu va creşte numărul posibil de rutări BD sau BI.

TABEL XII

Numărul de destinaţii pentru diferite dimensiuni ale reţelei utilizând diferite rutări.

Rutare	***Dimensiune* RMM**				
	64	**128**	**256**	**512**	**1024**
Rutare BD sau BI $(N-1) \times (\beta_p + \gamma_p)$	13	13	29	29	61
Rutări RD sau RI $(N-1) + (\eta_p)$	50	114	226	482	962

Întârzierile r_0, r_1, d_n şi d_m vor depinde de :

- valoarea traficului în reţea, care este o funcţie de P_u
- cererea de deservire la centrele individuale de deservire

6.2.2.3 Întârzierile în formarea cozilor de aşteptare din comutatoare.

Pentru a simplifica analiza, fiecare nivel din reţea este considerat izolat de celelalte nivele. Se consideră un centru de formare a cozilor de aşteptare cu n intrări.

Fie probabilitatea q că există cel puţin un pachet la una dintre intrări în fiecare ciclu dat şi că cererea de deservire a unui pachet este t cicluri. Numărul de cereri ce vin în coadă în timpul de deservire a oricărei cereri anterioare va forma o distribuţie binominală având următoarele caracteristici :

nt - numărul de încercări

q - probabilitatea de succes

Numărul mediu al cererilor ce sosesc este $E = ntq$ iar *varianţa* este $V = ntq\ (1 - q)$

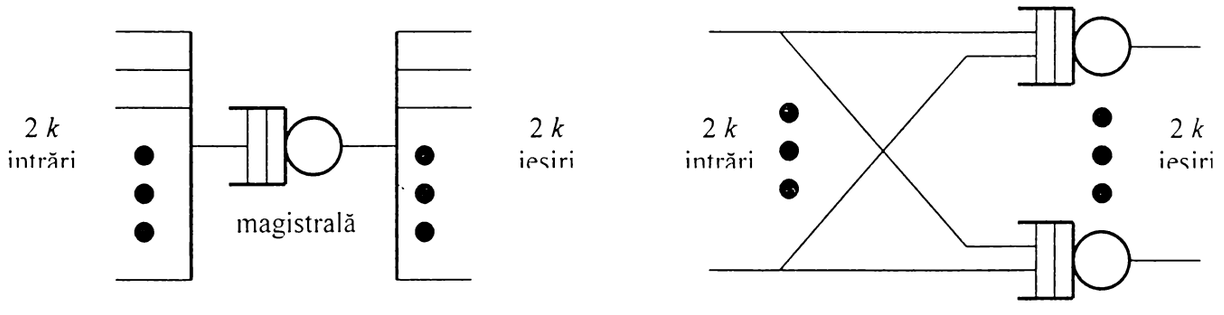
În literatură [BINK97], se prezintă metoda de calcul a *lungimii medii a cozii de aşteptare* Q în cadrul unei staţii de deservire.

$$Q = \frac{E}{2} + \frac{V}{2(1-E)} \tag{6.10}$$

Capacitatea de trecere a acestor cereri este E / t. *Timpul de răspuns mediu* al centrului este r, calculat prin aplicarea legii lui Little, devine:

$$r = \frac{Q.t}{E} = \frac{1}{2}\left(1 + \frac{1-q}{1-ntq}\right)t \tag{6.11}$$

Modelele de formare a cozilor de aşteptare a unui comutator RMM şi a unui comutator RIN - MN (B) sunt prezentate în fig. 6.13



a) Structura cozii de aşteptare în comutatorul RMN

b) Structura cozii de aşteptare în comutatorul RIN-MN (B)

Fig. 6.13 Cozi de aşteptare în comutatoarele RMN şi RIN-MN (B)

În cadrul unui comutator RMM există o competiţie între pachete pentru câştigarea accesului la magistrală. Aceste pachete provin, pe de o parte, de la cele k porturi din dreapta comutatorului, iar pe de altă parte, de la k porturi din stânga. Pentru comutatoare dispuse pe niveluri, altele decât primul şi ultimul, $n=2k$, $t=t_s$ şi $q=q_x = P_u p\,(1-m)(1-m_1)$. Se poate calcula *timpul mediu de răspuns* al comutatorului, r_{RMMs}, pentru oricare dintre aceste comutatoare RMM.

Acesta este :

$$r_{RMMs}=\frac{1}{2}\left(1+\frac{1-q_s}{1-2kt_sq_s}\right)t_s \tag{6.12}$$

Pentru comutatoare dispuse pe primul şi pe ultimul nivel, $n=2k$, $t=t_s$ şi

$q=q_s=P_{ul}p(1-m)(2-m_1)/2$

Timpul mediu de răspuns, r_{RMMi}, este:

$$r_{RMMi}=\frac{1}{2}\left(1+\frac{1-q_s}{1-2kt_sq_s}\right)t_s \tag{6.13}$$

Întârzierea reţelei RMM, d_n, va fi suma timpilor de răspuns a nivelurilor traversate de un pachet în timp ce este rutat prin reţea.

$$d_n=2((\log_k N-2)r_{RMMs}+2r_{RMMi}) \tag{6.14}$$

În cazul reţelei RIN-MN(B),într-un comutator există $2k$ intrări şi $2k$ ieşiri. Probabilitatea de apariţie a unei cereri la o intrare sau la o ieşire a unui comutator RIN-MN(B) dispus pe nivelul i va fi $P_{ul}p(1-m)p_i$.

Timpul de răspuns a unei RIN-MN(B) se calculează utilizând $n=2k$, $t=t_s$ şi

$q=P_{ul}p(1-m)p_i/n$.

Acesta este:

$$r_{RMBi}=\frac{1}{2}\left(1+\frac{1-\frac{q}{n}}{1-2kt_s\frac{q}{n}}\right)t_s \tag{6.15}$$

Întârzierea totală din reţea este d_n şi reprezintă suma timpilor de răspuns ale comutatoarelor din diferite niveluri.

În ambele tipuri de rețele, numărul mediu de cereri sosind la un modul de memorie este $E_m=q_i+t_m q_e$, unde q_i şi q_e sunt cereri interne şi externe pentru acel modul de memorie.

Varianța $V_m=q_i\,(\,1-q_i\,)+t_m q_e(1-q_e)$.

Astfel *lungimea medie a cozii de aşteptare la memorie* este :

$$l_m=\frac{q_i+t_m q_e}{2}+\frac{q_i(1-q_i)+t_m q_e(1-q_e)}{2(1-q_i+t_m q_e)} \tag{6.16}$$

iar *timpul de răspuns mediu al memoriei* este :

$$d_m=\frac{l_m t_m}{E_m} \tag{6.17}$$

Ecuația din care se determină P_u este neliniară. Ea se rezolvă utilizând tehnici de iterație.

O tehnică de iterație pentru a determina P_u este următoarea :

1. Se inițializează P_u cu o valoare ce reprezintă utilizarea presupusă a procesorului. Cu cât valoarea este mai bine aleasă cu atât vor fi mai puține iterații pentru calcul.
2. Se calculează probabilitățile de cerere la fiecare nivel al rețelei şi la fiecare modul de memorie. Un pas intermediar ar putea fi calculul valorii statice pentru p_i ce reprezintă probabilitatea ca un nivel să fie traversat.
3. Se calculează timpii medii de răspuns ai comutatorului şi timpul mediu de răspuns al memoriei, adică r_i respectiv r_m.
4. Se calculează întârzierea rețelei şi a memoriei considerând valorile anterior calculate şi utilizând ecuațiile pentru determinarea lui α, β, γ şi η.
5. Se calculează un nou P_u, considerând valorile anterioare.
6. Se repetă paşii 2 - 5 până când noul P_u este în limita unei toleranțe admise comparabil cu ultimul P_u. Uzual se lucrează cu o valoare inițială de 0.5 pentru P_u şi cu o precizie de 0.00001.

6.2.3 Rezultate numerice şi discuţii.

În această secţiune se prezintă performanţa relativă a RMM,RIN-MN(B) şi RIN-MN convenţională. RIN-MN(B) este o reţea RIN-MN *bidirecţională*, echivalentă cu RMM,dar constituită din comutatoare *crossbar*. Pentru simetria abrevierii vom utiliza acronimul RMB. Această reţea permite cele patru tipuri de rutări propuse pentru RMM. RIN-MN convenţională – abreviat RMC – este similară cu reţeaua utilizată în sistemul Butterfly [BINK 97] şi se caracterizează prin faptul că pachetele reprezentând cereri şi răspunsuri traversează reţeaua într-o singură direcţie, de la procesoare către memorii.

Se analizează *utilizarea procesorului* şi *timpul de răspuns* pentru cele trei tipuri de reţele luând în considerare valori diferite ale lui m.

S-au efectuat simulări ciclu cu ciclu pentru a verifica că rutările propuse funcţionează şi s-au măsurat probabilităţile şi întârzierile reţelei [Iyer96].

Simularea a fost făcută utilizând un mediu descentralizat distribuit cu comutare de pachete în mod sincron. Comutatoarele au tampoane de ieşire cu lungime infinită. Toate evenimentele apar în sistem la începutul ciclului maşină. Fiecare ciclu, utilizat în simulare, reprezintă timpul cerut pentru transmisia unui pachet de la un tampon de ieşire al unui comutator sau al unui procesor (memorie) spre tamponul de ieşire al destinaţiei sale. Astfel se include transmisia pachetului prin canal şi timpul necesar comutatorului să-l ruteze spre tamponul de destinaţie corespunzător. Timpul minim necesar pentru ca un pachet să ajungă la memoria de destinaţie se bazează pe numărul de comutatoare dispuse în calea sa. Întârzierea introdusă de memorie, timpul de deservire a cererii, a fost fixată la patru cicluri.

Rutările descrise în §6.2.1 au fost simultate. Simularea compară fiecare sursă şi destinaţie, rulând algoritmul de rutare optimă şi apoi selectând rutarea. Deciziile referitoare la tipul de rutare se bazează exclusiv pe etichetele generate de către algoritmul de rutare optimă.

Mesajele de la procesor la memorie sunt generate utilizând probabilităţile p, m, m_1 prezentate în §6.2.2.

S-au comparat performanțele RMM şi RMB obținute pe cale analitică, respectiv prin simulare, în termeni de utilizare a procesorului şi respectiv timpul de răspuns.

În fig. 6.14 se prezintă o comparație între rezultatele analizei şi cele ale simulării pentru utilizările procesorului în cazul RMM, pentru două valori ale lui m.

În fig. 6.15 este prezentată comparația pentru timpul de răspuns.

În ambele figuri se observă că rezultatele analitice se aproprie foarte mult de cele obținute din simulare. Aceasta indică că independența cozilor de aşteptare, presupusă în timpul analizei, nu a fost una grosieră şi pentru valori mai mari ale lui m sunt satisfăcute mai multe cereri de acces fără a se parcurge RMM. Astfel, pentru m=0.9, P_u este mult mai mare în fig. 6.14 iar timpul de răspuns mult mai mic în fig. 6.15. Cu cât p creşte, sunt generate mai multe cereri şi creşte timpul de răspuns iar P_u se reduce.

În fig. 6.16 se prezintă utilizarea procesorului la un sistem 1024×1024 pentru RMM, RMB şi RMC ca şi funcția probabilității cererii de acces la memorie, p, cu m_1 =(1-m)(k-1)/(N-1). Se observă că timpul de răspuns este acelaşi pentru toate rețelele atunci când m=0.9 Pentru m=0.1, RMB se comportă mai bine ca şi RMM, care la rândul lui se comportă mai bine ca şi RMC. Pentru valori mai mari ale probabilității de cerere de acces la memoria locala, m, sunt satisfăcute mai multe cereri fără a se traversa RIN, cu cât p este mai mare, mai multe cereri sunt generate. Concluzia este că apare o diferență semnificativă în performanță între RMM şi RMC pentru valori coborîte ale lui m şi valori mari ale lui p. Diferența este de aproximativ 3% pentru valori superioare ale lui m şi aproximativ 10% pentru valori inferioare ale lui m.

În fig. 6.17 se prezintă timpii de răspuns a celor trei rețele pentru valori diferite ale lui m ca şi o funcție de p cu m_1=(1-m)(k-1)/(N-1). *Timpul de răspuns* este definit ca şi diferența dintre momentul când un procesor generează o cerere de acces şi momentul când el primeşte răspunsul din partea memoriei. Timpul de răspuns pentru toate RIN este mai mare când m are valori inferioare. Acest lucru se datorează traficului intens prin RIN. Din nou se observă că diferența maximă este de aproximativ 3% pentru valori ridicate ale lui m şi de aproximativ 10% pentru valori inferioare ale lui m.

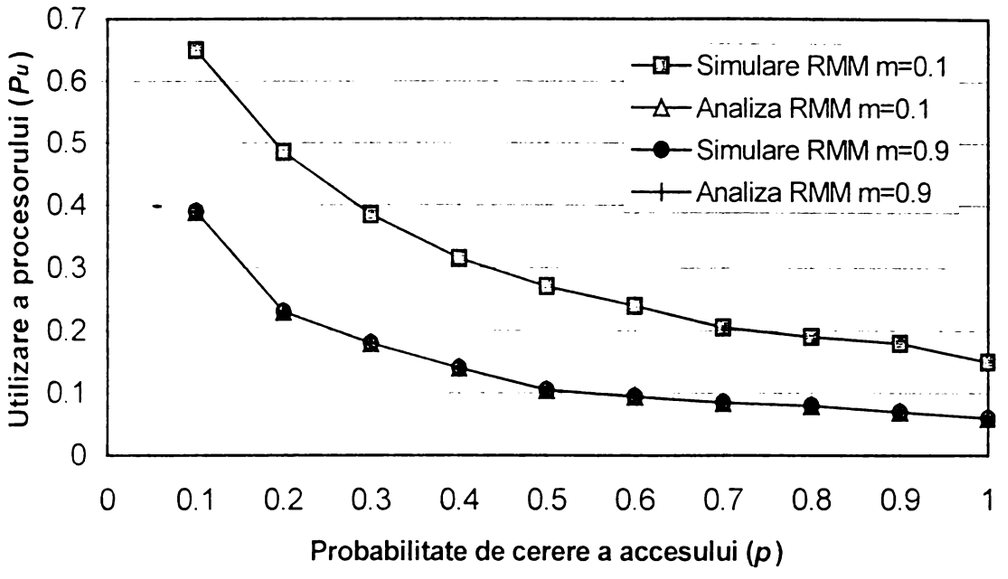


Fig. 6.14 Comparație a rezultatelor analizei și simulării pentru utilizările procesorului la RMM, variind m

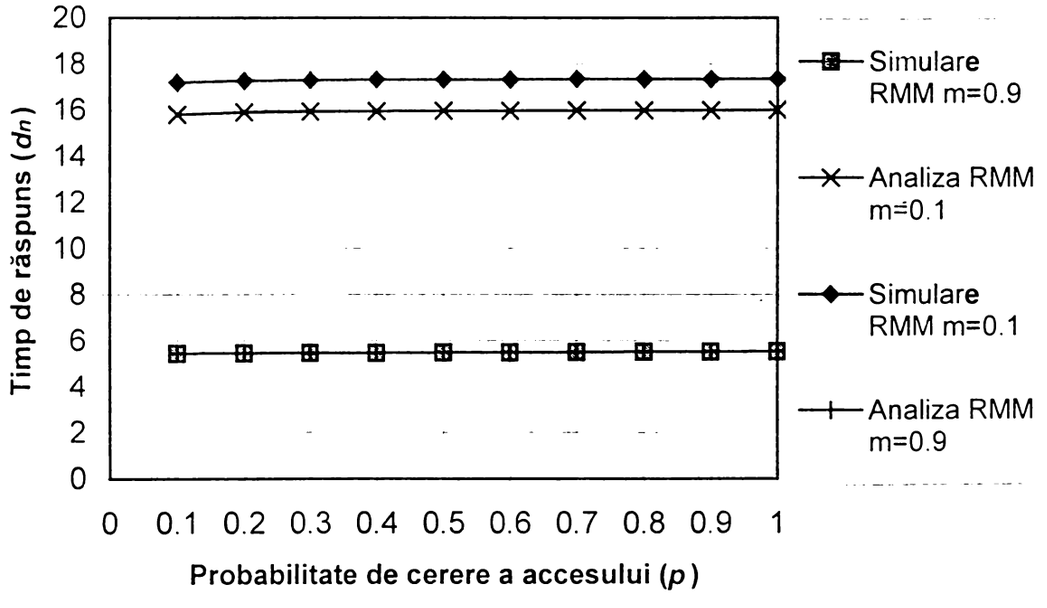


Fig. 6.15 Comparație a rezultatelor analizei și simulării pentru timpul de răspuns al RMM, variind m

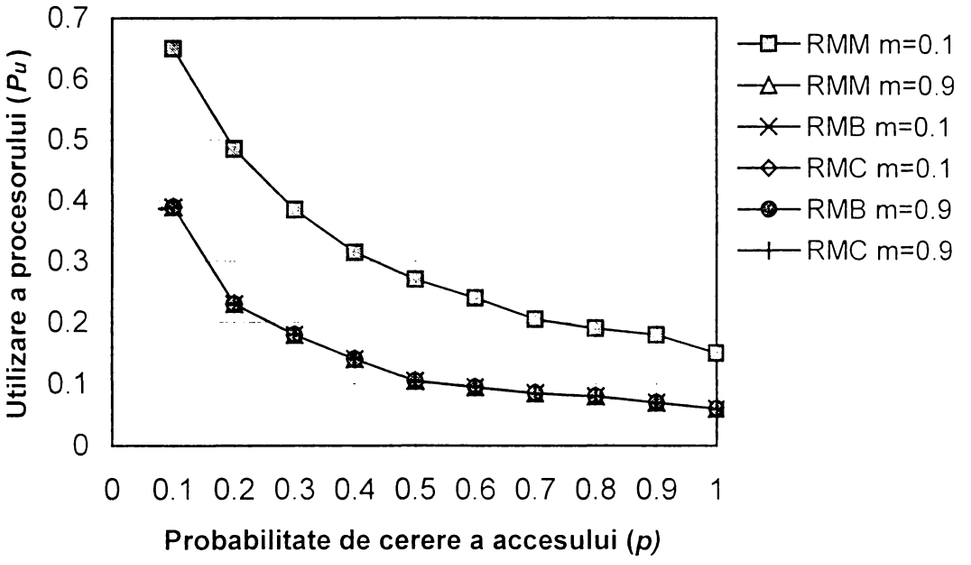


Fig. 6.16 Comparația între utilizările procesorului la diferite RIN, variind m

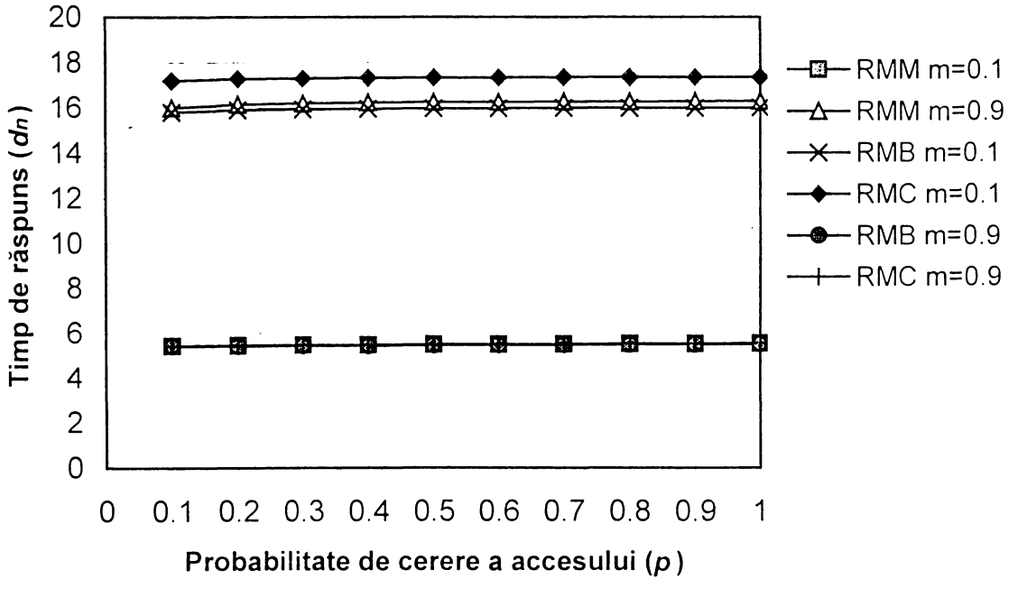


Fig. 6.17 Comparația între timpii de răspuns la diferite RIN, variind m

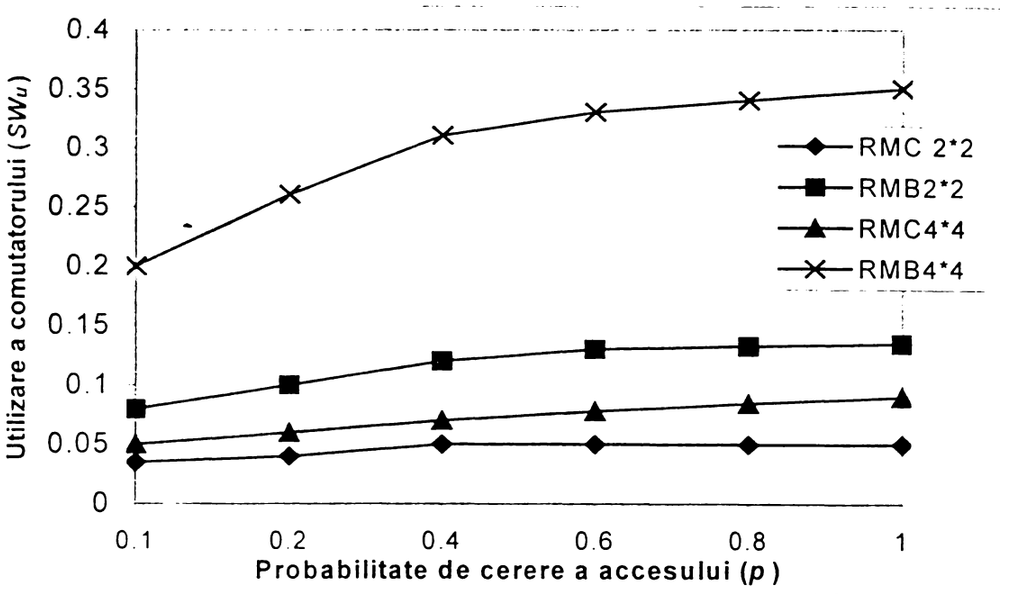


Fig. 6.18 Utilizarea comutatorului în RMM şi RMC variind dimensiunea comutatorului

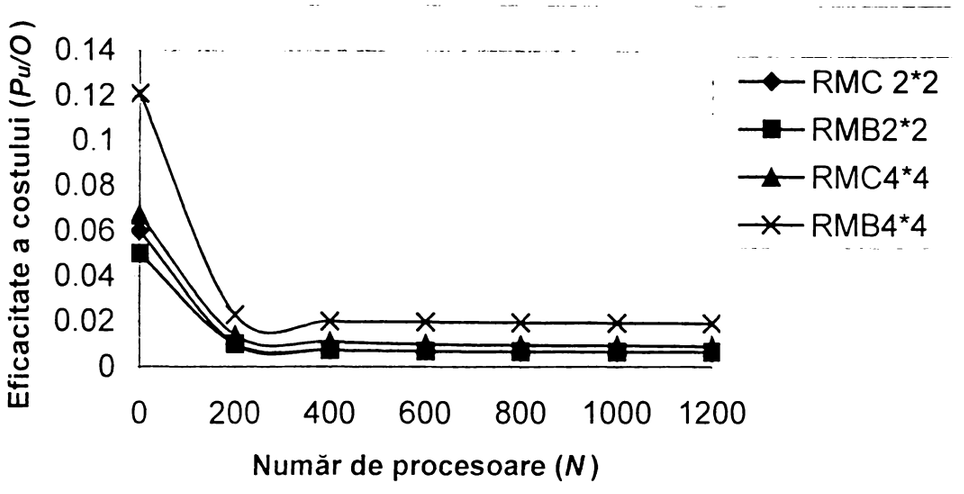


Fig. 6.19 Eficacitatea costului în RMM şi RMC variind dimensiunea comutatorului

În fig.6.18 se compară *utilizarea magistralei* într-un comutator RMM şi *utilizarea legăturii* într-un comutator *crossbar* din RMC, pentru 2 dimensiuni diferite ale comutatoarelor într-un SMP 1024×1024. Utilizarea comutatorului RMC este mai mică de 10%. Utilizarea unui comutator RMM 2×2 este mai puțin de 15% iar a unui comutator RMM 4×4 mai puțin de 35%. Pentru valori inferioare ale lui p utilizarea comutatorului este redusă şi eă creşte cu p pentru toate cazurile. Se confirmă teoria conform căreia conexiunile din RMC sunt neutilizate în mod masiv în comparație cu utilizarea comutatoarelor RMM.

Eficacitatea costului sistemelor bazate pe RMM şi RMC este prezentată în fig. 6.19 ca şi o funcție de talia sistemului. Deoarece *costul* unei RMM comparativ cu a unei RMC este identic atunci când se utilizează comutatoare 2×2, eficacitatea costului este aceiaşi pentru cele două sisteme. O rețea RMM cu comutatoare 4×4 este mai eficientă din punct de vedere a costului decât o rețea RMC echivalentă. Diferența este de aproximativ 100% pentru p=0.5. În fig.6.19 se observă clar superioritatea unei RMM în raport cu RMC.

În fig.6.20 şi 6.21 se prezintă *utilizarea procesorului* şi *timpii de răspuns* pentru diferite talii ale sistemelor. Rezultatele au fost obținute cu o probabilitate de cerere de acces a memoriei locale, m, fixată la 0.5 şi pentru două valori diferite ale lui p, adică 0.1 respectiv 0.5. Se poate constata că performanța rețelei RMM rămâne apropiată de cea a RMB chiar şi în situația când talia sistemului creşte. Se observă că în toate situațiile RMM se comportă mai bine decât RMC. Rețeaua RMM este scalabilă la un nivel ridicat în condițiile unui trafic impus.

În TABEL XIII se prezintă dimensiunile optime ale unui comutator pentru o valoare dată a lui p şi pentru diferite talii de sistem (m=1/N şi m_1=(k-1)/N). Pentru p=0.5 toate sistemele sunt optime pentru comutatoare 4×4. În acest caz dimensiuni superioare ale comutatorului vor conduce la un trafic crescut şi prin urmare mai puțină performanță şi mai puțină eficacitate a costului. Comutatoarele 2×2 vor avea un număr sporit de etaje şi prin urmare o întârziere mai mare şi vor fi mai puțin eficace din punct de vedere a costului. O excepție este sistemul de talie 512×512. Oricum, pentru p=0.1 situația se schimbă. Talia comutatorului 8×8 este optimă oriunde acesta este utilizat. Pentru dimensiuni ale comutatorului egale sau mai mari de 16×16 magistralele sunt saturate şi prin urmare performanța este redusă iar rețeaua are o eficacitate a costului diminuată.

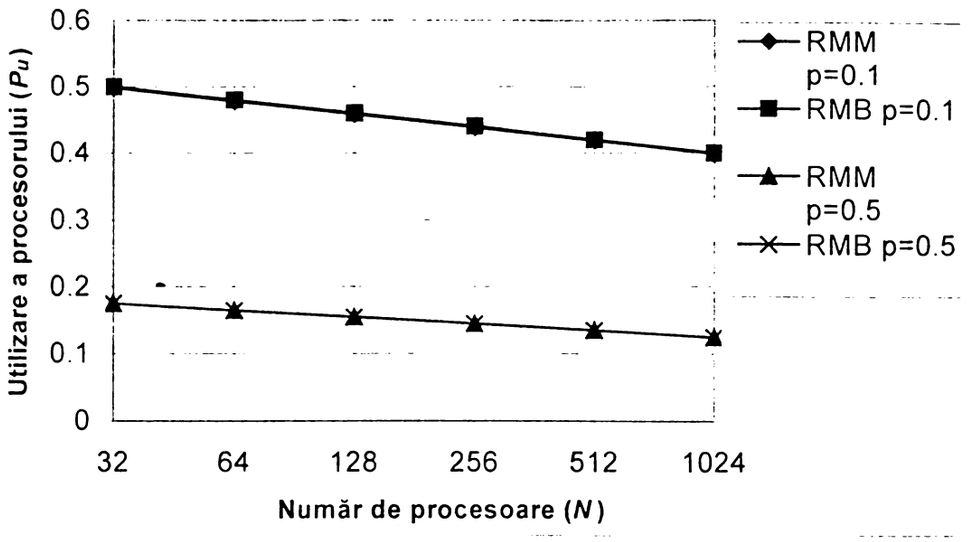


Fig. 6.20 Utilizarea procesorului: Scalabilitatea RMM în raport cu RMB,variind p

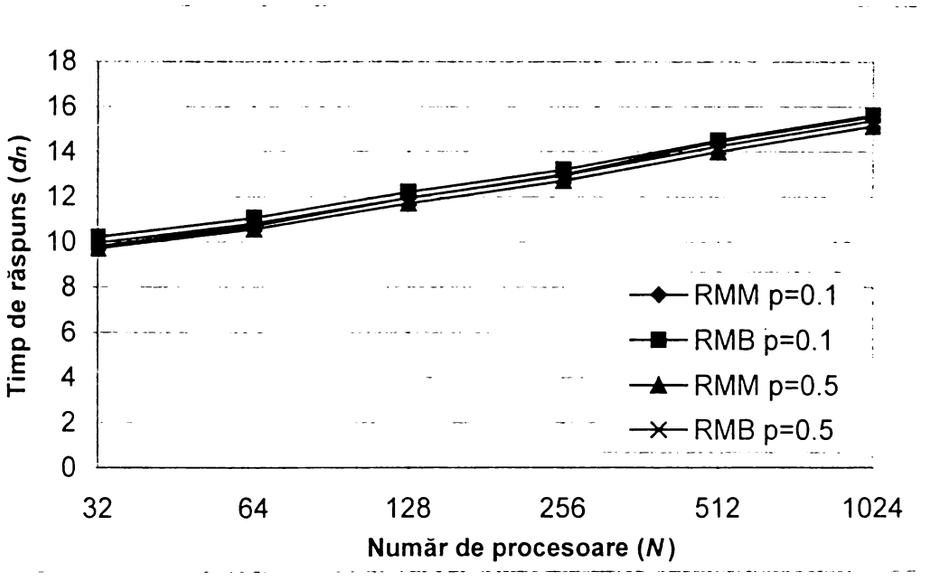


Fig. 6.21 Timp de răspuns : Scalabilitatea RMM în raport cu RMB, variind p

În TABEL XIV se prezintă utilizarea procesorului şi timpul de răspuns ale RMM pentru diferite dimensiuni ale comutatorului şi număr diferit de procesoare. În tabel

câteva locuri sunt necompletate deoarece o RMM $N\times N$ nu poate fi construită utilizând doar comutatoare $k\times k$. Atât p cât şi m sunt fixate la 0.5. Pentru N=64, se observă o scădere a lui P_u când k creşte de la 4 la 8. Aceasta datorită faptului că RMM este mai puţin eficientă în condiţiile apariţiei unui conflict mărit şi a unei întârzieri augmentate în cazul unui comutator RMM 8×8. Pe de altă parte se constată o ameliorare a performanţei într-un sistem 512×512, unde k creşte de la 2 la 8. Numărul de comutatoare din întreaga reţea este încă mare, menţinînd redusă rata conflictelor ceea ce este suficient pentru câştigul în performanţă.

Comparând performanţa RMM cu cea a RMB se poate constata că pe măsură ce dimensiunea comutatorului creşte, RMB oferă o utilizare mai mare a procesorului şi un timp de răspuns mai mic. Acest câştig în performanţă se datorează unui număr mai mic de conflicte în comutatorul *crossbar*. Oricum RMB oferă o performanţă crescută în detrimentul costului. Luând în considerare numărul de conexiuni într-un comutator $k\times k$ care este k^2, că pentru un comutator bazat pe magistrală el este $2k$, iar costurile totale ale RMB şi RMM sunt $kN\log_k N$ şi $2N\log_k N$ iar apoi coroborând aceste date cu *utilizarea procesorului* şi cu *timpul de răspuns*, rezultă că *eficacitatea costului* unei RMM este mai mare decât a unei RMB. Se constată de asemenea că un comutator 4×4 este cel mai eficient din punct de vedere al costului pentru diferite dimensiuni de reţea.

TABEL XIII

Dimensiunea optimă a unui comutator pentru eficacitatea costului.

N	**Dimensiuni posibile ale comutatorului**	**Dimensiune optimă**	
		p=0.5	p=0.1
16	2×2, 4×4, 16×16	4×4	4×4
64	2×2, 4×4, 8×8, 64×64	4×4	8×8
256	2×2, 4×4, 16×16, 256×256	4×4	4×4
512	2×2, 8×8, 512×512	2×2	8×8
1024	2×2, 4×4,32×32, 1024×1024	4×4	4×4

TABEL XIV

Utilizarea procesorului şi timpul de răspuns al RMM şi RMB variind k şi n

Nr.proc	Diferite talii ale comutatorului											
N	k=2				k=4				k=8			
	P_u		T_r		P_u		T_r		P_u		T_r	
	RMM	RMB	RMM	RMB	RMM	RMB	RMM	RMB	RMM	RMB	RMM	RMB
64	0.150	0.153	10.96	10.73	0.181	0.191	9.06	8.17	0.179	0.210	9.19	7.51
256	0.130	0.132	13.17	12.88	0.166	0.176	9.95	9.30	-	-	-	-
512	0.121	0.123	14.40	14.07	-	-	-	-	0.158	0.190	10.69	8.54
1024	0.114	0.116	15.37	15.03	0.151	0.161	11.20	10.41	-	-	-	-

6.2.4 Toleranța la defect şi fiabilitatea RMM

6.2.4.1 Toleranța la defect

În cadrul analizei fiabilității la RMM, modelul de defect de bază este *defectul de comutator*. La apariția unei defecțiuni la nivel de comutator, acesta devine total inutilizabil şi, pe cale de consecință, toate legăturile conectate de către acest comutator devin inutilizabile [BNA94], [BNT94].

În cazul RIN-MN convenționale există doar o singură cale între fiecare pereche sursă-destinație. Prin urmare când un comutator se defectează în oricare nivel al rețelei, rețeaua devine inutilizabilă. În cazul RMM există căi multiple între fiecare pereche sursă-destinație ceea ce conferă rețelei capacitatea de a tolera toate defecțiunile de tip *comutator singular* şi, în cazuri bine determinate, defecțiunile de tip *comutatoare multiple*.

În capitolul 5, se arată că toleranța la defect în cazul defectării unui singur comutator, implică existența a cel puțin două căi distincte între fiecare pereche sursă-destinație. Aceasta înseamnă că toate comutatoarele dintr-o cale trebuie să fie distincte de comutatoarele din cealaltă.

În cazul RMM există trei cazuri ce trebuie să fie luate în considerare depinzând de etichetele sursă şi destinație. $S=s_0s_1...s_{l-1}$ şi $D=d_0d_1...d_{l-1}$

Cazul 1: $s_i=d_i$ pentru toți i ,adică $S=D$.

În acest caz cererea de acces parcurge doar magistrala internă şi nu traversează nici un comutator. Prin urmare nu sunt relevante defectele comutatoarelor.

Cazul 2: $s_i=d_i$ pentru toți i, cu excepția $i=l-1$, adică $S \neq D$ doar pentru bitul din extrema dreaptă.

În acest caz atât sursa cât şi destinaţia sunt conectate la acelaşi comutator în nivelul 0 şi de asemenea la acelaşi comutator în nivelul l–1. Cererea poate utiliza oricare dintre aceste comutatoare pentru a efectua o buclare şi pentru a ajunge la destinaţie. Prin urmare, când unul dintre aceste comutatoare se defectează, RMM poate încă funcţiona normal.

Cazul 3: $s_i \neq d_i$, adică S şi D diferă cel puţin într-un bit i , dar $i \neq l$–1.

În acest caz cererea traversează reţeaua fie în sens Direct, fie în sens Invers, utilizând rutările RD sau RI.

În cazul rutării RD, o cerere soseşte în poziţia $s_j s_{j+1}...s_{l-2}d_0d_1...d_{j-1}s_{j-1}$ la stânga nivelului j. Această poziţie corespunde comutatorului SW1= $s_j s_{j+1}...s_{l-2}d_0d_1...d_{j-1}$ în nivelul i.

În cazul rutării RI, o cerere soseşte în poziţia $d_j d_{j+1}...d_{l-2}s_0s_1...s_{j-1}s_j$ în dreapta nivelului j. Această poziţie corespunde comutatorului SW2=$d_j d_{j+1}...d_{l-2}s_0s_1...s_{j-1}$ în nivelul j .

Deoarece există cel puţin un i, altul decât i=l-1, astfel ca $s_i \neq d_i$, comutatoarele SW1 şi SW2 diferă unul de celălalt în fiecare nivel j. Prin urmare calea în sens Direct şi calea în sens Invers sunt *disjuncte* în termeni de comutatoare.

Prin urmare, RMM este tolerant la defectele de tip *un comutator defect*.

6.2.4.2 Fiabilitatea

Anduranţa RMM la defecţiuni multiple de comutator poate fi exprimată în termeni de *fiabilitate de teminal medie* (*Average Terminal Reliability*) –*FTM*

Fiabilitatea de terminal (*FT*) a unei perechi sursă – destinaţie date, este definită ca fiind probabilitatea ca să existe cel puţin o cale liberă de defect între sursă şi destinaţie, cunoscându-se că fiecare comutator are o anumită fiabilitate.

FTM a reţelei este media *FT*-urilor a tuturor perechilor sursă-destinaţie. Pentru o RIN-MN cu l niveluri, există doar o cale de lungime l între fiecare pereche sursă-destinaţie cu excepţia situaţiei când sursa şi destinaţia sunt identice. Prin urmare *FTM* pentru o RIN-MN este:

$$FTM=R^l$$

RMM este bidirecţională şi există mai multe căi între o pereche sursă-destinaţie. Câteva comutatoare sunt partajate de către două sau mai multe căi. *Graful de redundanţă* pentru RMM ce descrie toate căile disponibile între o pereche sursă-destinaţie, este prezentat în fig.6.22 El constă din două noduri speciale, nodul sursă S şi nodul destinaţie D, şi un set de noduri reprezentând comutatoarele ce se găsesc pe diferitele trasee între S şi D.

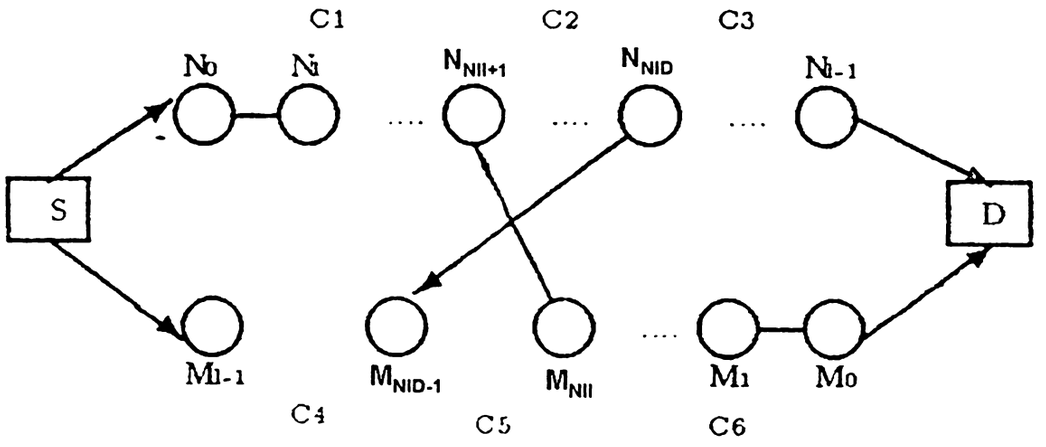


Fig.6.22 Graf de redundanță în RMM

$$C_1=N_0...N_{NII}$$
$$C_2=N_{NII+1}...N_{NID}$$
$$C_3=N_{NID-1}...N_{I-1}$$
$$C_4=M_{I-1}...M_{NID}$$
$$C_5=M_{NID-1}...M_{NII}$$
$$C_6=M_{NII-1}...M_0$$

Calea parcursă în cazul RD constă din componente $C_1 \cdot C_2 \cdot C_3$ şi calea în cazul RI din componente $C_4 \cdot C_5 \cdot C_6$. În mod similar căile parcurse în cazul BD şi BI constau din componente $C_1 \cdot C_2 \cdot C_5 \cdot C_6$ şi, respectiv , din $C_4 \cdot C_5 \cdot C_2 \cdot C_3$. Fiabilitatea acestor căi va depinde de fiabilitatea componentelor individuale ce sunt funcții de NID şi NII.

Fiabilitatea de terminal medie (*FTM*) a RMM, respectiv a RMC, este prezentată în fig.6.23. S-au utilizat comutatoare 2×2 şi diferite valori ale *fiabilității comutatorului* – *FSW*. În fig.6.24 sunt prezentate diagramele pentru comutatoare 4×4.

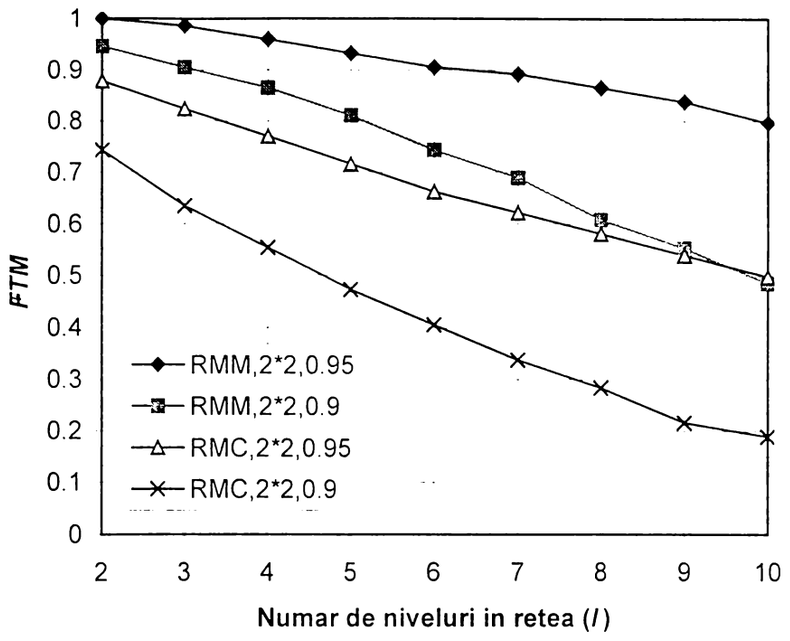


Fig. 6.23 Fiabilitatea rețelelor în cazul comutatoarelor 2×2

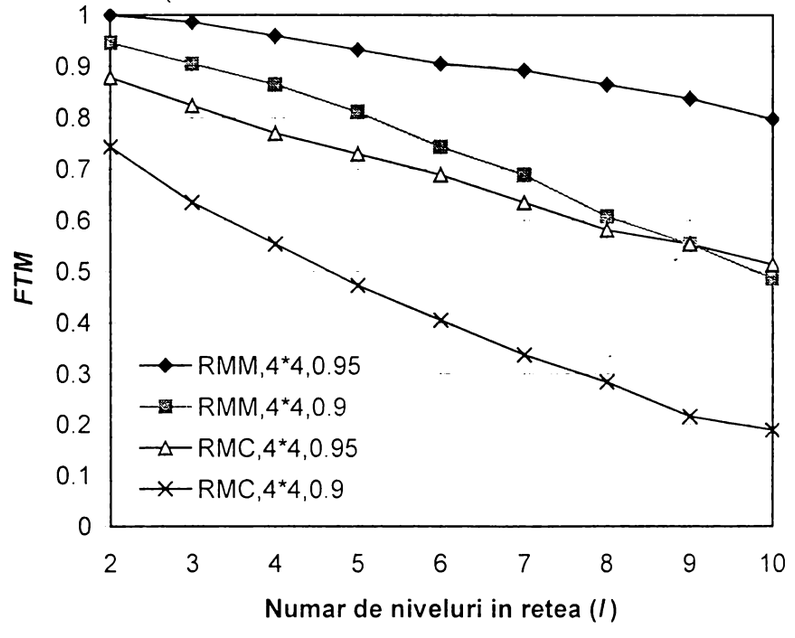


Fig. 6.24 Fiabilitatea rețelelor în cazul comutatoarelor 4 × 4

Fiabilitatea ambelor rețele scade ca şi o funcție de *l*, deoarece numărul de comutatoare în fiecare cale creşte cu *l*. Se observă că RMM are o mai bună fiabilitate decât RMC în întreg spectrul de valori ale parametrilor. Chiar şi în situația în care *FSW*=0,9 pentru RMM, această rețea este mai fiabilă decât RMC cu *FSW*=0,95. Fiabilitatea mai ridicată a RMM se datorează în mod evident căilor multiple dintre perechile de noduri sursă-destinație, în contrast cu căile unice într-o RMC.

6.3 PROTOCOALE DE MENȚINERE A COERENȚEI

Un modul de memorie este conectat la toate procesoarele într-o structură de *arbore* ca şi în fig. 6.25. Modulul formează *apex*-ul, comutatoarele sunt *noduri* intermediare şi procesoarele sunt *frunze*.

Graful arborelui este identic cu talia I/E a unui comutator. Un bloc este localizat într-un modul de memorie, fie intern, fie extern. Toate cererile de acces sau răspunsurile pentru acel bloc vor parcurge arborele având ca apex modulul de memorie. Prin urmare protocoalele de menținere a coerenței pot fi descrise în termeni de ierarhie de nivele ale unui arbore, ceea ce corespunde etajelor din RMM.

Procesoarele şi memoriile cache asociate sunt dispuse pe *nivelul* 0. Primul etaj de comutatoare din RMM este dispus pe *nivelul* 1, al doilea pe *nivelul* 2,etc.

Porturile superioare ale unui comutator de pe nivelul *i* sunt conectate cu porturile inferioare ale comutatoarelor de pe nivelul $i+1$.

Un comutator este un *predecesor* al unui procesor dacă o cerere de acces la memorie lansată de procesor traversează acel comutator.

Un procesor sau o cache este *în aval* față de un comutator, dacă comutatorul este un predecesor al acelui procesor. Dacă un procesor nu este *în aval față de* un comutator, atunci el este *în amonte* față de acel comutator. Comutatoarele de pe *nivelul i* care sunt conectate la un comutator de pe *nivelul* $i+1$ se numesc *descendenți direcți* ale comutatorului dispus pe *nivelul* $i+1$.

Descrierea mişcării unui bloc partajat se va face în continuare în termenii mai sus menționați.

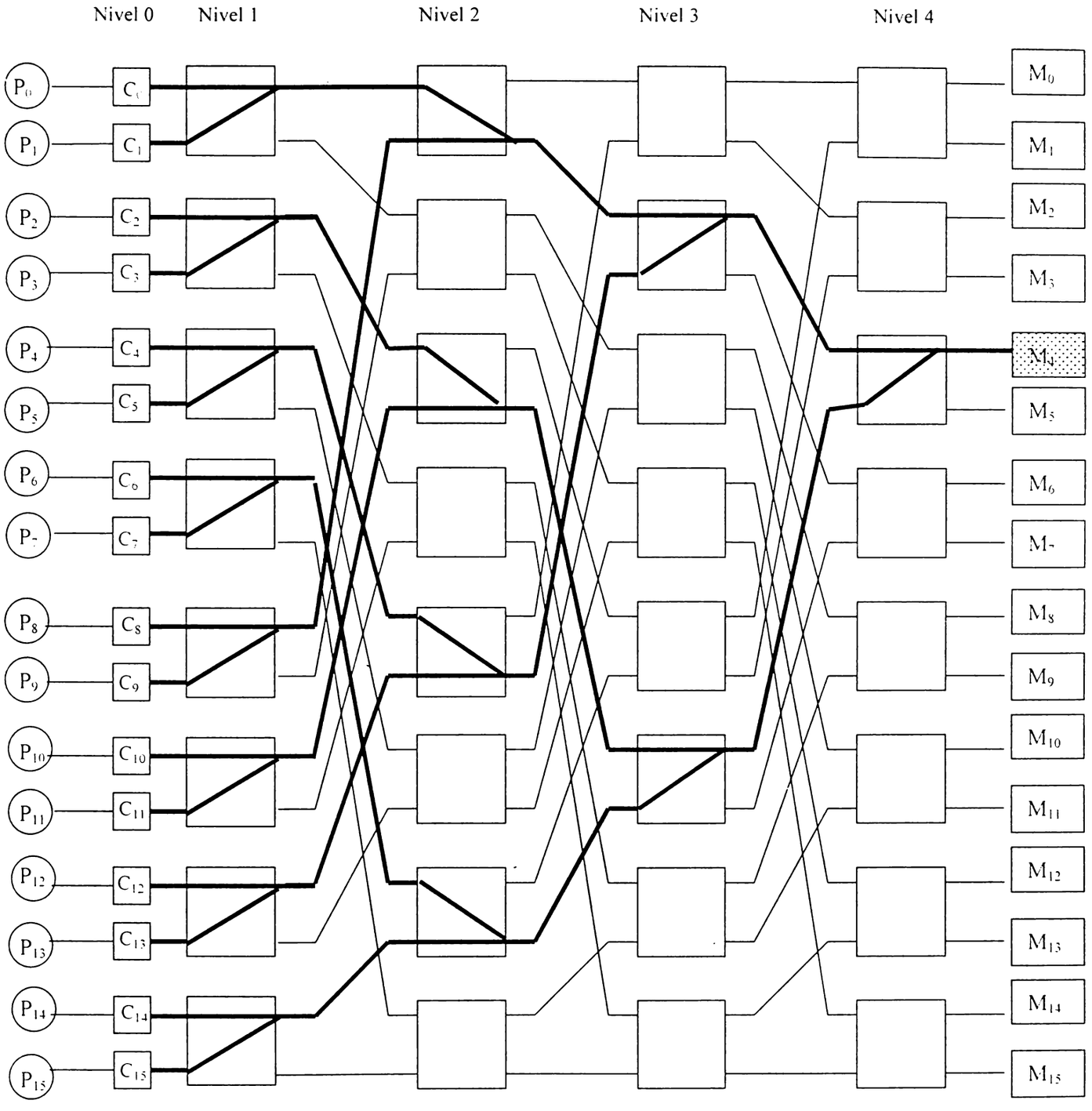
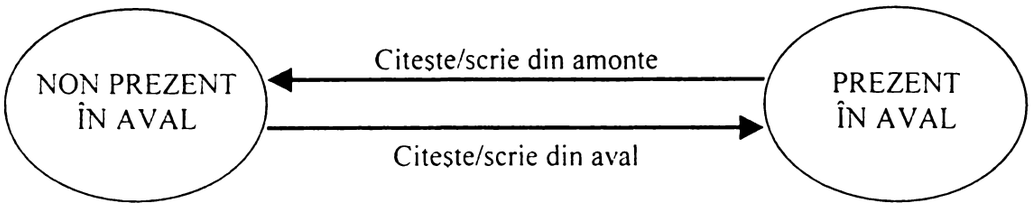
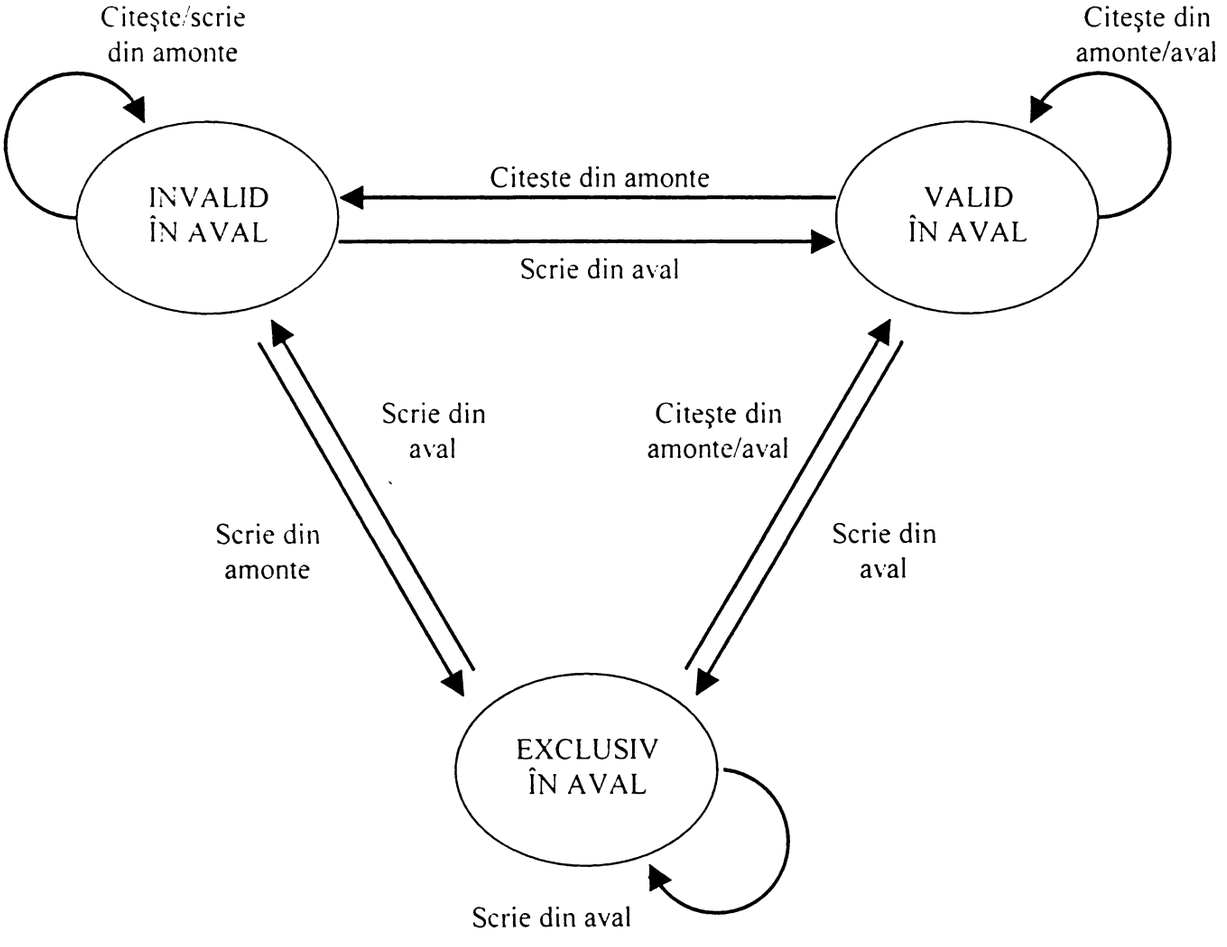


Fig. 6.25 Un *arbore* ce conectează M_4 cu toate procesoarele



a)



b)

Fig.6.26 Diagrame ale tranziţiilor de stare în protocol
a) Protocol pentru o singură copie
b) Protocol pentru copii multiple

A. Protocol pentru o singură copie.

Acest protocol permite prezența în sistem a unei singure copii a unui bloc partajat la un moment dat. Un bloc partajat poate fi în starea *prezent în aval* sau în starea *non-prezent în aval* în raport cu un comutator la un nivel dat. Această situație este prezentată în fig. 6.26(a). Starea *prezent în aval* înseamnă că un procesor este localizat sub comutatorul ce are o copie a blocului. Stare *non - prezent în aval* înseamnă că blocul este fie prezent în alt procesor, fie în memorie. Dacă este *prezent în aval* directorul din comutator păstrează informația referitoare la descendentul ce trebuie să fie căutat pentru a obține o copie a blocului partajat. Dacă *nu este prezent în aval* cererea traversează RMM spre memorie aşa cum se indică în eticheta de rutare. În final, procesorul ajunge la deținătorul curent al blocului prin intermediul directorilor din comutatoare, încarcă blocul şi devine noul deținător. Astfel fiecare referire de citire sau de scriere a blocului de la un alt procesor implică o mişcare a datelor de la deținătorul curent la acel procesor.

B. Protocol pentru copii multiple.

Acest protocol permite existența unor copii multiple a blocurilor partajate din sistem şi se bazează pe *invalidarea la scriere.*

Un bloc din memoria cache poate fi în una din cele trei stări : *invalid în aval, valid în aval* şi *exclusiv în aval* în raport cu o memorie cache locală sau cu comutatoare de pe un nivel intermediar.

Semnificația stărilor în raport cu comutatorul de pe nivelul *i* sau cu memoriile cache locale este după cum urmează.

- *Invalid în aval* înseamnă că blocul nu este prezent în oricare dintre memoriile cache care sunt în aval față de comutator.
- *Valid în aval* înseamnă că blocul este prezent în starea *valid* în una sau mai multe cache *în aval* de comutator.
- *Exclusiv în aval* înseamnă că blocul este în starea *exclusiv* într-o cache *în aval* de comutator.

Diagrama de tranziții de stare pentru un bloc partajat în raport cu comutatorul de pe un nivel *i* sau cu o memorie cache locală este prezentată în fig. 6.26 (b) Acțiunile efectuate

de controlerele de coerență la diferite nivele pot fi descrise în termeni de *eşec la citire*, *succes la scriere* şi *eşec la scriere* într-un bloc partajat.

În cazul unui *succes la citire*, cache-ul local furnizează blocul şi nu mai este necesară nici o acțiune. Toate blocurile sunt marcate *invalid* în cache-uri şi comutatoarele de pe toate nivelurile sunt aduse în stare inițială.

În cazul unui *eşec la citire* în cazul unei cereri de acces la un bloc partajat din cache-ul local, cererea este transferată comutatorului de pe nivelul 1 care este antecesorul acelui cache în raport cu blocul ratat. Dacă comutatorul de pe nivelul 1 arată că blocul este în starea *valid în aval* sau în *exclusiv în aval*, se extrage o copie din cache-ul plasat pe nivelul 0 ce deține această copie şi ea este furnizată solicitantului. Dacă blocul este în starea *exclusiv în aval*, memoria este actualizată în raport cu politica de rescriere de către cache-ul ce furnizează răspunsul.

Dacă blocul este în starea *invalid în aval*, cererea traversează rețeaua spre comutatoarele de pe nivel superior până când starea blocului este identificată ca fiind fie în *valid în aval* sau în *exclusiv în aval* la un anumit nivel. Dacă blocul este în starea *valid* într-un comutator de pe un nivel superior dincolo de nivelul 1, blocul este extras din memoria principală prin comutatoarele de pe nivel superior. În cazul *exclusiv*, blocul este intotdeauna furnizat de cache-ul ce deține copia exclusivă. Dacă blocul este este *invalid în aval* la toate nivelurile, el este extras din memoria principală. Comutatoarele de pe traseul de răspuns schimbă starea blocului în *valid în aval*, în directoarele lor locale.

Dacă există un *succes la scriere* la un bloc aflat în starea *exclusiv în aval* în cache-ul local, blocul este scris imediat şi nu este necesară nici o acțiune de menținere a coerenței.

În cazul unui *succes la scriere* asupra unui bloc valid, se transmite un semnal de invalidare spre comutatoarele plasate pe un nivel superior. Semnalul de invalidare traversează rețeaua în sens direct spre cel mai de sus nivel pentru a schimba starea blocului în *exclusiv în aval* în comutatoarele dispuse de-a lungul căii. În acelaşi timp, comutatorul de pe nivelul i transmite mai departe, în aval, spre comutatorul dispus pe nivelul i-1, un semnal de invalidare, doar dacă blocul este în starea *valid în aval* în acel comutator.

În final, toate cache-urile de pe nivelul 0 ce au o copie validă a blocului invalidează copiile lor şi transmit un semnal de validare a invalidării înapoi spre comutatoarele dispuse pe nivelul 1, urmând aceeaşi cale.

Comutatoarele de pe nivelurile intermediare colectează aceste semnale şi după ce toate semnalele sunt recepţionate, transmit un semnal de validare a acestora.

Toate comutatoarele de pe calea de invalidare schimbă starea blocului în *invalid în aval*.

În final solicitantul recepţionează un semnal de validare, schimbă starea blocului în *exclusiv în aval* şi finalizează operaţiunea. Comutatoarele predecesoare ale solicitantului schimbă starea în *exclusiv în aval*.

În cazul unui *eşec la scriere* în cache-ul local, cererea traversează în sus reţeaua până când ajunge la un comutator ce conţine blocul fie în starea *valid în aval* fie în *exclusiv în aval*. Dacă blocul este în starea *valid în aval* în cadrul unui comutator, se transmite un semnal de invalidare spre comutatoarele dispuse pe nivelurile superioare şi inferioare.

Semnalul se propagă ca şi în cazul *succesului la scriere*. Cache-urile având o copie validă schimbă starea blocului în *invalid*. Blocul este extras din memoria principală prin comutatoarele de pe nivelul superior şi este încărcat în stare *exclusiv în aval*. Dacă blocul este în starea *exclusiv în aval* în comutator atunci cache-ul având copia, transmite blocul spre solicitant. Cache-ul furnizor şi predecesorii săi schimbă starea blocului în *invalid în aval*. Solicitantul şi toţi predecesorii săi schimbă starea blocului în *exclusiv în aval*.

C. Organizarea tabelului director.

În SMP există două tipuri de referiri la memorie :

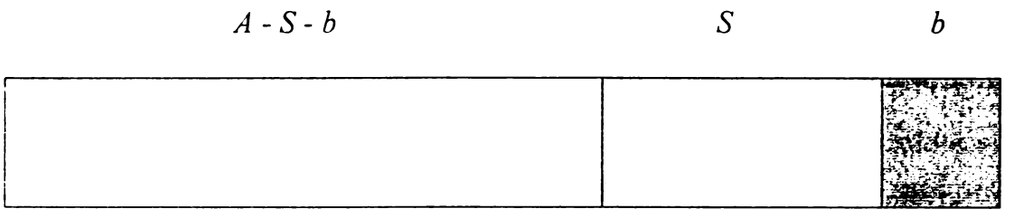
- *private* - instrucţii, date private, date partajate de tip citeşte-numai
- *partajate* - date din locaţii partajate de tip R/W

Se presupune că în cursul compilării, compilatorul distinge între datele private şi cele partajate astfel că ele sunt menţinute în modulele de memorie diferite [MN99], [MNLS97].

Presupunem că numărul de blocuri partajate, N_{sb}, în cadrul sistemului este fix şi că blocurile partajate sunt distribuite în modulele de memorie. Compilatorul detectează aceste structuri de date şi le plasează în locaţii corespunzătoare.

Fie A numărul total de biți din adresa fizică a unei locații de memorie. Sunt necesari b biți pentru a adresa un cuvânt în interiorul unui bloc de memorie. Prin urmare A-b biți sunt utilizați pentru a adresa un bloc de memorie, privat sau partajat. Deoarece N_{sb} reprezintă numărul de blocuri partajate, sunt necesari $S = \log_2 N_{sb}$ biți pentru a distinge printre blocurile partajate.

O configurație tipică de adresă de memorie este prezentată în fig. 6.25



A - număr de biți în adresa fizică

S - număr de biți pentru adresarea unui bloc partajat

b - număr de biți pentru adresarea unui cuvânt

Fig.6.27 Configurarea adresei fizice

Configurarea se face astfel :

Când c.m.s. biți (A-S-b) sunt în intervalul 000...00 la 111...10, adresa este a unui bloc *privat*. Când aceşti biți sunt toți 1, adresa este a unui bloc *partajat*.

Controlerul memoriei cache poate distinge un bloc partajat prin decodificarea doar a bitului (S+b+1). Controlerul validează o linie de comandă indicând că această adresă este a unui bloc partajat. Această linie face parte din liniile de adresă şi se transmite comutatoarelor din RMM. Cei S biți sunt utilizați pentru adresarea în mod unic a unui bloc partajat în director.

Directorul prezent într-un comutator RMM este organizat ca şi în fig. 6.28.

Talia directorului este fixă şi el poate conține informații referitoare la N_{sb} blocuri partajate. Câmpul *Stare* în cadrul unei intrări a directorului memorează starea blocului

partajat. Pentru protocolul prezentat în fig. 6.26(b) sunt necesari doar 2 biți pentru codificarea informațiilor de stare. Câmpul *Prezență* înmagazinează informația asupra blocului în raport cu cei *k* descendenți ai comutatorului (*k* este numărul de intrări în comutator).

	Stare	*Biți de prezență*			
		Ch1	Ch2	.. .	Ch *k*
Bloc 1					
Bloc 2					
. . .				. . .	
Bloc N_{sb}					

Fig. 6.28 Organizarea directorului într-un comutator RMM

Dacă câmpul *Stare* indică că blocul este *invalid în aval*, toți biții de prezență sunt pe 0.

Dacă câmpul *Stare* indică că blocul este *exclusiv în aval* doar unul şi numai unul din biții de prezență - corespunzător descendentului unde blocul este în această stare - se poziționează pe 1.

Dacă se indică că blocul este *valid in aval*, toți biții de prezență corespunzători descendenților unde blocul este valid sunt 1, ceilalți sunt 0.

Se poate observa că directorul păstrează informația de stare doar referitoare la descendenții săi, şi nu la toate procesoarele situate în aval. Aceasta permite ca lungimea cuvântului a unei intrări a directorului să fie redusă şi uniformă de-a lungul tuturor nivelurilor. Deoarece organizarea directorului dintr-un comutator RMM este similară cu cea a unei memorii cache şi pentru că talia directorului, în termeni de biți, este mult mai mică decât cea a unei cache, directorul încorporat în comutator este cel puțin la fel de rapid ca şi un controler al unei memorii cache.

6.4. SIMULAREA PILOTATĂ DE EXECUȚIE

În ultimă instanță, timpul de execuție al unei aplicații rulate pe o arhitectură multiprocesor reprezintă un indicator final al preformanței. În scopul verificării modelului analitic al RMM şi a comparării lui cu modelele analitice ale RMC şi ale RMB. au fost efectuate simulări ale mai multor aplicații. Simulatorul utilizat [BINK97], [BD91], [Iyer96], [Iyer99] se bazează pe simulatorul Proteus [BD91] dezvoltat la MIT. Acest simulator este un simulator pilotat de execuție ce simulează SMP-MP şi SMP-TM. El poate de asemenea simula câteva tipuri de RIN cât şi configurații de memorii.

6.4.1 Construcția simulării

Simularea pilotată de execuție constă din două părți de bază : *codul aplicației* şi *codul simulatorului* [Iyer96] [Kum96], [KBI96].

Aplicația este scrisă într-un superset C. Se definesc cuvinte-cheie ce sunt utilizate pentru declararea de variabile partajate şi operatori pentru accesarea variabilelor partajate. Simulatorul este apelat după un acces la o variabilă partajată sub forma unei citiri sau a unei scrieri. Simulatorul oferă o bibliotecă de rutine pentru transferul de mesaje, managementul memoriei, sincronizare, etc. Sarcina principală în construcția unei simulări ce modelează rețeaua luată în considerare se realizează prin adăugarea sa la codul simulatorului. În fig. 6.29 se prezintă etapele necesare în construcția unei simulări.

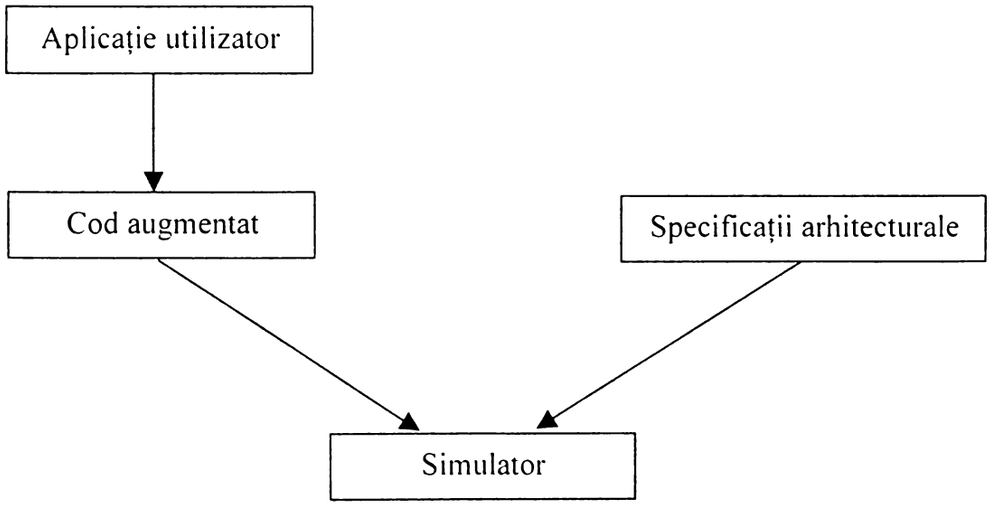


Fig. 6.29 Simulator pilotat prin execuție

6.4.2 Interfața cu rețeaua

Configurația nodului şi interfața cu rețeaua în cadrul simulatorului sunt prezentate în fig. 6.30.

Atât controlerul memoriei cache (CC), cât şi controlerul memoriei (CM), sunt conectate la interfață, aşa că ele pot comunica direct cu alte noduri. Se presupune că procesoarele au un singur flux şi nu se efectuează nici o aducere prealabilă a blocurilor de date cache. Sistemul este presupus secvențial consistent, adică nu există tampoane de scriere, şi eşecurile de acces cache la operații de tip LOAD/STORE blochează procesorul.

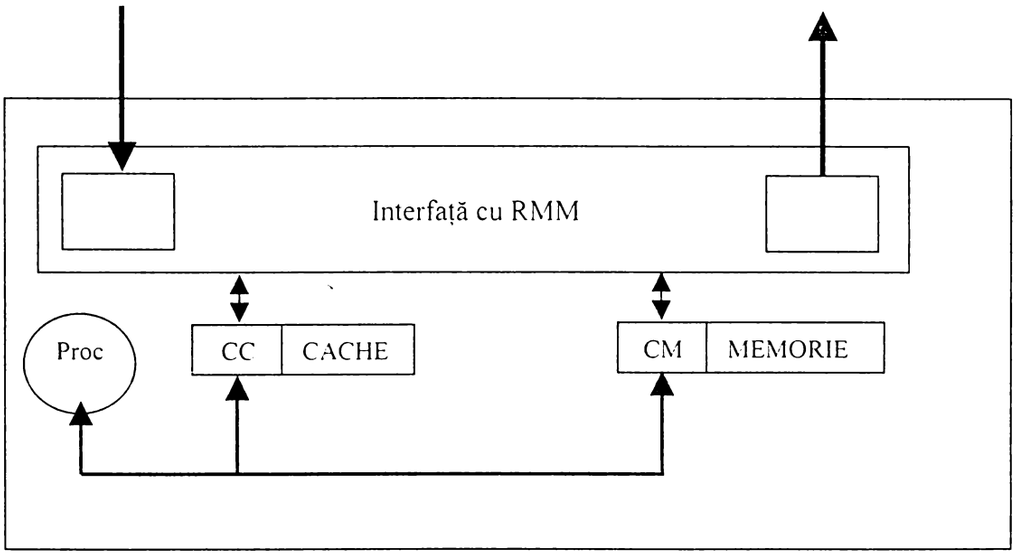


Fig. 6.30 Configurația unui nod şi interfața cu rețeaua

Interfața cu rețeaua (IR), furnizează spațiul tampon pentru toate mesajele ce vin sau ce pleacă în/din nod. Se presupune că IR are suficient spațiu pentru a înmagazina toate mesajele. Interfața oferă de asemenea servicii constând în divizarea mesajelor în pachete, inițializarea header-ului sau a oricărui pachet cu informații necesare pentru rutare, etc.

Comutatoarele conțin un număr limitat de tampoane la fiecare intrare destinate pentru recepționarea mesajelor. Spațiul tampon este organizat în termeni de pachete de dimensiune fixată. Mesajele sunt divizate în pachete de dimensiune fixată, iar fiecare pachet conține informația de rutare. Pachetul este transferat de la un comutator la celălalt

ca şi un întreg. Un pachet este avansat către comutatorul din etajul următor doar dacă are suficient spaţiu pentru a înmagazina întregul pachet. În caz de conflict la ieşire se apelează la disciplina de ordonarea *round-robin*.

6.4.3 Parametrii de simulare

Parametrii de sistem utilizaţi în simulare sunt prezentaţi în TABEL XV. Am simulat un SMP-MPD cu 16 noduri ce utilizează o reţea cu 4 niveluri, conţinând comutatoare 2 X 2. S-a presupus că timpul de întârziere datorat comutatorului este de 1 ciclu, suficient pentru a produce decizia de rutare în ceea ce priveştea header-ul pachetului sau pentru primul flit al mesajului în cazul rutării de tip wormhole. Dimensiunea flit-ului şi dimensiunea legăturii este de 16 bit. În acest mod se poate considera că transferul unui flit prin legătură durează un ciclu. În sistemele cache-coerente, mesajele de date conţinând blocuri de memorii sunt mai lungi, în timp ce mesajele de cerere de acces şi de menţinere a coerenţei conţinând doar informaţii referitoare la protocolul transmisiei şi adrese, sunt mai scurte.. S-au luat în considerare mesaje de lungime de 8 byte pentru menţinerea coerenţei şi de 40 bit pentru date. Aceste dimensiuni se regăsesc şi în experimentele citate în literatură [KB93], [BINK97]. În cazul comutării de pachete, pachetele au o lungime de 8 byte (1 – header, 7 – date). Un pachet consumă un timp de întârziere format din 1 + 4 cicli pentru comutare şi respectiv pentru transmisia spre nivelul următor din RMM.

TABEL XV

Parametrii pentru simulare

Parametru	Valoare
Număr de procesoare	16
Dimensiunea memoriei partajate în nod	32 Kbytes
Dimensiunea memoriei cache	8 Kbytes
Dimensiunea liniei memoriei cache	32 bytes
Dimensiunea setului	2
Timpul de acces la memoria cache	1
Timpul de acces la memorie	8
Întârzierea de comutare	1
Lăţimea conexiunii	16 biţi
Lungimea unităţii de transmis din pachet	16 biţi
Dimensiunea pachetului	8 bytes

6.4.4 Protocolul de menținere a coerenței și sincronizarea

Se utilizează un protocol de menținere a coerenței bazat pe director. În această schemă fiecărui nod îi este asignat un bloc de memorie partajată, denumit *nod local* ce menține intrările directorului pentru acel bloc. Fiecare intrare în director este un vector de aceeași lungime ca și numărul de noduri. Directorul menține de asemenea informația despre starea blocurilor. Ori de câte ori o copie a unui bloc de memorie este transmis spre un cache, bitul corespunzător acelui nod este poziționat. Un protocol de invalidare a fost implementat în cadrul căruia toate copiile cache-ate ale unui bloc sunt invalidate la o operație de scriere.

Sosirea unor mesaje în afara ordinii prestabilite provoacă probleme dacă protocolul de menținere a coerenței nu este modificat. De exemplu, o situație poate apare acolo unde un mesaj de invalidare neplanificat ajunge la un nod înainte ca data solicitată de o cerere de citire să fi ajuns. Dacă invalidarea este confirmată, ea poate conduce la o stare inconsistentă unde un nod crede că are un acces exclusiv la un bloc, în timp ce alt nod are o copie a blocului. Din această cauză se utilizează un protocol de menținere a coerenței modificat unde CC detectează dacă un mesaj a sosit în afara ordinii și îl reține pentru a fi prelucrat ulterior.

Metoda de sincronizare utilizată în cadrul sincronizării se bazează pe utilizarea operației TAS [MCS91]. Au fost implementate bariere utilizându-se un numărător partajat. Sarcina de calcul datorită existenței unui conflict între variabilele de sincronizare este semnificativă în unele aplicații.

6.4.5 Programe de evaluare a performanțelor

Au fost selectate câteva aplicații numerice pentru evaluarea performanței rețelei în SMP-MPD. Aceste aplicații sunt:

- multiplicarea a două matrici 2D – MATMUL
- algoritmul de selectare a căii cele mai scurte dintr-un graf –Floyd–Warshall–FWA
- factorizarea LU a unei matrici dense 2D – LU
- transformarea Fourier rapidă – FFT

Sunt aplicații clasice citate în [BINK97], [BIK00], [HP96].

MATMUL operează asupra a două matrici 128 X 128 în dublă precizie. Structurile de date principale sunt pentru tabloul 2D de numere reale: două matrici de intrare, o matrice transpusă şi una de ieşire. Datele partajate ocupă aproximativ 512 Kbytes.

Pentru FWA se utilizează un graf cu 128 de noduri cu ponderi aleatoare atribuite muchiilor. Structurile principale de date sunt două tablouri 2D de întregi: o matrice distanţă şi altă -matrice predecesor. Datele partajate ocupă aproximativ 128 Kbytes. Programul parcurge atâtea iteraţii câte vertice există şi în timpul unei iteraţii se citeşte de către toate procesoarele o linie anume din matricile distanţă respectiv predecesor. Fiecare iteraţie este urmată de o barieră.

Aplicaţia de descompunere LU utilizează o matrice 256 X 256 structurată în blocuri 8 X 8. Structura principală de date este un tabel 2D având prima dimensiune blocul iar a doua conţinând toate punctele de date din acel bloc. Astfel toate punctele, operate de acelaşi procesor, sunt alocate în mod contiguu.

Pentru aplicaţia FFT, la intrare se dau 2^{14} puncte. Structurile principale de date sunt două tablouri 2D de numere complexe. Nu există o partajare a datelor în primele $\log_2(N\text{-}P)$ etaje, unde N este numărul de puncte de date şi P este numărul de procesoare. În restul celor $\log_2 P$ etaje, fiecare punct de dată este partajat de două procesoare. În aceste etaje se întreţes tablouri de intrare şi de ieşire pentru a se evita erorile de conflict.

6.4.6 Rezultate ale simulării

Caracteristicile aplicaţiilor, luate în considerare în cadrul simulării, sunt prezentate în TABEL XVI. Aceste caracteristic sunt: numărul de referiri la memoria partajată, eşecurile în referirile la memoria partajată, numărul total de mesaje generate în timpul execuţiei.

TABEL XVI

Caracteristici ale aplicaţiilor utilizate în evaluare

Aplicaţia	Referinţe la memorie	Eşecuri acces cache	Mesaje generate
MATMUL	9,215,571	743,258	1,474,220
FWA	11,050,133	323,159	739,624
FFT	2,146,864	394,972	1,093,376
LU	112,270,761	706,579	1,671,854

Pentru început se prezintă latențele medii ale mesajelor în cele două rețele comparate – RMM şi RMB – când aplicațiile menționate anterior sunt rulate.

Valorile prezentate în TABEL XVII demonstrează că timpul de răspuns al mesajelor nu diferă semnificativ atunci când se utilizează RMM în loc de RMB.

TABEL XVII

Latențele medii ale mesajelor utilizând RMM şi RMB

Aplicația	**Latența RMB**	**Latența RMM**
MATMUL	224.48	231.25
FWA	491.24	506.74
FFT	183.86	187.26
LU	162.00	165.93

Performanța RMM într-un mediu mai apropiat de realitate este dată de o evaluare bazată pe timpul de execuție al acestor aplicații.

Simulatorul oferă timpul de execuție al unei aplicații în milioane de cicluri.

În fig. 6.31 se prezintă timpii de execuție pentru diferitele programe de aplicație anterior prezentate.

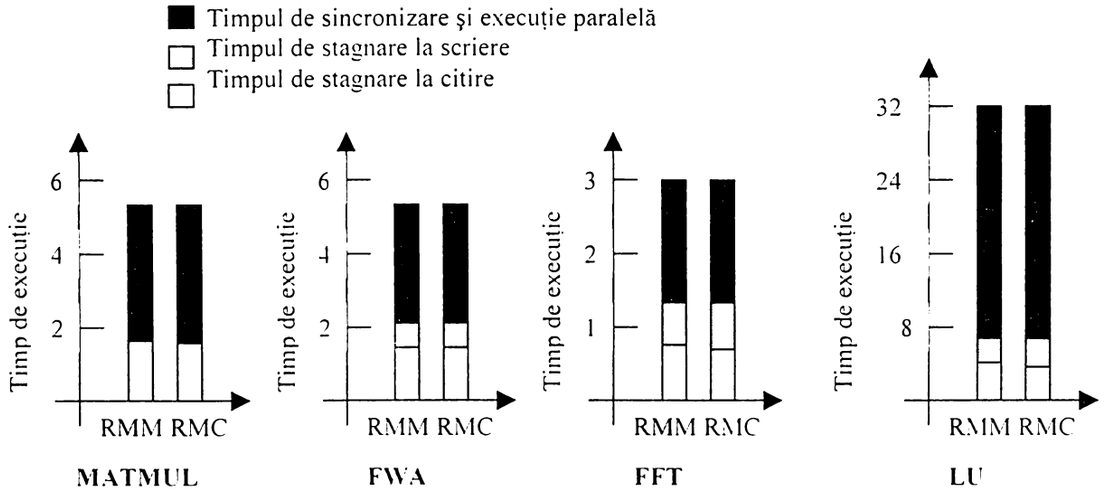


Fig. 6.31 Rezultate ale simulării bazate pe execuție

Timpii sunt repartizați astfel:

- timpul consumat pentru calcul și sincronizare
- timpul de staționare în citire
- timpul de staționare în scriere

Graful din fig. 6.31 este divizat în patru seturi de către două bare fiecare. Fiecare set este alocat unei anumite aplicații și prezintă timpul de execuție utilizând RMM în comparație cu RMB. Se observă că performanța RMM este foarte apropiată de cea a RMB. Ameliorarea timpului de execuție în general se ameliorează datorită timpului de staționare în citire și a timpului de staționare în scriere ce este direct proporțional cu latența rețelei. La RMM timpul de staționare în scriere este mai coborât decât cel de citire datorită unui numărm mai mic de eșecuri la scriere.

6.5 EFECTUL POLITICILOR DE MANAGEMENT AL MEMORIEI

Non-uniformitatea în accesele la memorie reprezintă un subiect ce are un impact semnificativ asupra performanțelor sistemelor multiprocesor cu memorie partajată distribuită. Alocarea și plasarea datelor în astfel de sisteme tinde să creeze puncte critice, din punct de vedere al acumulării de mesaje, în rețea și în modulele de memorie. În literatură [BR90], [RN91], [BIK00] se remarcă efortul cercetătorilor de a dezvolta abordări experimentale în reducerea sau chiar în eliminarea unor astfel de puncte critice. Toate aceste politici se referă la mișcarea *dinamică* a datelor utilizând politici de replicare și de migrare a lor. Aceste abordări măresc semnificativ costul total deoarece necesită resurse hardware suplimentare. În acest paragraf ne vom referi la organizări ale ierarhiei memoriilor și la politici de plasare în mod *static* a paginilor ce pot fi utilizate pentru a ameliora în mode semnificativ performanța.

Aceste politici sunt:

1. Întrețeserea în Grad Înalt (High-Order Interleaving) – IGI
2. Întrețeserea în Grad Redus (Low-Order Interleaving) – IGR
3. Plasarea Paginii cu Întrețeserea în Grad Înalt (Page Placement with High-Order Interleaving) – PPIGI

4. Plasarea Paginii cu Întreţeserea în Grad Redus (Page Placement with Low-Order Interleaving) – PPIGR
5. Plasarea Paginii bazat pe Tipul de Acces (Access Pattern based Page Placement with High-Order Interleaving) – PPAIGI

În primul tip de organizare – IGI, atât memoria privată cât şi cea partajată sunt strâns întreţesute.

A doua organizare reprezintă politica de întreţesere în grad redus. În figura 6.32 se prezintă organizarea modelului experimental în situaţia existenţei a 4 procesoare. Fiecare procesor P_i are o memorie cache C_i ce este conectată la RIN prin interfaţa I_i. Spaţiul de adresă al memoriei este divizat în două părţi, *memoria privată* PM şi *memoria partajată* SM. Memoria PM este uzual întreţesută într-un grad ridicat, în schimb SM, în termeni de blocuri, este slab întreţesută. Rezultatul constă în maniera distribuită în care blocurile de date partajate sunt accesate. Se evită astfel apariţia punctelor critice de acumulare a acceselor.

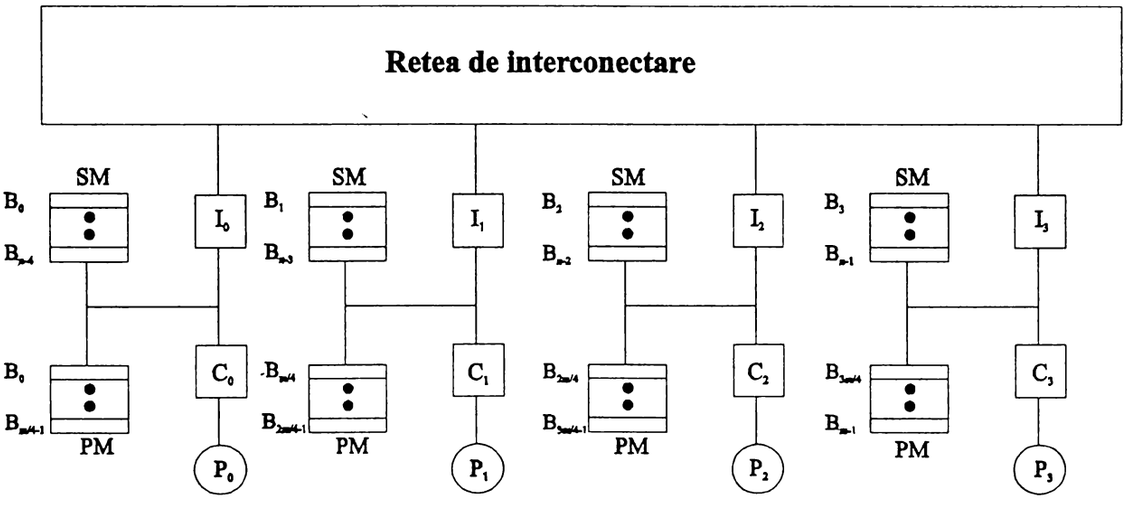


Fig. 6.32 Organizare IGR a unui SMP – MPD

În a treia, respectiv a patra organizare, se utilizează diferite politici de întreţesere pentru implementarea amplasării paginilor. În exemplul prezentat, amplasarea paginii se bazează pe apariţia erorii în procesor. Când un procesor se defectează la un acces al unui bloc, pagina corespunzătoare este extrasă dintr-o memorie secundară şi plasată în memoria locală a procesorului. Dacă spaţiul memoriei locale partajate a procesorului unde apare

eroarea nu este disponibil pentru plasarea paginii, atunci pagina este plasată la cel mai apropiat procesor.

A cincea politică este una în care aplicațiile sunt rulate cu o tehnică PPIGI şi numărul paginii accesate de către fiecare procesor este salvat ca şi informație de configurație de acces. Cu ajutorul acestor configurații de acces a paginii se defineşte un algoritm pentru determintarea celei mai bune amplasări atunci când informația este dată. Această politică însă este dificil de implementat deoarece pretinde cunoaşterea configurațiilor de acces ale paginii. Accesele la pagină ar putea fi estimate cu unele tehnici asistate de compilator [Iyer 96].

În literatură [Iyer 96] studiul acestor politici s-a efectuat prin incorporarea tehnicilor mai sus amintite în simulatoare pilotate de aplicație şi derivate din simulatorul Proteus. S-a introdus paramentrul *Page_Size* şi s-a optat pentru o valoare de 1024 a dimensiunii paginii. Atunci când dimensiunea tamponului se măreşte de la un pachet la două pachete, performanța se ameliorează semnificativ, dar în continuare, orice augmentare a taliei tamponului nu mai produce îmbunătățiri relevante.

6.5.1 Algoritm pentru determinarea amplasării paginilor bazată pe acces

În literatură [Iyer 96], se precizează că politica PPAIGI de gestionare a memoriei implică determinarea celei mai bune amplasări a datelor când sunt cunoscute *apriori* câteva informații de către alocaturul de memorie. Cea mai utilizată informație o reprezintă numărul de accese la oricare pagină ce sunt efectuate din partea oricărui procesor. Algoritmul prezentat parcurge o serie de iterații, având această informație, pentru a determina cea mai bună amplasare a fiecărei pagini. Se utilizează un tabel bidimensional *P_to_p* ce reprezintă harta de alocare *procesor la pagină*.

Algoritmul este următorul:

1. Pentru fiecare pagină, se extrage nodul al cărui procesor efectuează numărul maxim de accese şi se introduce această valoare în tabelul *P_to_p*.
2. Pentru fiecare nod, se extrag atât de multe pagini cât se pot potrivi din linia respectivă în tabelul *P_to_p* cu scopul de a reduce numărul de accese.
3. Se verifică dacă toate paginile au fost alocate.
4. Dacă pasul 3 este *adevărat* atunci *exit*.

5. Dacă pasul 3 este *fals*, atunci:
    a) Se extrag paginile anterior alocate din lista de alocare
    b) Se extrag procesoarele selectate de către paginile nealocate din listele lor individuale şi
    c) Se porneşte din nou de la pasul 1 până când toate paginile disponibile devin memorie alocată

Algoritmul este astfel proiectat încât paginile şi procesoarele sunt ambele beneficiare ale alocării. Aceasta ar putea reduce numărul de accese la un nod îndepărtat deoarece procesoarele selectează o pagină să fie în memoria locală a nodului său dacă frecvența de acces este ridicată.

Deficiența majoră a acestui algoritm constă în faptul că informația este reprezentată doar de *numărul de accese* şi nu de *perioada acceselor*. Dacă o etichetă de marcare a timpului este asociată fiecărui acces, s-ar putea ameliora semnificativ algoritmul, astfel încât pagina să se plaseze perfect. Cea mai potrivită informație de care este nevoie este calea critică în execuție.

6.6 CONCLUZII

Sistemul multiprocesor utilizat ca şi model experimental este un sistem cu memorie partajată distribuită de tip CC-NUMA.

Rețeaua de interconectare este o Rețea cu Magistrală Multinivel. S-a analizat RMM şi s-a demonstrat potențialul său în raport cu RIN existente. Inițial RMM a fost comparată cu o RIN convențională bidirecțională (RMB). S-au prezentat patru sisteme de autorutare pentru RMM ce sunt valabile şi pentru RMB. A fost prezentat un algoritm pentru selectarea căii de distanță minimă dintre o sursă şi o destinație. Acest algoritm reduce numărul de niveluri traversate în RIN.

Pentru evaluarea performanțelor RMM şi ale RMB s-au utilizat analize probabilistice şi de formare a cozilor de aşteptare. Simulările au confirmat că RMM se comportă mai bine în comparație cu RMC şi similar în raport cu RMB. A rezultat că scalabilitatea unui RMM este aproape identică cu cea a unui RMB.

Performanța unui RMM este similară cu cea a unui RMB în termeni de timp de răspuns şi de utilizare a procesorului. Ea este însă mai bună decât a unui RMC.

Pentru a reliefa potențialul de performanță al unei RMM, a fost prezentată o evaluare bazată pe execuție într-un mediu multiprocesor cu memorie partajată, coerent din punct de vedere cache. Mecanismul de asigurare a coerenței cache se bazează pe director.

Simulatorul bazat pe execuție utilizat este PROTEUS. S-a evidențiat că pentru un număr de patru aplicații performanța RMM este aproximativ egală cu cea a unei RMB în termeni de latență a mesajului şi timpi de execuție. Întrucât RMM oferă o simplitate din punct de vedere hardware, acest tip de rețea reprezintă o alternativă viabilă la RIN existente. În plus, există facilități de toleranță la defect incorporate. Studiul politicilor de gestionare a memoriei scot în evidență faptul că modul de plasare a paginii depinde puternic de configurațiile de acces ale aplicațiilor şi sunt necesare tehnici de investigare în termeni de politici de mişcare în mod dinamic a paginii. Se evidențiază că politica de plasare a paginii cu un grad coborât de întrețesere reprezintă cea mai bună alternativă deoarece ameliorează întrețeserea cu grad redus prin plasarea primului bloc la procesorul defect.

7. CONCLUZII GENERALE ŞI DIRECŢII VIITOARE DE CERCETARE

A. Concluzii generale

Lucrarea de faţă vizează un domeniu actual din ştiinţa calculatoarelor, acela al arhitecturilor paralele din clasa MIMD cu facilităţi de toleranţă la defect.

Actualitatea problematicii abordate în teză rezidă din faptul că sistemele multiprocesor au devenit industriale şi prin urmare, pe lângă performanţa ce validează anumite concepte teoretice, dependabilitatea acestor sisteme devine un factor determinant. Pe cale de consecinţă, trebuie abordată proiectarea sistemelor multiprocesor la nivel de arhitectură, prin prisma unui raport corect între performanţă şi dependabilitate.

Lucrarea este concepută ca şi o dezvoltare succesivă a temelor majore ce apar în *clasificarea*, *analiza*, *proiectarea* şi *simularea* SMP din clasa MIMD. Fiecare capitol se finalizează cu concluzii şi recomandări ce contribuie la abordarea mai eficientă a tematicilor conţinute în capitolul următor.

Capitolul 1 este dedicat prezentării principalelor tendinţe arhitecturale în dezvoltarea SMP cât şi a unor noi concèpte ce permit o abordare mult mai nuanţată a arhitecturilor paralele. Examinarea sistematică, în termeni calitativi şi cantitativi, a diferitelor arhitecturi se face utilizând conceptul de *convergenţă* între paradigmele *memorie partajată* şi *transfer de mesaje*. Prin prisma acestui concept sunt descrise în continuare patru teme majore pentru proiectare: *modelul de comunicare*, *modelul de programare*, *comunicarea şi replicarea*, *performanţa*. *Coerenţa* şi *consistenţa* memoriilor cache reprezintă subiecte majore în abordarea corectă a reducerii latenţei comunicaţiei. Diferite modele de consistenţă a memoriei au fost tratate ulterior în Capitolul 2 şi în Capitolul 3. *Arhitectura reţelelor de interconectare* contribuie determinant la reducerea latenţei şi la tolerarea unei creşteri semnificative a procesoarelor conectate, menţinând scalabilitatea. Tot în acest capitol sunt prezentate principalele metode de obţinere a tolerării latenţei: *reducerea timpului de acces la fiecare nivel al unei ierarhii extinse de memorie*, *structurarea sistemului pentru a reduce frecvenţa acceselor cu latenţă ridicată*, *structurarea aplicaţiei pentru a reduce frecvenţa acceselor cu latenţă ridicată*. În fine, se introduce *toleranţa la defect* ca şi un obiectiv major în proiectarea SMP şi în evaluarea performanţei sale.

În Capitolul 2 se prezintă o analiză a arhitecturilor SMP din clasa MIMD din perspectiva conceptelor prezentate anterior.

Se constată că în ceea ce priveşte SMP-MP simetrice elementele cheie din punct de vedere tehnic sunt organizarea şi implementarea sistemului de memorie partajată, care este utilizată pentru comunicarea între procesoare, suplimentar la gestionarea tuturor acceselor la memorie. Cele mai multe dintre SMP-MP cu număr redus de procesoare utilizează magistrala sistemului ca şi un mediu de interconectare pentru comunicare şi problemele care se ridică sunt menţinerea coerenţei datelor partajate în MCH private ale procesoarelor şi asigurarea toleranţei la defect. Există o mare varietate de opţiuni disponibile arhitectului de sistem, incluzând seturi de stări asociate cu blocurile din MCH, acţiuni şi tranzacţii de magistrală, alegerea dimensiunii unui bloc cache şi protocoale de actualizare şi invalidare. Sarcina principală a arhitectului de sistem este de a face o corectă alegere între menţinerea în limite acceptabile ale configuraţiilor de date partajate şi de a face implementarea lor cât mai uşoară.

Se poate aştepta ca SMP-MP de scară redusă să continue să rămână atractive din cel puţin trei motive. Primul îl reprezintă raportul cost/performanţă atractiv. În al doilea rând microprocesoarele actuale sunt proiectate să fie integrate în medii multiprocesor, ele având tot setul şi toate facilităţile necesare interfaţării în astfel de sisteme. În fine, al treilea motiv îl reprezintă existenţa unui suport software performant pentru maşinile paralele.

Deşi interconectarea prin magistrala partajată reprezintă un mecanism puternic, există în mod clar limitări în ceea ce priveşte lăţimea de bandă pe măsură ce numărul de procesoare sau viteza procesorului cresc. O soluţie generală în construcţia de SMP-MP coerente din punct de vedere cache şi scalabile, este aceea de a distribui fizic memoria printre noduri şi de a utiliza o interconectare scalabilă împreună cu protocoale de menţinere a coerenţei ce nu se bazează pe monitorizare. Indiferent de evoluţie, plasarea reţelelor de interconectare în cadrul ierarhiei de memorie, problema menţinerii coerenţei cache şi diferite protocoale de menţinere a coerenţei la nivel de tranziţie de stare sunt fundamentale pentru toate arhitecturile SMP-MP coerente din punct de vedere cache, independent de modul de interconectare utilizat.

Cele mai moderne sisteme construite pe scară largă sunt proiectate pornind de la noduri de procesare cu utilizare generală, dotate cu ierarhie de memorie, ce au în plus şi facilităţi de interfaţare, până la reţele scalabile. Este disponibilă o gamă largă de opţiuni de proiectare pentru asistarea facilităţilor de comunicare. Proiectarea este puternic influenţată de nivelul unde are loc interfaţarea dintre facilitatea de comunicare cu arhitectura nodūlui: nivelul procesorului, al controllerului MCH, al magistralei de memorie sau al magistralei I/E. Este de asemenea influenţată de arhitectura de comunicaţie şi de modelul de programare. Modelele de programare sunt implementate utilizând protocoale ce se bazează pe primitive de tip tranzacţii de reţea.

Proiectarea SMP este strâns legată de revoluţia tehnologică a anilor '80 – microprocesorul şi comutatorul de reţea scalabilă încapsulate fiecare într-un singur chip. Ca şi rezultat, proiectarea maşinilor paralele s-a divizat în două direcţii: SMP orientate în jurul conceptelor de transfer de mesaj, de grupuri de procesoare, etc. şi SMP ce integrează într-un grad ridicat RIN în sistemul de memorie pentru a furniza un acces coerent cache la un spaţiu global de adresă fizică.

În Capitolul 3 se analizează performanţa diferitelor arhitecturi de SMP. În urma analizei a trei tipuri de RIN şi anume: RIN-MM, RIN-MN şi RIN-CB se constată că RIN-CB manifestă cea mai bună probabilitate de acceptanţă în raport cu creşterea numărului de procesoare având o lăţime de bandă scalabilă şi o bună utilizare a procesorului. Toleranţa la defect este acceptabilă, cel mai bun comportament însă în acest sens fiind manifestat de RIN-MM. Sub raport performanţă/cost la un număr mare de procesoare, RIN-MN rămâne însă soluţia cea mai bună. Utilizarea unor tehnici moderne de transmisie a mesajelor ca şi tehnica *wormhole*, cât şi introducerea unor capacităţi redundante încorporate în reţea elimină în mare măsură deficienţele evidenţiate anterior în urma analizei. În acest sens, reţelele ASEN devin atractive prin faptul că pot deveni tolerante la mai mult de un defect.

Reducerea semnificativă a latenţei unui SMP poate fi obţinută şi prin utilizarea MCH private în cadrul unei ierarhii de memorie. Există însă o interdependenţă între caracteristicile arhitecturale şi mecanismele de menţinere a coerenţei. Aceste caracteristici sunt: *strategia de detectare a coerenţei, strategia de întărire a coerenţei,*

dimensiunea informației referitoare la partajarea blocului, dimensiunea blocului cache cât şi altele, mai puțin importante dar nu de neglijat.

În arhitecturile moderne de SMP se pot combina diferite mecanisme de menținere a coerenței bazate pe *snooping* împreună cu cele bazate pe *director*. Abordările hibride vor oferi cea mai bună oportunitate pentru creşterea performanței şi pentru reducerea costului de implementare.

Capitolul 4 este dedicat analizei unor arhitecturi MIMD tolerante la defect. Se analizează mai multe abordări propuse pentru parcurgerea celor trei faze ale toleranței la defect dinamice: *detectarea* defectului şi *localizarea* lui, *recuperarea* din defect şi *reconfigurarea* sistemului în jurul procesorului defect. Referitor la recuperarea din defect câteva concluzii se impun şi anume: tehnicile de recuperare sunt diferite pentru SMP-MP, SMP-TM şi SMP-MPD. Pe de altă parte atât *rularea înapoi* cât şi *rularea înainte* utilizează tehnici de creare şi de exploatare a PV-urilor. Ierarhia de memorie oferă soluții de amplasare atât pentru datele active cât şi pentru datele PV-urilor. Atât recuperarea din defect prin *rulare înapoi* cât şi prin *rulare înainte* reprezintă procese complexe în care crearea corectă şi gestionarea PV-urilor sunt cele mai importante aspecte. Totuşi, coordonarea creeri şi exploatării PV-urilor este neglijabilă în comparație cu sarcina de salvare a stărilor.

Referitor la proiectarea şi utilizarea RIN-MN tolerante la defect se constată existența unor caracteristici comune ce includ *modelul de defect, criteriul de toleranță la defect, metoda de creare a toleranței la defect, toleranța la defecte singulare sau multiple, complexitatea rutării* şi *complexitatea hardware*. Tehnicile de obținere a toleranței la defect sunt în general independente de topologia rețelei. Acestea sunt: transmisia multiplă pe căi distincte a aceleiaşi informații, parcurgerea de mai multe ori a RIN-MN, etc.

În faza de proiectare însă se pot prevedea tehnici ce modifică topologia unei RIN-MN şi care în general se caracterizează prin intervenții hardware la nivel de: comutator, niveluri de comutatoare, conexiuni suplimentare, etc. În ultimă instanță replicarea *in integrum* a unei RIN-MN reprezintă o soluție fiabilă dar costisitoare. Finalitatea ultimă însă o reprezintă stabilirea unui raport corect performanță/cost în ceea ce priveşte gradul de acoperire la defect.

Datorită unor condiții restrictive ce au fost expuse pe larg în acest capitol se constată că doar câteva din RIN-MN sunt tolerante la defect singular. Condiția creşterii numărului de rețele tolerante la acest tip de defect o reprezintă relaxarea modelului de defect comun şi presupunerea unei siguranțe ridicate în funcționare a nivelurilor de comutatoare. Aria de cercetare a RIN-MN tolerante la defect este departe de a fi epuizată iar abordarea noilor tehnologii hardware şi software în contextul mai larg al dependabilității vor ameliora semnificativ performanța acestora.

În Capitolul 5 se tratează modelarea dependabilității pentru SMP-MP. Sunt prezentate etapele preliminare în descrierea unui model care sunt: *definirea*, *parametrizarea* şi *evaluarea*. Modelele de fiabilitate luate în considerare sunt cele de tip *terminal*, de tip *multiterminal* bazat pe sarcină şi de tip *rețea*. Se consideră fiabilitatea ca fiind una dintre cele mai importante atribute ale SMP. Cele mai utilizate tehnici de analiză de fiabilitate sunt *modelarea combinatorie* şi *modelarea prin lanțuri Markov*.

Modelarea combinatorie bazată pe tehnici din teoria probabilității reliefează modurile diferite în care un sistem poate rămâne operațional. Fiabilitatea sistemului este în general exprimată în termeni de fiabilități ale componentelor individuale. În practică se utilizează două abordări şi anume *modelarea serie* respectiv *modelarea paralel*. Se remarcă un dezavantaj major al metodei combinatorii şi anume faptul că sistemele complexe nu pot fi modelate convenabil deoarece expresiile fiabilității devin foarte complexe. De asemenea acoperirea defectului este dificil de incorporat în expresiile combinaționale.

Modelele Markov utilizează concepte de *stare* şi de *tranziție a stării*. Principalul beneficiu al utilizării MK în modelarea dependabilității îl reprezintă abilitatea de a gestiona acoperirea defectului într-o manieră sistematică. Incorporarea acoperirii C în analiza MK permite analiza siguranței sistemului.

Modelarea disponibilității presupune incorporarea conceptului de *reparare*. Lanțurile Markov au proprietatea de a fi ciclice şi, prin urmare, se permite reîntoarcerea de la o stare mai puțin operațională la o stare mai operațională sau chiar complet operațională. Evaluarea experimentală a disponibilității nu este întotdeauna posibilă din cauza constrângerilor de timp şi de cost. Din această cauză se utilizează frecvent parametrii *MTTF* sau *MTTR* ceea ce conduce la evaluarea disponibilității stării ferme, A_{SS}.

În fine, se prezintă modele de dependabilitate raportată la performanță introducându-se noțiunea de *performabilitate* ce reprezintă funcția de distribuție a probabilității performanței de sistem acumulată în timp.

Se abordează în mod sistematic modelarea disponibilității utilizând *descompunerea de sistem* pentru SMP-MP orientate pe RIN-MN. Se tratează configurații cu 16 noduri şi un model generalizat.

Arhitectura SMP-MP, utilizat ca şi model experimental, este prezentată în Capitolul 6. Acest sistem este unul cu memorie partajată distribuită, coerent din punct de vedere cache, şi cu acces la memorie de tip NUMA. Rețeaua de interconectare este o rețea de tip RMM. S-au considerat 16 noduri procesoare şi o ierarhie de memorie structurată pe 3 straturi. S-a analizat RMM şi s-a demonstrat potențialul său în raport cu RIN existente. Rețelele de interconectare luate în considerare la comparație au fost o RIN convențională bidirecțională (RMB) şi o RIN convențională (RMC). S-au elaborat doi algoritmi pentru cercetarea comportării rețelei RMM. Primul algoritm determină numărul de căi posibile între o sursă şi toate destinațiile, în timp ce al doilea determină lungimea minimă de cale între o sursă şi o destinație dată. Rezultatele au fost prezentate în anexele 1, 2 şi 3. S-au evidențiat tehnicile de rutare RD, RI, BD şi BI.

Pentru evaluarea performanțelor RMM, RMB şi RMC s-au utilizat analize probabilistice şi de formare a cozilor de aşteptare. S-a constatat că RMM se comportă mai bine în comparație cu RMC şi similar în raport cu RMB. Scalabilitatea unui RMM este aproximativ similară cu cea a unui RMB. Performanța unui RMM este similară cu cea a unui RMB în termeni de timp de răspuns şi de utilizare a procesorului. Avantajul RMM îl reprezintă însă o simplificare din punct de vedere hardware al comutatorului în comparație cu comutatorul clasic *crossbar* utilizat în RMB. Există şi facilități de toleranță la defect incorporate. Simulatorul bazat pe execuție care a fost utilizat este Proteus elaborat în laboratoarele MIT.

B. Direcții de cercetare

În ultima perioadă de timp se constată o convergență semnificativă între arhitecturile SMP-MP şi SMP-TM. Cele mai recente SMP-MPD utilizează arhitecturi bazate pe accesul neuniform la memorie şi coerente cache (CC-NUMA).

Performanța unui calculator este primordial dictată de existența unei latențe reduse cât şi de furnizarea unei mari lățimi de bandă. Performanța raportată la dependabilitate reprezintă un raport corect ce trebuie stabilit între o performanță uşor diminuată în avantajul unei toleranțe la defect confortabile. Evident, raportul cost/performanță în aceste condiții trebuie să fie unul cât mai acceptabil. Prin urmare, direcțiile viitoare de cercetare în proiectarea SMP-MPD tolerante la defect trebuie să vizeze *reducerea* şi *tolerarea latenței*, *creşterea lățimii de bandă cu păstrarea scalabilității* şi *creşterea toleranței la defect*.

O direcție de cercetare viitoare o reprezintă proiectarea comutatoarelor cu capacități de memorie cache incorporate. Procesoarele uzuale ca şi Pentium Pro şi MIPS R10000 sunt superscalare, utilizând niveluri multiple de memorii cache şi tolerând mai multe eşecuri în acces decât microprocesoarele clasice. Proprietatea de superscalaritate are trei implicații în traficul prin rețea ce trebuie să fie studiate atent. În primul rând, intensitatea traficului poate fi redusă datorită nivelurilor multiple de cache-are. În al doilea rând, numărul de cereri de acces simultane poate creşte semnificativ. În final, analizarea impactului latenței RIN asupra timpului de stagnare a aplicației va deveni dificilă datorită suprapunerii comunicării cu calculul.

Proiectarea comutatoarelor *crossbar* sau cu *magistrală inclusă* trebuie să adopte soluții hardware de cache-are. De exemplu înglobarea unor MCH de tip SRAM în comutatoarele *crossbar* ar permite captarea datelor partajate pe măsură ce ele parcurg RIN şi apoi livrarea lor, în mod eficient, către următorii solicitanți.

În ceea ce priveşte politicile de management al memoriei, utilizarea unor politici de management dinamic poate reduce solicitările ce congestionează traficul prin RIN, generate de aplicații, printr-o mişcare optimă a datelor spre memoria locală.

Se cunoaşte că utilizarea unor tehnici de mascare a latenței, ca de exemplu aducerea în avans a datelor din memorie, poate produce o degradare semnificativă în performanță.

Dacă însă aducerea în avans a datelor se produce în MCH utilizate ca şi comutatoare, atunci aceste tehnici îşi recâştigă eficienţa şi ameliorează semnificativ performanţa.

C. Contribuţii personale ale autorului

Pornind de la obiectivele declarate ale acestei dizertaţii, în continuare se evidenţiază principalele contribuţii originale, structurate pe capitole:

1.1. Abordarea sistematică a arhitecturilor sistemelor multiprocesor din clasa MIMD, utilizând concepte moderne ca şi: *convergenţa arhitecturilor paralele, evaluarea şi analizarea arhitecturilor prin aplicaţii, metodologii sistematice pentru evaluarea performanţei.*

1.2. Utilizarea în cadrul caracterizării şi analizei diferitelor arhitecturi a unor concepte de proiectare fundamentale ca şi: *modelul de comunicare, modelul de programare, comunicarea şi replicarea, performanţa.*

1.3. Utilizarea în analiza arhitecturilor sistemelor multiprocesor cu memorie partajată a unor concepte-cheie ca şi: *coerenţa memoriilor cache, consistenţa memoriei, sincronizarea.*

2.1. Prezentarea unei taxonomii a schemelor de asigurare a coerenţei memoriilor cache

2.2. Sistematizarea prezentării reţelelor de interconectare în funcţie de: structura organizatorică, topologia de interconectare, mecanisme de rutare, structura comutatorului.

3.1. Prezentarea sistematică a metodelor de analiză de performanţă la sistemele multiprocesor din clasa MIMD.

3.2. Analiza de performanţă la reţelele de interconectare cu prezentarea comparativă a rezultatelor.

3.3. Sistematizarea unor scheme şi a unor protocoale de asigurare a coerenţei memoriilor cache la multiprocesoarele cu memorie partajată, respectiv la multiprocesoarele cu transfer de mesaj.

3.4. Evidenţierea şi sistematizarea caracteristicilor arhitecturale ce afectează mecanismele de menţinere a coerenţei memoriilor cache.

4.1. Sintetizarea unor parametri ce definesc *dependabilitatea* ca fiind calitatea serviciilor furnizate de un sistem de calcul.

4.2. Abordarea sistematică a metodelor de simulare a toleranței la defect, a strategiilor de recuperare din defect şi a tehnicilor de reconfigurare în sistemele multiprocesor.

4.3. Sistematizarea metodelor şi tehnicilor de obținere a toleranței la defect pentru sistemele multiprocesor.

5.1. Abordarea într-o variantă unitară a modelelor de dependabilitate şi raportarea lor la performanțe.

5.2. Abordarea modelării disponibilității, utilizând descompunerea de sistem pentru sistemele multiprocesor cu memorie partajată orientate pe rețele de interconectare multinivel.

6.1. Definirea unei arhitecturi de sistem multiprocesor cu memorie partajată distribuită având 16 noduri procesoare în vederea evaluării performanței.

6.2. Proiectarea unei rețele de interconectare cu magistrală multinivel (RMM) de dimensiuni 16 X 16.

6.3. Elaborarea unui algoritm şi a programului de calcul pentru determinarea tuturor rutelor posibile într-o RMM cu N=16, 64, 256, 1024 de noduri cât şi a numărului minim de comutatoare traversate.

6.4. Elaborarea unui algoritm şi a programului de calcul pentru determinarea căii optime într-o RMM cu N=16, 64, 256, 1024 de noduri.

6.5. Evaluarea performanțelor rețelei cu magistrală multinivel (RMM), rețelei multinivel bidirecțională (RMB) şi a rețelei multinivel convențională (RMC) pentru a evidenția comportamentul RMM.

6.6. Utilizarea simulatorului PROTEUS

ANEXA 1

Program de determinare a numărului de căi pentru fiecare tip de rutare între toate sursele şi destinaţiile cât şi de calcul a lungimilor minime ale căilor.

```
#include <stdio.h>
#define D 256 //numar procesoare
#define MARGD 15 // log2(D)*2-1 - indicele marginii din dreapta

int stiva[2*D][3]; //stiva pentru determinarea rutelor
int stat[D][4]; //vector de numarare a rutelor
int min[D][4]; //vector de calcul a minimelor

// initializeaza vectorul de numarare a rutelor
void init_stat(void) {
      int i;

      for(i=0;i<D;i++) {
            stat[i][0]=0;
            stat[i][1]=0;
            stat[i][2]=0;
            stat[i][3]=0;
            }
      }

// tipareste vectorul de numarare a rutelor
void print_stat(void) {
      int i;

      for(i=0;i<D;i++)
            if (i!=stiva[1][0])

      printf("%d\t%d\t%d\t%d\t%d\n",i,stat[i][0],stat[i][1],stat[i][2],stat[i][3
]);
            else
                  printf("%d\t-\t-\t-\n",i);
      }

//initializeaza vectorul de calcul al minimelor
void init_min(void) {
      int i;

      for(i=0;i<D;i++) {
            min[i][0]=-1;
            min[i][1]=-1;
            min[i][2]=-1;
            min[i][3]=-1;
            }
      }

//calculeaza minimul pt o pozitie si un tip de cale
void save_min(int k,int i) {
      int j;
```

```
	j=stiva[k+1][0];
	if ((min[j][i]==-1) || (min[j][i]>(k+1)/2))
		min[j][i]=(k+1)/2;
	}

//tipareste vectorul de calcul al minimelor
void print_min(void) {
	int i;

	for(i=0;i<D;i++)
		if (i!=stiva[1][0])

	printf("%d\t%d\t%d\t%d\t%d\n",i,min[i][0],min[i][1],min[i][2],min[i][3]);
		else
			printf("%d\t0\t0\t0\t0 \n",i);
	}

//initializeaza stiva pe nodul sursa
void init_st(int k, int s, int margin) {

	stiva[k][0] = s; stiva[k][1] = margin; stiva[k][2] = 0;

	}

//rotire stanga
int rol (int i) {
	int tmp, ret;

	tmp = i % 2;
	ret = i / 2;
	if (tmp)
		ret += D / 2;
	return ret;
	}

//rotire dreapta
int ror (int i) {
	int tmp=0, ret;

	if (i>=D/2) {
		tmp = 1;
		i -= D/2;
		}
	ret = i*2 + tmp;
	return ret;
	}

//genereaza succesorul unui nod
int succ(int k) {

	int i=stiva[k][0];
	int j=stiva[k][1];
	int dir=stiva[k][2];

	if (dir < 4) {

		//in functie de directia (0-3) selectata genereaza succesorul
```

```
      switch (dir) {

            //continuarea sensului pe acelasi nivel orizontal
            case 0: stiva[k+1][0] = i;
                  stiva[k+1][2] = 0;
                  if (j % 2 == 0)
                        stiva[k+1][1] = j+1;
                  else
                        stiva[k+1][1] = j-1;
                  break;

            //continuarea sensului pe celalalt nivel orizontal
            case 1: stiva[k+1][2] = 0;
                  if (j % 2 == 0) {
                        stiva[k+1][1] = j+1;
                        if (i % 2 == 0)
                              stiva[k+1][0] = i+1;
                        else
                              stiva[k+1][0] = i-1;
                        }
                  else {
                        stiva[k+1][1] = j-1;
                        if (i % 2 == 0)
                              stiva[k+1][0] = i+1;
                        else
                              stiva[k+1][0] = i-1;
                        }
                  break;

            //intoarcere
            case 2: stiva[k+1][2] = 0;
                  stiva[k+1][1] = j;
                  if (i % 2 == 0)
                        stiva[k+1][0] = i+1;
                  else
                        stiva[k+1][0] = i-1;
                  break;

            //cont sensului pe urmatorul nivel de comutatoare
            case 3: stiva[k+1][2] = 0;
                  if (j % 2 == 0) {
                        stiva[k+1][0] = rol(i);
                        stiva[k+1][1] = j-1;
                        }
                  else {
                        stiva[k+1][0] = ror(i);
                        stiva[k+1][1] = j+1;
                        }
                  break;

      }

      stiva[k][2]++;
      return 1;
      }
else
      return 0;
```

```
	}

// verificarea validitatii nodului urmator
int valid (int k) {

	int i;
	int bucle=0;

	//excludere marginilor
	if (stiva[k+1][0] < 0 || stiva[k+1][0] >= D)
		return 0;
	if (stiva[k+1][1] < 0 || stiva[k+1][1] > MARGD)
		return 0;

	//excluderea buclelor inchise
	for(i=1; i<=k; i++)
		if (stiva[i][0]==stiva[k+1][0] && stiva[i][1]==stiva[k+1][1])
			return 0;
	//excluderea repetarii unor segmente la infinit
	if ((stiva[k-1][2]-1==3) && (stiva[k][2]-1==3))
		return 0;
	if ((stiva[k][2]-1!=3) && (stiva[k-1][2]-1!=3))
		return 0;
	//excluderea cailor cu mai mult de o intoarcere
	for (i=1; i<k; i++)
		if (stiva[i][2]-1==2)
			bucle++;
	if (bucle>1)
		return 0;
	return 1;

	}

//verificarea ajungerii la solutie
int solutie (int k) {

	if ((stiva[k+1][1]==MARGD) || (stiva[k+1][1]==0))
		return 1;
	return 0;
	}

//inregistrarea solutiei
void inregistrare (int k) {

	int i;
	int j;

	j=stiva[k+1][0];
	for(i=0;i<=k;i++)
		if (stiva[i][2]-1==2) {
			if (stiva[1][1]==0) {
				stat[j][1]++;
				save_min(k,1); }
			else {
				stat[j][3]++;
				save_min(k,3); }
			return;
```

```
                }
        if (stiva[1][1]==0) {
                stat[j][0]++;
                save_min(k,0); }
        else {
                stat[j][2]++;
                save_min(k,2); }
/*      for(i=1;i<=k;i++)
                printf("%d %d %d\t",stiva[i][0],stiva[i][1],stiva[i][2]-1);
        printf("\n");*/
        }

//generarea datelor pentru un nod sursa
void genereaza (int s, int m) {

        int k=1;
        int as, ev;

        stiva[0][2]=4;
        init_st(1,s,m);
        while (k >= 0) {

                do {
                        as = succ(k);
                        if (as)
                                ev = valid(k);

                } while ((as) && !(as && ev));

                if (as)
                        if (solutie(k))
                                inregistrare(k);
                        else {
                                k++;
                                }
                else
                        k--;
        }
}

void main (void) {
        int i;
        //selectarea fiecarui nod sursa pt generare date
        for(i=0;i<D;i++) {
                init_stat();
                init_min();
                genereaza(i,0);
                genereaza(i,MARGD);
                printf("Nr cai Sursa: %d\n",i);
                printf("DEST\tRD\tBD\tRI\tBI\n");
                print_stat();
                printf("Nr minim comutatoare Sursa: %d\n",i);
                printf("DEST\tRD\tBD\tRI\tBI\n");
                print_min();
                printf("\n");
        }
}
```

ANEXA 2

Numărul de căi între o sursă dată şi toate destinaţiile şi lungimile minime ale căilor pentru fiecare tip de rutare într-o RMM 16 X 16

Număr de căi **S=0**

D	RD	BD	RI	BI
0	0	0	0	0
1	1	1	1	1
2	1	8	1	2
3	1	8	1	2
4	1	4	1	4
5	1	4	1	4
6	1	8	1	4
7	1	8	1	4
8	1	2	1	8
9	1	2	1	8
10	1	8	1	8
11	1	8	1	8
12	1	4	1	8
13	1	4	1	8
14	1	8	1	8
15	1	8	1	8

Număr minim de comutatoare **S=0**

D	RD	BD	RI	BI
0	0	0	0	0
1	4	1	4	1
2	4	7	4	3
3	4	7	4	3
4	4	5	4	5
5	4	5	4	5
6	4	7	4	5
7	4	7	4	5
8	4	3	4	7
9	4	3	4	7
10	4	7	4	7
11	4	7	4	7
12	4	5	4	7
13	4	5	4	7
14	4	7	4	7
15	4	7	4	7

Număr de căi **S=1**

D	RD	BD	RI	BI
0	1	1	1	1
1	0	0	0	0
2	1	8	1	2
3	1	8	1	2
4	1	4	1	4
5	1	4	1	4
6	1	8	1	4
7	1	8	1	4
8	1	2	1	8
9	1	2	1	8
10	1	8	1	8
11	1	8	1	8
12	1	4	1	8
13	1	4	1	8
14	1	8	1	8
15	1	8	1	8

Număr minim de comutatoare **S=1**

D	RD	BD	RI	BI
0	4	1	4	1
1	0	0	0	0
2	4	7	4	3
3	4	7	4	3
4	4	5	4	5
5	4	5	4	5
6	4	7	4	5
7	4	7	4	5
8	4	3	4	7
9	4	3	4	7
10	4	7	4	7
11	4	7	4	7
12	4	5	4	7
13	4	5	4	7
14	4	7	4	7
15	4	7	4	7

Număr de căi **S=2**

D	RD	BD	RI	BI
0	1	8	1	2
1	1	8	1	2
2	0	0	0	0
3	1	1	1	1
4	1	8	1	4
5	1	8	1	4
6	1	4	1	4
7	1	4	1	4
8	1	8	1	8
9	1	8	1	8
10	1	2	1	8
11	1	2	1	8
12	1	8	1	8
13	1	8	1	8
14	1	4	1	8
15	1	4	1	8

Număr minim de comutatoare **S=2**

D	RD	BD	RI	BI
0	4	7	4	3
1	4	7	4	3
2	0	0	0	0
3	4	1	4	1
4	4	7	4	5
5	4	7	4	5
6	4	5	4	5
7	4	5	4	5
8	4	7	4	7
9	4	7	4	7
10	4	3	4	7
11	4	3	4	7
12	4	7	4	7
13	4	7	4	7
14	4	5	4	7
15	4	5	4	7

Număr de căi **S=3**

D	RD	BD	RI	BI
0	1	8	1	2
1	1	8	1	2
2	1	1	1	1
3	0	0	0	0
4	1	8	1	4
5	1	8	1	4
6	1	4	1	4
7	1	4	1	4
8	1	8	1	8
9	1	8	1	8
10	1	2	1	8
11	1	2	1	8
12	1	8	1	8
13	1	8	1	8
14	1	4	1	8
15	1	4	1	8

Număr minim de comutatoare **S=3**

D	RD	BD	RI	BI
0	4	7	4	3
1	4	7	4	3
2	4	1	4	1
3	0	0	0	0
4	4	7	4	5
5	4	7	4	5
6	4	5	4	5
7	4	5	4	5
8	4	7	4	7
9	4	7	4	7
10	4	3	4	7
11	4	3	4	7
12	4	7	4	7
13	4	7	4	7
14	4	5	4	7
15	4	5	4	7

Număr de căi **S=4**

D	RD	BD	RI	BI
0	1	4	1	4
1	1	4	1	4
2	1	8	1	4
3	1	8	1	4
4	0	0	0	0
5	1	1	1	1
6	1	8	1	2
7	1	8	1	2
8	1	4	1	8
9	1	4	1	8
10	1	8	1	8
11	1	8	1	8
12	1	2	1	8
13	1	2	1	8
14	1	8	1	8
15	1	8	1	8

Număr minim de comutatoare **S=4**

D	RD	BD	RI	BI
0	4	5	4	5
1	4	5	4	5
2	4	7	4	5
3	4	7	4	5
4	0	0	0	0
5	4	1	4	1
6	4	7	4	3
7	4	7	4	3
8	4	5	4	7
9	4	5	4	7
10	4	7	4	7
11	4	7	4	7
12	4	3	4	7
13	4	3	4	7
14	4	7	4	7
15	4	7	4	7

Număr de căi **S=5**

D	RD	BD	RI	BI
0	1	4	1	4
1	1	4	1	4
2	1	8	1	4
3	1	8	1	4
4	1	1	1	1
5	0	0	0	0
6	1	8	1	2
7	1	8	1	2
8	1	4	1	8
9	1	4	1	8
10	1	8	1	8
11	1	8	1	8
12	1	2	1	8
13	1	2	1	8
14	1	8	1	8
15	1	8	1	8

Număr minim de comutatoare **S=5**

D	RD	BD	RI	BI
0	4	5	4	5
1	4	5	4	5
2	4	7	4	5
3	4	7	4	5
4	4	1	4	1
5	0	0	0	0
6	4	7	4	3
7	4	7	4	3
8	4	5	4	7
9	4	5	4	7
10	4	7	4	7
11	4	7	4	7
12	4	3	4	7
13	4	3	4	7
14	4	7	4	7
15	4	7	4	7

Număr de căi **S=6**

D	RD	BD	RI	BI
0	1	8	1	4
1	1	8	1	4
2	1	4	1	4
3	1	4	1	4
4	1	8	1	2
5	1	8	1	2
6	0	0	0	0
7	1	1	1	1
8	1	8	1	8
9	1	8	1	8
10	1	4	1	8
11	1	4	1	8
12	1	8	1	8
13	1	8	1	8
14	1	2	1	8
15	1	2	1	8

Număr minim de comutatoare **S=6**

D	RD	BD	RI	BI
0	4	7	4	5
1	4	7	4	5
2	4	5	4	5
3	4	5	4	5
4	4	7	4	3
5	4	7	4	3
6	0	0	0	0
7	4	1	4	1
8	4	7	4	7
9	4	7	4	7
10	4	5	4	7
11	4	5	4	7
12	4	7	4	7
13	4	7	4	7
14	4	3	4	7
15	4	3	4	7

Număr de căi **S=7**

D	RD	BD	RI	BI
0	1	8	1	4
1	1	8	1	4
2	1	4	1	4
3	1	4	1	4
4	1	8	1	2
5	1	8	1	2
6	1	1	1	1
7	0	0	0	0
8	1	8	1	8
9	1	8	1	8
10	1	4	1	8
11	1	4	1	8
12	1	8	1	8
13	1	8	1	8
14	1	2	1	8
15	1	2	1	8

Număr minim de comutatoare **S=7**

D	RD	BD	RI	BI
0	4	7	4	5
1	4	7	4	5
2	4	5	4	5
3	4	5	4	5
4	4	7	4	3
5	4	7	4	3
6	4	1	4	1
7	0	0	0	0
8	4	7	4	7
9	4	7	4	7
10	4	5	4	7
11	4	5	4	7
12	4	7	4	7
13	4	7	4	7
14	4	3	4	7
15	4	3	4	7

Număr de căi **S=8**

D	RD	BD	RI	BI
0	1	2	1	8
1	1	2	1	8
2	1	8	1	8
3	1	8	1	8
4	1	4	1	8
5	1	4	1	8
6	1	8	1	8
7	1	8	1	8
8	0	0	0	0
9	1	1	1	1
10	1	8	1	2
11	1	8	1	2
12	1	4	1	4
13	1	4	1	4
14	1	8	1	4
15	1	8	1	4

Număr minim de comutatoare **S=8**

D	RD	BD	RI	BI
0	4	3	4	7
1	4	3	4	7
2	4	7	4	7
3	4	7	4	7
4	4	5	4	7
5	4	5	4	7
6	4	7	4	7
7	4	7	4	7
8	0	0	0	0
9	4	1	4	1
10	4	7	4	3
11	4	7	4	3
12	4	5	4	5
13	4	5	4	5
14	4	7	4	5
15	4	7	4	5

Număr de căi **S=9**

D	RD	BD	RI	BI
0	1	2	1	8
1	1	2	1	8
2	1	8	1	8
3	1	8	1	8
4	1	4	1	8
5	1	4	1	8
6	1	8	1	8
7	1	8	1	8
8	1	1	1	1
9	0	0	0	0
10	1	8	1	2
11	1	8	1	2
12	1	4	1	4
13	1	4	1	4
14	1	8	1	4
15	1	8	1	4

Număr minim de comutatoare **S=9**

D	RD	BD	RI	BI
0	4	3	4	7
1	4	3	4	7
2	4	7	4	7
3	4	7	4	7
4	4	5	4	7
5	4	5	4	7
6	4	7	4	7
7	4	7	4	7
8	4	1	4	1
9	0	0	0	0
10	4	7	4	3
11	4	7	4	3
12	4	5	4	5
13	4	5	4	5
14	4	7	4	5
15	4	7	4	5

Număr de căi **S=10**

D	RD	BD	RI	BI
0	1	8	1	8
1	1	8	1	8
2	1	2	1	8
3	1	2	1	8
4	1	8	1	8
5	1	8	1	8
6	1	4	1	8
7	1	4	1	8
8	1	8	1	2
9	1	8	1	2
10	0	0	0	0
11	1	1	1	1
12	1	8	1	4
13	1	8	1	4
14	1	4	1	4
15	1	4	1	4

Număr minim de comutatoare **S=10**

D	RD	BD	RI	BI
0	4	7	4	7
1	4	7	4	7
2	4	3	4	7
3	4	3	4	7
4	4	7	4	7
5	4	7	4	7
6	4	5	4	7
7	4	5	4	7
8	4	7	4	3
9	4	7	4	3
10	0	0	0	0
11	4	1	4	1
12	4	7	4	5
13	4	7	4	5
14	4	5	4	5
15	4	5	4	5

Număr de căi **S=11**

D	RD	BD	RI	BI
0	1	8	1	8
1	1	8	1	8
2	1	2	1	8
3	1	2	1	8
4	1	8	1	8
5	1	8	1	8
6	1	4	1	8
7	1	4	1	8
8	1	8	1	2
9	1	8	1	2
10	1	1	1	1
11	0	0	0	0
12	1	8	1	4
13	1	8	1	4
14	1	4	1	4
15	1	4	1	4

Număr minim de comutatoare **S=11**

D	RD	BD	RI	BI
0	4	7	4	7
1	4	7	4	7
2	4	3	4	7
3	4	3	4	7
4	4	7	4	7
5	4	7	4	7
6	4	5	4	7
7	4	5	4	7
8	4	7	4	3
9	4	7	4	3
10	4	1	4	1
11	0	0	0	0
12	4	7	4	5
13	4	7	4	5
14	4	5	4	5
15	4	5	4	5

Număr de căi **S=12**

D	RD	BD	RI	BI
0	1	4	1	8
1	1	4	1	8
2	1	8	1	8
3	1	8	1	8
4	1	2	1	8
5	1	2	1	8
6	1	8	1	8
7	1	8	1	8
8	1	4	1	4
9	1	4	1	4
10	1	8	1	4
11	1	8	1	4
12	0	0	0	0
13	1	1	1	1
14	1	8	1	2
15	1	8	1	2

Număr minim de comutatoare **S=12**

D	RD	BD	RI	BI
0	4	5	4	7
1	4	5	4	7
2	4	7	4	7
3	4	7	4	7
4	4	3	4	7
5	4	3	4	7
6	4	7	4	7
7	4	7	4	7
8	4	5	4	5
9	4	5	4	5
10	4	7	4	5
11	4	7	4	5
12	0	0	0	0
13	4	1	4	1
14	4	7	4	3
15	4	7	4	3

Număr de căi **S=13**

D	RD	BD	RI	BI
0	1	4	1	8
1	1	4	1	8
2	1	8	1	8
3	1	8	1	8
4	1	2	1	8
5	1	2	1	8
6	1	8	1	8
7	1	8	1	8
8	1	4	1	4
9	1	4	1	4
10	1	8	1	4
11	1	8	1	4
12	1	1	1	1
13	0	0	0	0
14	1	8	1	2
15	1	8	1	2

Număr minim de comutatoare **S=13**

D	RD	BD	RI	BI
0	4	5	4	7
1	4	5	4	7
2	4	7	4	7
3	4	7	4	7
4	4	3	4	7
5	4	3	4	7
6	4	7	4	7
7	4	7	4	7
8	4	5	4	5
9	4	5	4	5
10	4	7	4	5
11	4	7	4	5
12	4	1	4	1
13	0	0	0	0
14	4	7	4	3
15	4	7	4	3

Număr de căi **S=14**

D	RD	BD	RI	BI
0	1	8	1	8
1	1	8	1	8
2	1	4	1	8
3	1	4	1	8
4	1	8	1	8
5	1	8	1	8
6	1	2	1	8
7	1	2	1	8
8	1	8	1	4
9	1	8	1	4
10	1	4	1	4
11	1	4	1	4
12	1	8	1	2
13	1	8	1	2
14	0	0	0	0
15	1	1	1	1

Număr minim de comutatoare **S=14**

D	RD	BD	RI	BI
0	4	7	4	7
1	4	7	4	7
2	4	5	4	7
3	4	5	4	7
4	4	7	4	7
5	4	7	4	7
6	4	3	4	7
7	4	3	4	7
8	4	7	4	5
9	4	7	4	5
10	4	5	4	5
11	4	5	4	5
12	4	7	4	3
13	4	7	4	3
14	0	0	0	0
15	4	1	4	1

Număr de căi **S=15**

D	RD	BD	RI	BI
0	1	8	1	8
1	1	8	1	8
2	1	4	1	8
3	1	4	1	8
4	1	8	1	8
5	1	8	1	8
6	1	2	1	8
7	1	2	1	8
8	1	8	1	4
9	1	8	1	4
10	1	4	1	4
11	1	4	1	4
12	1	8	1	2
13	1	8	1	2
14	1	1	1	1
15	0	0	0	0

Număr minim de comutatoare **S=15**

D	RD	BD	RI	BI
0	4	7	4	7
1	4	7	4	7
2	4	5	4	7
3	4	5	4	7
4	4	7	4	7
5	4	7	4	7
6	4	3	4	7
7	4	3	4	7
8	4	7	4	5
9	4	7	4	5
10	4	5	4	5
11	4	5	4	5
12	4	7	4	3
13	4	7	4	3
14	4	1	4	1
15	0	0	0	0

ANEXA 3

Programul de determinare a căii optime între o sursă şi o destinaţie date în RMM.

```
#define LMAX 255
#include <stdio.h>
#include <math.h>
#include <stdlib.h>

char S[LMAX], D[LMAX], EC[LMAX], ECR[LMAX];

short int NID, NII, l;

short int cai_directe = 1; //se selecteaza căi directe (1) sau cai
inverse (0)

void verifica (char * S) {

        int i;
        for(i=0; i<l; i++)
                if (S[i] != '0' && S[i] != '1') {
                        printf("Eroare.\n");
                        exit(2);
                        }
        }

void introducere (void) {

        printf("\n\n\nIntroduceti lungimea (l) [max %d]: ",LMAX);
        scanf("%d",&l);
        if (l>255) {
                printf("Eroare. l > %d\n",LMAX);
                exit(1);
                }
        gets(S);
        printf("Introduceti S: ");
        gets(S);
        verifica(S);
        printf("Introduceti D: ");
        gets(D);
        verifica(D);
        }

int egalitate (char * S, char * D) {

        int i;
        for (i=0; i<l; i++)
                if (S[i] != D[i])
                        return 0;
        return 1;

        }

char xor (char s, char d) {
        if (s == d)
                return '0';
```

```
        else
                return '1';

        }

void calc_EC (void) {

        int i;
        for (i=0; i<l; i++)
            EC[i] = xor (S[i], D[i]);

        }

void calc_ECR (void) {

        int tmp=EC[l-1];
        int i;
        for (i=l; i<l; i++)
                ECR[i] = EC[i-1];
        ECR[0] = tmp;

        }

void calc_NID (void) {

        int i = l-1;
        while (i >= 0 && (ECR[i] == '0'))
                i--;
        NID=i;

        }

void calc_NII (void) {

        int i = 0;
        while (i < l && (EC[i] == '0'))
                i++;
        NII=i;

        }

void rotire_st (char * S) {

        char tmp = S[0];
        int i;
        for (i=0; i<l-1;i++)
                S[i] = S[i+1];
        S[l-1] = tmp;

        }

void rotire_dr (char * S) {

        char tmp = S[l-1];
        int i;
        for (i=l-1; i>0; i--)
                S[i] = S[i-1];
```

```
        S[0] = tmp;

        }

void cale_RD (char * S, char * D) {

        int j = 0;
        while (j < l) {
                printf("Nivel %d st: ",j);
                puts(S);
                S[l-1] = D[j];
                printf("Nivel %d dr: ",j);
                puts(S);
                rotire_st (S);
                j++;
                }
        }

void cale_RI (char * S, char * D) {

        int j = l-1;
        while (j >= 0) {
                printf("Nivel %d dr: ",j);
                puts(S);
                if (j == 0)
                        S[l-1] = D[l-1];
                else
                        S[l-1] = D[j-1];
                printf("Nivel %d st: ",j);
                puts(S);
                rotire_dr (S);
                j--;
                }
        }


void cale_BD (char * S, char * D) {

        int j = 0;
        while (j <= NID) {
                printf("Nivel %d st: ",j);
                puts(S);
                if (j == NID) {
                        if (j==0)
                                S[l-1] = D[l-1];
                        else
                                S[l-1] = D[NID-1];
                        printf("Nivel %d st: ", j);
                        puts(S);
                        rotire_dr (S);
                        j--;
                        break;
                        }
                S[l-1] = D[j];
                printf("Nivel %d dr: ",j);
                puts(S);
                rotire_st (S);
```

```
                j++;
                }

        while (j >= 0) {
                printf("Nivel %d dr: ",j);
                puts(S);
                if (j == 0)
                        S[1-1] = D[1-1];
                else
                        S[1-1] = D[j-1];
                printf("Nivel %d st: ",j);
                puts(S);
                rotire_dr (S);
                j--;
                }
        }


void cale_BI (char * S, char * D) {

        int j = l-1;
        while (j >= NII) {
                printf("Nivel %d dr: ",j);
                puts(S);
                if (j == NII) {
                        S[1-1] = D[NII];
                        printf("Nivel %d dr: ",j);
                        puts(S);
                        rotire_st (S);
                        j++;
                        break;
                        }
                if (j == 0)
                        S[1-1] = D[1-1];
                else
                        S[1-1] = D[j-1];
                printf("Nivel %d st: ",j);
                puts(S);
                rotire_dr (S);
                j--;
                }

        while (j < l) {
                printf("Nivel %d st: ",j);
                puts(S);
                S[1-1] = D[j];
                printf("Nivel %d dr: ",j);
                puts(S);
                rotire_st (S);
                j++;
                }
        }

void main (void) {

        short int dl,du;
```

```
  introducere();

  calc_EC();
  calc_ECR();
  dl = floor ((double)l/2.);
  du = ceil ((double)l/2.);

  printf("S: "); puts(S);
  printf("D: "); puts(D);
  printf("EC: "); puts(EC);
  printf("ECR: "); puts(ECR);

  if (egalitate (S, D)) {
          printf("Cerere de acces pentru memoria locala\n");
          exit(0);
          }
  else {
          calc_NID();
          calc_NII();
          }

  printf("NID=%d NII=%d\n", NID,NII);

if ((NID == (l-1-NII))==0) {

          if (cai_directe) {
                  printf ("Directionare BD:\n");
                  cale_BD (S, D);
                  }
          else {
                  printf ("Directionare BI:\n");
                  cale_BI (S, D);
                  }
          }
  else if (NID < dl) {
          printf ("Directionare BD:\n");
          cale_BD (S, D);
          }
  else if (NID >= du) {
          printf ("Directionare BI:\n");
          cale_BI (S, D);
          }
  else {
          if (cai_directe) {
                  printf ("Directionare RD:\n");
                  cale_RD (S, D);
                  }
          else {
                  printf ("Directionare RI:\n");
                  cale_RI (S, D);
                  }
          }
```

BIBLIOGRAFIE

[AAS87] G.B. Adams, D.P. Agrawal and H.J. Siegel, "A Survey and Comparison of Fault-Tolerant Multistage Interconnection Networks", *Computer*, vol.20 (June 1987), pp.14 - 27

[ADT91] M.S.Algudady, C.R.Das and M.J. Thazhuthaveetil, "A Write Update Cache Coherence Protocol for MIN -Based Multiprocessors with Accessibility -Based Split Caches", *IEEE Trans On Computers*, pp.544-552, July 1990

[AG89] G.S. Almasi and A. Gottlieb, *Highly Parallel Computing*, Redwood City, CA, Benjamin/Cummings, 1989

[Agra88] P. Agrawal, "Fault Tolerance in Multiprocessor Systems Without Dedicated Redundancy", *IEEE Transactions on Computers*, vol.37 (3), pp. 358 – 362, 1988

[AFM90] R. Ahmed, R. Frazier and P. Marinos, "CARER: Cache-Aided Rollback Error Recovery Algorithm for Shared Memory Multiprocessors", *Proc. 20th Int. Symp. Fault-Tolerant Computing*, pp.82 – 88, June 1990

[Arch88] J. K. Archibald, "A Cache Coherence Approach for Large Multiprocessor Systems", *ACM International Conference on Supercomputing*, pp.337 – 345, 1988

[Ban90] P. Banerjee, "Strategies for Reconfiguring Hypercubes under Faults", *Proc. 20th Int. Conf. Fault-Tolerant Computing Systems*, pp.210 – 215, June 1990

[BD91] E. Brewer, C. Dellarocas, "PROTEUS: A High Performance Parallel Architecture Simulator", TR516 MIT/LCS, Mass. Inst. of Technology, Sept.1991

[BIK00] L. Bhuyan, R.Iyer and A. Kumar, "Impact of CC-NUMA Memory Management Policies on the Application Performance of Multistage

Switching Networks ", *IEEE Trans on Parallel and Distributed Systems*, 13(3), pp.230-246, March 2000

[BINK97] L.Bhuyan, R. Iyer, A.Nanda and M. Kumar, "Performance of Multistage Bus Networks for a Distributed Shared Memory Multiprocessor ", *IEEE Trans. On Parallel and Distributed Systems*, 8(1),pp. 82-95, January 1997

[BLA89] L.Bhuyan, B.Liu, and L.Ahmed "Analysis of MIN Based Multiprocessors with Private Cache Memories ", *Proc 1989 International Conf. On Parallel Processing* ", vol I. pp. 51-58, August 1989

[BN91] L.Bhuyan and A. Nanda,"Multistage Bus Network (MBN) An Interconnection Network for Cache Coherent Multiprocessors ",*Proc Third IEEE Symp. Parallel and Distributed Processing*, Dec1991.

[BNA94] L.Bhuyan,A.Nanda and T.Askar,"Performance and Reliability of the MBN",*Proc.1994 Int'l Conf. Parallel Processing*,pp. 26-33

[BNT94] L. Bhuyan, A. Nanda and T. Askar, "Performance and Realiability of MBN", *Proc. 1994 Int'l Conf. Parallel Processing*, pp. 26-33

[BP92] N. Bowen and D. Pradhan, "Virtual Checkpoints: Arhitecture and performance", *IEEE Transactions on computers*, vol.41, pp. 516 – 525, May 1992

[BP93] N. Bowen and D. Pradhan, "Processor and Memory Based Checkpoint and Rollback Recovery", *Computer*, pp. 22 – 31, Feb. 1993

[BRA89] C. Botting, S. Rai and D. Agrawal, "Reliability Computation of MINs", *IEEE Trans. on Reliability*, vol.38 (1), April 1989

[BRL94] A. Bhattacharya, R. Rao and T. Lin, "Cumulative Performance Measure for Gracefully Degradable MINs", *Proc. 1st International Workshop on Parallel Processing*, pp. 234 – 239, December 1994

[BT88] I.Blake and K.Trivedi, "Reliabilities of Two Fault - Tolerant INs ",Proc.18th Intern. Symp. Fault - Tolerant Computing, pp.300-305, June 1988

[BT89] I.Blake and K.Trivedi,"Reliability Analysis of INs Using Hierarchical Composition ", *IEEE Trans. On Reliability*, 38(1), pp.111-119, April 1989

[BT93] N. Benitez and K. Trivedi, "Multiprocessor Performability Analysis ", *IEEE Trans. On Reliability*, 42(4), pp. 579-587, Dec 1993

[BWK98] L. Bhuyan, H.Wang and A.Kumar, "Impact of Switch Design on the Application Performance of Cache-Coherent Multiprocessors ", *International Parallel Processing Symposium*, pp.466-474, March 1998

[BYA89] L.N. Bhuyan, Qing Yang, D.P. Agrawal, "Performance of Multiprocessor Interconnection Networks", *Computer*, vol.22 (February 1989), pp.25 – 37

[CBZ95] J. Carter, J. Bennett and W. Zwaenepoel, "Techiques for Reducing Consistency-Related Information in Distributed Shared Memory Systems", *ACM Transaction on Computer Systems*, vol.13 (3), pp. 205 – 243, Aug. 1995

[CF78] L.M.Censier and P.Feautrier, "A new solution to coherence problems în multicache systems", *IEEE Transactions on Computers*, C-27 (December 1978), pp.1112 – 1118

[CFKA90] D. Chaiken, C. Fields, K.Kurihara and A. Agarwal, "Directory-Based Cache Coherence in Large Scale Multiprocessors", *IEEE Computer*, vol.23 (6), pp.49 – 58, June 1990

[Choi96] L. Choi, *Compiler and Hardware Support for Cache Coherence in Large Scale Multiprocessors*, PhD Dissertation, University of Illinois at Urbana, Aug. 1996

[CK92] F. Chong and T. Knight, jr., "Design and Performance of Multipath MIN Architectures", *Proc. 4th Annual ACM Symposium on Parallel Algorithms and Architectures*, pp.286 – 295, 1992

[Coif94] V. Coifan, "Stadiul actual al sistemelor de calcul multiprocesor", referat la teza de doctorat, Universitatea Politehnica Timişoara, Fac. de Automatică şi Calculatoare, Timişoara 1994

[Coif01] V. Coifan, "Optimizarea performanțelor la sistemele multiprocesor în funcție de dependabilitate", *va apare în Editura Orizonturi Universitare*

[CSG99] D. Culler, J. Singh and A. Gupta, *Parallel Computer Architecture: A Hardware/Software Approach*, San Francisco, Morgan Kaufmann Publishers, 1999

[CWH01] V. Coifan, R. Wagmann and N. Hanigovszki, "A Routing Algorithm for Multistage Bus Networks for a Distributed Shared Memory Multiprocesor", *Buletinul Științific al Universității Politehnica*, Timişoara, 2001

[DA93] T.Dahlberg and D. Agrawal, "Dependability Analysis for Large Systems : a Hierarchical Modeling Aproach ", *IEEE Trans on Reliability*, pp.450-459

[Dal92] W. J. Dally, "Vitual-Channel Flow Control", *IEEE Transactions on Parallel and Distributed Systems*, vol. 3 (2), pp.194 – 205, Mar. 1992

[Dasg89] S. Dasgupta, "*Computer Arhitecture - A Modern Synthesis*", vol.2, John Wiley and sons, 1989

[DB94] J. Ding and L. Bhuyan, "Finite Buffers Analysis of MINs", *IEEE Transactions on Computers*, vol. 43(2), pp.243 – 247, Feb. 1994

[DKT90] C.R.Das, J.T.Kreulen and M.J.Thazhuthaveetil, "Dependability Modeling for Multiprocessors ",*Computer*, October 1990, pp. 8-19

[DSB88] M.Dubois, C.Scheurich and F.Briggs, "Synchronization Coherence and Event Ordering in Multiprocessors", *Computer*, February 1988, pp.9 – 21

[DTB90] C.R. Das,L. Tien and L.N. Bhuyan, "Availability Evaluation of MIN - Connected Multiprocessors using Decomposition Technique ",

[DTB93] C.R.Das, L.Tien and L.N. Bhuyan, "An Availability Model for MIN - Based Multiprocessors", *IEEE Transactions on Parallel and Distributed Systems*, vol.4, October 1993, pp. 1118-1129

[FP90] E. Fujiwara and D. Pradhan, "Error Control Coding in Computers", *Computer*, pp. 63 – 72, July 1990

[GGH91] K.Gharachorloo, A.Gupta and J. Hennessy, "Performance Evaluation of Memory Consistency Models for Shared Memory Multiprocessors", *Int. Conf. on Architectural Support for Programming Languages and Operating Systems*, pp. 245 – 257, 1991

[GHGM91] A.Gupta, J.Hennessy, K.Gharacholoo and T.Mowry, "Comparative Evaluation of Latency Reducing and Tolerating Techniques", *Int. Symp. on Computer Architecture*, pp. 254 – 263, 1991

[GLL+90] K. Gharachorloo, D. Lenoski, J. Laudon, P. Gibbons, A. Gupta et al., "Memory Consistency and Event Ordering in Scalable Shared Memory Multiprocessors", *Proceedings of the Internationa Symposium on Computer Architecture*, pp.15 – 26, 1990

[Good83] J. R. Goodman, "Using Cache Memory to Reduce Processor-Memory Traffic", *International Symposium on Computer Architecture*, pp. 124 – 131, 1983

[Gros99] D. Grosu, "*Rollback Recovery Techniques in Message Passing Systems : A Survey and Synthesis* " Technical Report, Dept. of Computer Engineering, Technical University of Timisoara, June 1999

[GVW89] J. Goodman, M. Vernon and P. Woest, "A Set of Efficient Syncronisation Primiteves for a Large Scale Shared Memory Multiprocessors". *Int. Conf. on Architectural Support for Programming Languages and Operating Systems*, pp.64 – 73, 1989

[GW88] J. Goodman and P. Woest, "The Wisconsin Multicube: A New Large Scale Cache Coherent Multiprocessor", *International Symposium on Computer Architecture*, pp. 422 – 431, 1988

[GY93] P. Gawghan and S. Yalamanchi, "Adaptive Routing Protocols for Hypercube Interconnection Networks", *IEE Transactions on Computers*, vol. 26 (5), pp. 12 – 23, May 1993

[HM87] D. Hunt and P. Marinos, "A General Purpose Cache-Aided Rollback Error Recovery (CARER) Tehnique", *17th Symp. On Fault Tolerant Computing*, pp. 170 – 175, June 1987

[HP94] J.L. Hennessy and D.A. Patterson, "*Computer Organization and Design*", Morgan Kaufmann, San Francisco, CA, 1994

[HP96] J.L. Hennessy and D.A. Patterson, "*Computer Organization - A Quantitative Aproach*", Morgan Kaufmann, San Francisco, CA, 1996

[Hwan93] K. Hwang, "*Advanced Computer Architecture : Parallelism, Scalability, Programmability*", Mc Graw-Hill Inc., 1993

[IB99] R. Iyer and L. Bhuyan, "Switch Cache: A Framework for Improving the Remote Memory Access Latency of CC-NUMA Multiprocessors", *Fifth International Conference on High Performance Computer Arhitecture (HPCA –5)*, pp.152 – 160, Jan. 1999

[IB00] R. Iyer and L.Bhuyan, "Design and Evaluation of a Switch Cache Architecture for CC-NUMA Multiprocessors, *IEEE Trans . on Computers*, 49(8), pp. 779-797, August 2000

[Iyer96] R. Iyer, "Distributed Shared Memory Multiprocessors Using Multistage Bus Networks", MS Thesis, Texas A&M Univ. Dept. of Comp. Science, Aug. 1996

[Iyer99] R.Iyer, "High Performance Switch Architectures for CC-NUMA Multiprocessors", PhD Dissertation, Texas A&M.Univ. Dept. of Comp. Science, Aug. 1999

[JF91] B. Janssens and W. Fuchs, "Experimental Evaluation of Multiprocessor Cache-Based Error Recovery", *1991 International Conference on Parallel Processing*, pp. 505 – 508, Aug. 1991

[Kak96] S.H. Kakas, "*Contribuţii la analiza şi sinteza de fiabilitate a sistemelor multiprocesor*", teză de doctorat, Universitatea Politehnica din Timişoara, Fac. Automatică şi Calculatoare, Timişoara, 1996

[KB96] A. Kumar and L. Bhuyan, "Evaluationg Virtual Channels for Cache Coherent Shared Memory Multiprocessors", *ACM International Conference on Supercomputing*, pp. 253 – 260, May 1996

[KBI96] A. Kumar, L. Bhuyan and R. Iyer, "Execution Based Evaluation of MINs for Cache Coherent Multiprocessor", *International Conference on Advanced Computing (ADCOMP `96)*, India, May 1996

[KR87] V.P. Kumar and S.M. Reddy, "Augmented Shuffle - Exchange Multistage Interconnection Networks", *Computer*, vol.20 (June 1987), pp.30 – 40

[KS83] C. Kruskal and M. Snir, "The Performance of MINs for Multiprocessors", *IEEE Trans. on Computers*, vol.32 (12), pp.1091-1098, Dec.1983

[Kum96] A. Kumar, *Execution-Driven Evaluation of Cache Coherent Shared Memory Multiprocessors*, PhD Dissertation, Dept. of Computer Science, Texas A&M University, Aug. 1996

[Lamp79] L. Lamport, "How to Make a Multiprocessor Computer that Corectly Executes Multiprocess Programs", *IIE Transactions on Computers*, vol. C-28 (9), pp. 690 – 691, September 1979

[LFA90] J. Long, W. Fuchs and J. Abraham, "A Forward Recovery Using Checkpointing in Parallel Systems", *Proceedings of the 1990 International Conference on Parallel Processing*, pp.272 – 275, Aug. 1990

[LFA91] J. Long, W. Fuchs and J. Abraham, "Implementing Forward Recovery Using Checkpoints in Distributed Systems", *IFIP 2nd Int. Working Conf. Dependable Computing for Critical Applications*, Feb.1991

[Lil93] D. J. Lilja, "Cache Coherence in Large Scale Shared Memory Multiprocessors", *ACM Computing Surveys*, vol. 25 (3), pp. 303 – 338, Sept. 1993

[LSP82] L.Lamport, R.Shoshtak and M.Pease, "The Byzantine General`s Problem", *ACM Trans. Prog. Lang. Syst.*, July 1982, pp.382 – 401

[Marq89] D. Marquardt, "C2MP: A Cache Coeherent Distributed Memory Multiprocessor System", *Proceedings Supercomputing '89,* pp.466 – 475, 1989

[MBL89] H.E.Mizrahi, J.L.Baer, E.D. Lazowska, "*Introducing Memory into the Switch Elements of Multiprocessor Interconnection Networks* ", University of Washington, Dept. of Comp. Sci, Technical Report 88-11-03, Nov 1988

[MD96] P.Mohapatra and C.Das, "PerformanceAnalysis of Finite - Buffered Asyncronous MINs",*IEEE Trans. On Parallel and Distributed Systems*, 7(1), pp.18-25, January 1996.

[MD98] A. Moga and M. Dubois, "The Effectiveness of SRAM Network Caches on Clustered DSMs", *Fourth International Symposium on High Performance Computer Arhitecture*, pp. 103 – 112, Feb. 1998

[MHB86] T.N. Mudge, I.P. Hayes, G.D. Buzzard, "Analysis of Multiple-Bus Interconnection Netwoks", *Journal of Parallel and Distributed Computing*, (1986), pp.328 – 343

[MHW87] T.N. Mudge, J.P. Hayes, D.C. Winsor, "Multiple-Bus Architectures", *Computer*, vol. 20 (June 1987), pp.42 – 48

[MN99] M. Michael and A. Nanda, "Design and Performance of Directory Caches for Scalable Shared Memory Multiprocessors", *Fifth International Conference on High Performance Computer Arhitecture (HPCA – 5)*, pp.142 – 151, 1999

[MNLS97] M. Michael, A. Nanda, B. Lim and M. Scott, "Coherence Controller Arhitectures for Shared Memory Multiprocessors", *Proceedings of the International Symposium on Computer Arhitecture (ISCA '97)*, pp. 219 – 228, 1997

[Mold93] D.I. Moldovan, "*Parallel Processing : From Applictions to Systems*", Morgan Kaufman, 1993

[MP89] J. Meyer and D. Pradhan, "Dynamic Testing Strategy for Distributed Systems", *IEEE Transactions on Computers*, vol 38(3), pp. 356 – 365, March 1989

[NB93] A.K.Nanda and L.N.Bhuyan, "Design and Analysis of Cache Coherent Multistage Interconnection Networks ",*IEEE Trans. On Computers*, vol.42,no.4, April 1993

[Ng91] T. P. Ng, *Checkpointing Algorithms for Shared Memory Systems*, Technical Report, Dept. of Computer Science, University of Illinois, Urbana, 1991

[NK93] L. Ni and P. McKinley, "A Survey of Wormhole Routing Tehniques în Direct Netoworks", *Computer*, Vol. 26 (2), pp. 62 – 76, Feb. 1993

[NOS96] B. Nayfeh, C. Olukotun and J. Singh, "The Impact of Shared-Chache Clustering in Small-Scale Shared memory Multiprocessors", *Proceedings of the 2nd International Symposium on High Performance Computer Arhitecture*, pp.74 – 84, Feb. 1996

[PA96] R. Potlapalli and D. Agrawal, "On the Design of Efficient MINs", *IEEE Proc. of MASCOTS '96*

[Pat81] J. Patel, "Performance of Processor-Memory Interconnection for Multiprocessors", *IEEE Transactions on Computers*, vol. C-30 (10), pp. 771 – 780, Oct. 1991

[PF90] D. Pradhan and E. Fujiwara, "Error Computing Codes în Fault Tolerant Computers", *Computer*, vol.23 (7), pp. 62 – 72, July 1990

[Prad89] D. K. Pradhan, *Redundancy Schemes for Recovery*, TR-89-CSE-16, ECE Department, University of Massachusets, 1989

[Prad96] D.K. Pradhan, "*Fault-tolerant Computer System Design*", Prentice Hall PTR, New Jersey, 1996

[Popa96] M.Popa, "Intruducere în arhitecturi paralele şi neconvenţionale", *Editura AS Computer Press*, Timişoara 1992

[PV94a] D. Pradhan and N. Vaydia, "Roll Forward Checkpointing Scheme: A Novel Fault-Tolerant Arhitecture", *IEEE Transactions on Computers*, vol. 43 (10), pp. 1163 – 1174, October 1994

[PV94] D. Pradhan and N. Vaydia, "Performance and Reliability Trade-offs in Roll Forward Schemes", *Proc. FTCS-24*, Austin Texas, June 1994

[RA90] S. Rai and D. Agrawal, *Distributed Computing Network Reliability*, Tutorial Textbook, IEEE CS Press, 1990

[RG87] D.A. Reed and D.C. Grunwald, "The Performance of Multicomputer Interconnection Networks, *Computer*, vol. 20 (June 1987), pp.63 – 73

[SD89] C. Scheurich and M. Dubois, "Dynamic Page Migration în Multiprocessors with Distributed Global Memory", *IEEE Transactions on Computers*, vol. 38 (8), pp. 1154 – 1163, Aug. 1989

[Sita93] R. Sitamaran, "Communication and Fault Tolerance în Parallel Computers", PhD Dissertation CS-TR-414-93, Dept. of Comp.Science, Princeton Univ., Princeton NJ, 1993

[SLM89] Sang Lyul Min, "*Memory Hierarchy Management Schemes în Large Scale Shared - memory Multiprocessor*", Technical Report # 89-08-07, Dept. of Computer Science and Engineering, FR-35, University of Washington, Seattle, WA 98195 USA, August 1989

[Smit82] A. J. Smith, "Cache Memories", *ACM Computing Surveys*, vol. 14 (3), pp. 473 – 530, 1982

[TD89] L. Tien and C. Das, "Reliability Evaluation of Butterfly Network-Based Multiprocessors", TR – 89 – 072, The Pennsylvania State University

[TH90] J.Torrellas and J.Hennessy, "Estimating the Performance Advantages of Relaxing Consistency in a Shared Memory Multiprocessor", *Int. Conf. on Parallel Processing*, vol. I, pp.26 – 33, 1990

[TZ94] J. Torrellas and Z. Zhang, "The Performance of the Cedar Multistage Switching Network", *IEEE Transactions on Parallel and Distributed Systems*, vol. 8 (4), pp. 321 – 336, Apr. 1994

[Vaid86] A. V. Veidenbaum, "A Compiler-Assisted Cache Coherence Solution for Multiprocessors", *International Conference on Parallel Processing*, pp. 1029 – 1036, 1986

[VP93] N. Vaidya and D. Pradhan, "Fault Tolerant Design Strategies for High Reliability and Safety", *IEEE Transactions on Computers*, vol. 43 (10), Oct. 1993

[VSD97] A. Vaidya, A. Sivasubramaniam and C. Das, "Performance Benefits of Virtual Channels and Adaptive Routing: An Application-Driven Study", *Proceedings of the 11th International Conference on Supercomputing*, pp. 140 – 147, Jul. 1997

[WFP90] K.Wu,K.Fuchs and I.Patel, "Error Recovery in Shared Memory Multiprocessors Using Private Caches ", *IEEE Trans. On Parallel and Distributed Systems*, 1(2), pp.231-240, April 1990

[Wils87] A. W. Wilson, "Hierarhical Cache/Bus Arhitecture for Shared Memory Multiprocessors", *Proceedings of the 14th Annual International Symposium on Computer Arhitecture*, pp. 244 – 252, 1987

[Witt81] L.D. Wittie, "Communication Structures for Large Networks of Microcomputers", *IEEE Transactions on Computers*, vol. 30 NO. 4 apr. 1981, pp.246 – 273

[YB91] Q. Yang and L. Bhuyan, "Analysis of Packet - Switched Multiple-Bus Multiprocessors ", *IEEE Trans. on Computers*, 40(3) pp.352-357, March 1991

[YLL90] H. Yoon, K. Lee and M. Liu, "Performance Analysis of Multibuffered Packet-Swithing Networks in Multiprocessor Systems", *IEEE Transactions on Computers*, vol. 39 (3), pp. 319 – 327, Mar. 1990

[ZB92] R.Zucker and J.-L.Baer, "A Performance Study of Memory Consistency Models", *Int. Symp. on Computer Architecture*, pp.2 – 12, 1992

400

LISTĂ SELECTIVĂ

A LUCRĂRILOR ŞTIINŢIFICE PUBLICATE, A INVENŢIILOR ŞI INOVAŢIILOR ELABORATE ŞI A CERCETĂRILOR CONTRACTUALE EFECTUATE

A. Lucrări ştiinţifice publicate

[Coif75] V. Coifan, "Bazele logice ale calculatoarelor", *Îndrumător de lucrări*, IPTVT 1975

[RPC77] Al. Rogojan, V. Pop şi V. Coifan, "Microcalculator didactic MCD1", *Buletinul Ştiinţific şi Tehnic al IPTVT*, tom 22, iulie – decembrie 1977, pp.383 – 388

[Coif81] V. Coifan, "Microcalculator didactic MCD85", *Primul simpozion de microprocesoare, microcalculatoare şi aplicaţii*, noiembrie 1981, pp. 68 – 76

[Coif84] V. Coifan, "Sistem ierarhizat de microcalculatoare construit în jurul familiei de calculatoare M18", *Conferinţa naţională a cercurilor ştiinţifice studenţeşti*, Timişoara, noiembrie 1984

[CC84] V. Creţu şi V. Coifan, "Executiv în timp real implementat pe un sistem de calcul ECAROM 881", *Conferinţa naţională a cercurilor ştiinţifice studenţeşti*, Timişoara, noiembrie 1984

[Coif85] V. Coifan, "An arithmetic processing unit APU – for the ECAROM 881 microcomputer", *Simpozionul de microprocesoare, microcalculatoare şi aplicaţii în economie*, noiembrie 1985, pp.67 – 68

[CP85] V. Coifan şi O. Proştean, "A HD 64180 µP-based SBC", *Simpozionul de microprocesoare, microcalculatoare şi aplicaţii în economie*, noiembrie 1985, pp.67 – 68

[CPDL+85] V. Coifan, O. Proştean, T. Dragomir ş.a., "Some aspects of automatic voltage regulation at hidrogenerator using a process computer", *Hydro-Turbo*, 1985, Brno, Cehoslovacia

[BC85] N. Budişan şi V. Coifan, "Studiul posibilităţilor de teleinvestigare cu ajutorul calculatorului a agregatului aeroelectric de 300 KVA de pe muntele Semenic", *IPA*, mai 1985

[CMP86] V. Coifan, I. C. Moş şi M. Popa, "Structura sistemelor de prelucrare a datelor numerice", *Îndrumător de lucrări*, IPTVT 1986

[CP88] V. Coifan şi O. Proştean, "Calculator personal cu facilităţi de conducere a proceselor realizat în jurul lui μP HD 64180", *Simpozionul naţional de calculatoare şi conducere automată a proceselor*, Timişoara 1988, pp. 19 – 22

B. Rapoarte la contracte de cercetare

[CB87] V. Coifan şi R. Boraci, "Cercetări privind realizarea unui regulator de tensiune destinat hidrogeneratoarelor reversibile", *Contract de cercetare* nr. 126/1987 cu CCSITEH Reşiţa

[CB88] V. Coifan şi R. Boraci, "Experimentarea şi analiza regulatorului de tensiune într-o centrală hidroelectrică", *Contract de cercetare* nr. 162/1988 cu CCSITEH Reşiţa

[CP84] V. Coifan şi I.Păpuşoi , "Comanda cu microcalculatorul ECAROM 881 a regimului de propulsie la vehicolul MAGNIBUS 01", *Contract de cercetare* nr. 128/1984 cu CCSIT Electroputere Craiova

[Coif87] V. Coifan. "Posibilităţi de utilizare într-un mediu multiprocesor a modulelor din familia MULTIPROM", *Contract de cercetare* nr. 7405/1987 cu IPA Bucureşti

C. **Invenţii şi inovaţii**

- *Procedeu de extindere a posibilităţilor de reglare automată bipoziţională a echipamentelor de calcul numerice* – brevet de invenţie
- *Metodă şi dispozitiv electronic pentru testarea poziţiei rotorului şi conducerea unei maşini sincrone cu ajutorul microprocesorului* – brevet de invenţie
- *Metodă de testare a întreruptoarelor din microcalculatorul ECAROM* 881 – certificat de inovator, nr. 133/1985
- *Schemă logică pentru protecţia memoriei partajate* – certificat de inovator nr. 200/1985

- *Modul de memorie pentru sisteme de calcul multiprocesor* – certificat de inovator nr. 489/1986
- *Comenzi de monitor specifice arhitecturilor de calcul multiprocesor* – certificat de inovator nr. 419/1986
- *Panou de testare a modulelor calculator monoplacă pe* 8 *bit*, certificat de inovator nr. 420/1986
- *Automat de acces la magistrala comună a unui multiprocesor pe* 8 *bit*, certificat de inovator nr. 423/1986
- *Logică pentru evitarea blocării magistralei comune în sistemele de calcul multiprocesor*, certificat de inovator nr. 487/1996
- *Metodă şi schemă de configurare dinamică a sensului de transfer a datelor la porturile paralele programabile*, certificat de inovator nr. 1072/1989