Arhitecturi pentru procesare paralelă

Prof.dr.ing. Gheorghe DODESCU,
Catedra de Informatică Economică, A.S.E., Bucureşti

Conceptul de sistem de calcul, constând din mai multe procesoare care lucrează în paralel pe diferite probleme sau asupra diferitelor părţi ale aceleiaşi probleme, nu este nou. Aspecte privind maşinile pentru calcul în paralel se regăsesc în literatura de specialitate din anul 1950. După această perioadă şi până în prezent s-au făcut eforturi în cercetare pentru înţelegerea şi implementarea conceptului de procesare paralelă. În ultimii ani există sute de proiecte în lume implicând diferite arhitecturi paralele pentru diferite tipuri de aplicaţii.
***Cuvinte cheie**: granularitate, paralelism, pipeline, SISD, SIMD, MISD, MIMD.*

Introducere

O caracteristică fundamentală a unui sistem de procesare paralelă este "granularitatea". Prin granularitate se înţelege dimensiunea unităţii de lucru alocate la procesare. Astfel de unităţi de lucru determină la rândul lor dimensiunea procesoarelor unei maşini şi numărul acestora. Există o legătură între dimensiunea unităţilor de lucru şi numărul procesoarelor. Paralelismul este clasificat în mare astfel:

- **Paralelism cu granulă grosieră** ("coarse-grain") care implică procese de calcul din nivelul exterior programului de control (astfel ca un ciclu iterativ al unui program); paralelismul implică un număr redus de procesoare mari şi complexe. Exemple de calculatoare pentru acest tip de paralelism: Cray 2 şi Cray X-MP, fiecare din ele cu patru procesoare şi sistemele ETA cu opt procesoare; super sisteme care costă circa 10 milioane de dolari, au o putere de calcul 100-1000 milioane operaţii/sec în virgulă mobilă (MFLOPS). În acest tip de paralelism este, de obicei, folosită descompunerea unei probleme în task-uri, care necesită să comunice între ele numai temporar.
- **Paralelism cu granulă fină** ("fine-grain"), care implică o unitate de lucru ce se execută de către o singură instrucţiune, cum ar fi evaluarea unei expresii sau chiar o singură operaţie logică sau aritmetică. Un astfel de paralelism implică un număr mare de procesoare, mici şi simple. Paralelismul cu granulă fină se poate întâlni în sisteme cu mii de procesoare simple, ca: Connection Machin of Thinking Machines Corporation, STARAN şi sistemele MPP Goodyear Aerospace şi în VLIW (Very Long Instruction Word) care au arhitecturi cu zeci de procesoare. În cadrul acestei arhitecturii, task-urile ce pot fi executate independent în fiecare procesor sunt de dimensiune redusă şi comunicaţia între procesoare poate domina volumul de lucru. De aceea obţinerea unei eficienţe superioare este dificilă. Pentru o aplicaţie dată, numărul de procesoare care trebuie utilizate eficient depinde de paralelismul intrinsec dintre aplicaţie şi dimensiunea problemei.
- **Paralelism cu granulă-medie** ("medium-grain") conţine un domeniu limită relativ la configuraţie, putere de calcul şi cost. Ea este o posibilitate intermediară între cele două tipuri de paralelism discutate anterior. Majoritatea sistemelor de procesare paralelă existente pe piaţă pot fi considerate medii, folosind microprocesoare puternice dar ieftine, ca: National Seminconductor's seria 3200, sistemele Motorola, MC68.000 şi M88.000, Intel 80.286 şi 80.386. Exemple de astfel de sisteme includ Encore Computer's busbased Multimax,

BBN's Butterfly şi Intel's şi PSC hypercube. Conceptele de bază pentru arhitecturile de procesare paralelă care au fost propuse şi sunt comercializate sau investigate într-o serie de studii şi proiecte sunt prezentate în fig.1.

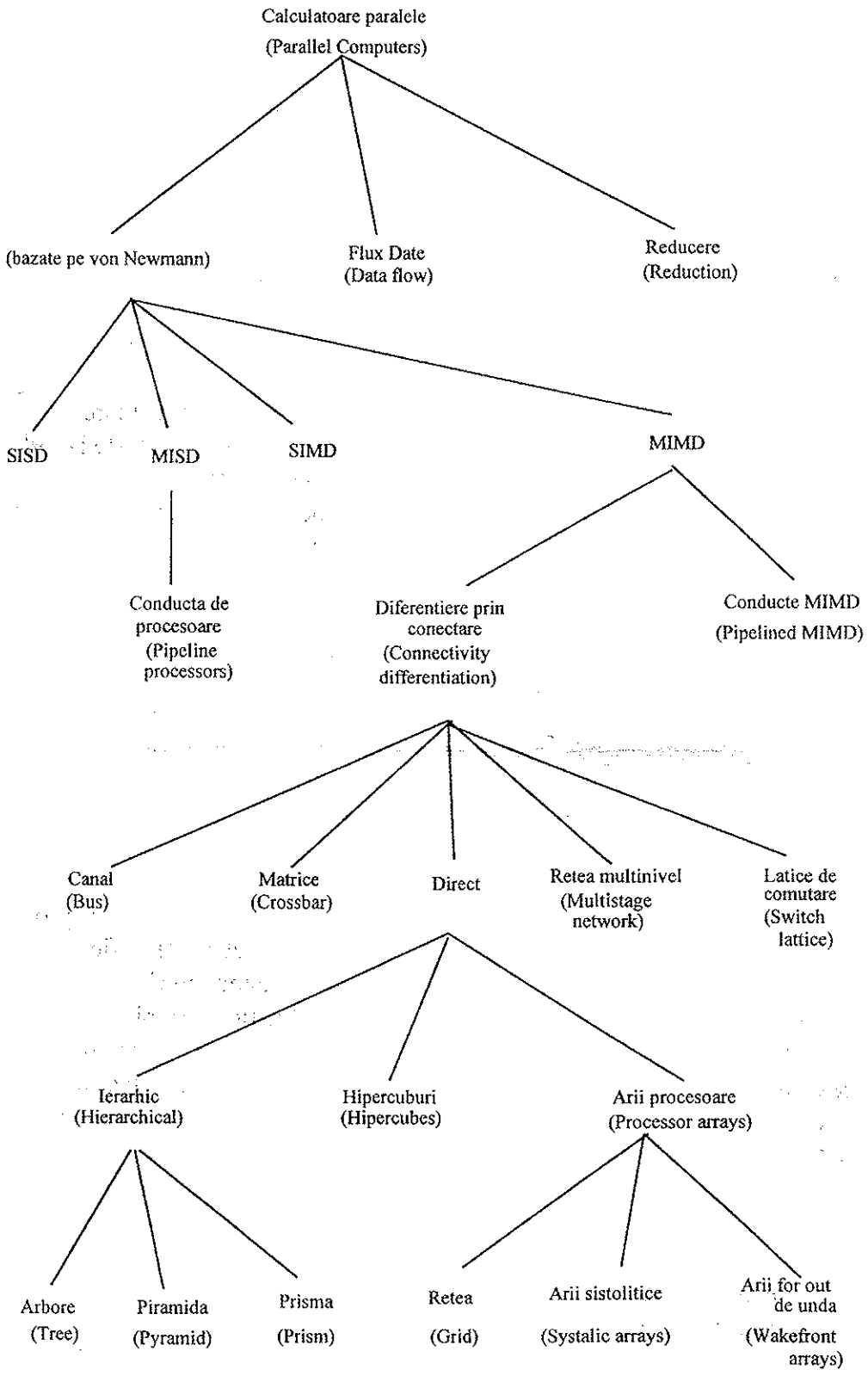


Fig.1. Principalele tipuri de arhitecturi pentru procesarea paralel

Conceptul **von Neumann** (uneori referit contra-comandă) pentru procesarea paralelă constă din interconectarea a două sau mai multe monoprocesoare de tip Neumann într-o varietate de configuraţii. Conceptul "**data flow**" (uneori referit "data driver" - control date) are la bază modul de execuţie a instrucţiunilor programului în ordinea în care operanzii aferenţi sunt gata, în loc de execuţia secvenţei dictate de ordinea instrucţiunilor din program, ca în conceptul von Neumann.

Conceptul "**reduction**" (referit uneori "demand-driver" - comanda la cerere) constă din execuţia instrucţiunilor când rezultatele sunt necesare pentru alte calcule. Programele sunt privite ca o colecţie de aplicaţii şi execuţia de bază se realizează prin reducerea succesivă a aplicaţiilor cele mai interioare, conform cu semantica operatorilor lor până când nu mai sunt aplicaţii. Calculele sunt executate numai când sunt necesare şi nu oricând operanzii lor sunt gata, ca în "**data flow**". În conceptul hibrid, procesoarele, în loc de a lucra pe orice instrucţiune care este gata de execuţie (ca în data flow), vor lucra la început pe instrucţiunile solicitate de ele dacă operanzii corespunzători sunt gata. Dacă operanzii nu sunt la dispoziţie, procesoarele îi vor cere de la alte procesoare care lucrează pe entităţi cu prioritate redusă (calcule data flow).

O dată cu dezvoltarea calculatoarelor un număr din ce în ce mai mare de operaţii elementare au fost executate prin suprapunere în timp. în cadrul celei de a treia generaţii de calculatoare au fost adăugate **n** UAL (unitaţi aritmetico logice) capabile să lucreze în paralel cu unitatea centrală. De asemenea, în a treia generaţie a fost implementat conceptul de lucru în conductă "**pipline**". O conductă de calculatoare este analoagă cu o linie de asamblare într-o fabrică de automobile. Operaţia calculatorului este divizată în **n** etape, în fiecare etapă se execută o operaţie elementară asupra unui set de operanzi.

Un set de operanzi intră în etapa E_1 se execută o operaţie elementară asupra lor şi rezultatul este transmis etapei E_2 unde se execută o altă operaţie asupra lor, iar rezultatul este transmis etapei E_3 ş.a.m.d. până când rezultatul final se obţine după etapa E_n la /E. Toate cele **n** etape se pot executa simultan, astfel **n** etape într-o conductă ("pipeline") pot opera pe o secvenţă de **n** seturi de operanzi, în acelaşi timp. Când primele seturi de operanzi de I/ sunt în etapa E_1 a conductei, celelalte **n-1** etape sunt libere în aşteptare. La sfârşit, când ultimul set de operanzi (rezultate) părăseşte etapa E_n la /E, primele **n-1** etape ale conductei sunt libere şi în aşteptare. De aici rezultă că prelucrarea în "**pipeline**" este eficientă pentru o secvenţă mare de operanzi sau pentru operanzi de tip vectori cu dimensiuni foarte mari. Datorită acestui fapt procesoarele din conductă sunt adesea numite procesoare-vector. Există o serie de căi pentru execuţia concurentă a operaţiilor pe calculatoare digitale . Posibilitatea extinderii apare chiar atunci când proiectantul plasează mai multe procesoare în cadrul sistemului. Conceptul de "**prelucrare paralelă**" se poate defini ca metodă de organizare a operaţiilor într-un sistem de calcul, astfel încât să se poată executa **n** operaţii concuremte (simultan), $n > 2$.

Termenul de procesare paralelă este general şi cunoaşte foarte multe metode de implementare. în literatură există o mare gamă de metode de clasificare a procesării paralele, dar metoda cea mai acceptată este dată de Flynn ("Flynn clasification"). în prezent metoda Flynn utilizează următoarele concepte:

- **Pr**, un procesor care conţine o unitate aritmetico-logică şi circuitele logice adiţionale;
- **UC**, unitate de control, care controlează Pr şi alte subsisteme, prin transmite-

rea unor semnale logice de control, adică instrucţiuni;
-MP, memoria principală divizată într-un număr de module;
-ML, memoria locală, de obicei asociată cu un Pr specific;
-I, instrucţiuni (semnale de control) sau secvenţă (stream) ;
-D, secvenţă de date (sau strem) ;
-P, un procesor complet, Unitate Centrală, care conţine unitatea de control (UC) şi Unitatea Aritmetico-Logică (UAL).
Sistemele de procesare paralelă care au la bază principiul von Neumann sunt calsificate conform modului în care procesează strimurile de instrucţiuni şi date. Clasificare Flynn recunoaşte patru tipuri de sisteme de bază.

O singură instrucţiune, o singură entitate de dată

(SISD -Single Instruction Single Data)

Sistemul este monoprocesor, el reprezintă o arhitectură clasică von Neumann cu nici un fel de paralelism (IBM 701). Totuşi se pot exemplifica anumite sisteme mai sofisticate unde au fost implementate anumite elemente de paralelism, cum ar fi unitaţi funcţionale multiple (IBM 360/91, CDC 6600, CYBER 205) respectiv conducte (UA x 8600) sau ambele.

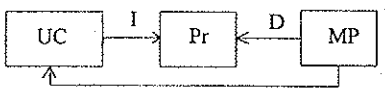


Fig.2. SISD

În acest caz calculatoarele sunt von Neumann tradiţionale (monoprocesor), dar printr-o extindere adecvată a conceptului de microprogramare orizontală, este posibilă interconectărea a **n** monoprocesoare şi controlul lor simultan, respectiv sincron, cu ajutorul unui cuvât instrucţiune lung (VLIW - Very Long Instruction Word). Aparent calculatorul are paralelism prin SO deoarece suportă task-uri multiple.

O instrucţiune şi date multiple

(SIMD - Single Instruction Multiple Data)

Sistemul conţine **n** procesoare care execută simultan aceleaşi instrucţiuni transmise de UC în strimuri de instrucţiuni. Fiecare instrucţiune se execută pe un set de date diferit transmise la fiecare procesor Pr_i sau sub forma unui stream de date D_i din memoria locală ML_i, cu i = = 1,2, ..., n. Rezultatele sunt memorate temporar în ML_i. Există un bus bidirecţional (bus interconnection) între MP şi ML_i. Programul este memorat în MP şi transmis la UC. Un astfel de sistem se mai numeşte şi arie de procesoare ("Array Processor"). Hardware-ul este constituit dintr-o arie de procesoare Pr_i.

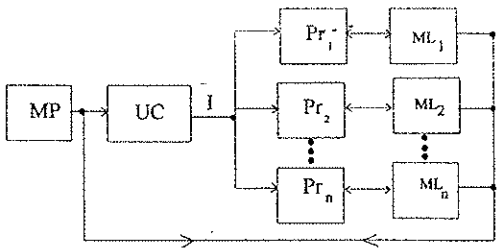


Fig.3. SIMD

Exemple de sisteme SIMD: ILLIAC, BSP şi MPP. Iniţial această paradigmă de prelucrare pare restrictivă, dar este foarte bine folosită pentru calcule paralele cu masive (vectori, matrice). Aceste structuri se găsesc în multe aplicaţii inginereşti, cum ar fi: prelucrarea imaginilor, simularea particulelor, metoda elementului finit. De exemplu, pentru calculul graficului temperaturii într-un apartament, se foloseşte ecuaţia lui Laplace în două dimensiuni. Această aplicaţie în urma discreditării conduce la un sistem linear **AX=B** cu 10^3 necunoscute, funcţie de pasul de discretizare introdus şi de condiţiile iniţiale. În acest caz pentru rezolvare trebuie calculată

matricea inversă $\mathbf{A^{-1}}$, operație care implică același fel de calcule asupra unor date diferite. În concluzie, paradigma (modelul) SIMD este foarte bună pentru această aplicație. Fiecare soluție a sistemului **AX=B** este doar aproximativă, dar repetând execuția modelului **AX=B** (sub diverse condiții) se realizează o convergență către soluția finală.

De asemenea, există calcule SPMD (Sygle Program Multiple Data). De fapt SPMD nu este o paradigmă hardware, mai mult el este software echivalent la SIMD. În cazul SPMD se rulează același program dar cu date diferite. Datorită faptului că programul se execută pe date diferite, există posibilitatea ca anumite ramuri ale programului să fie la fel și rezultă un pararlelism asincron. SPMD este considerat ca un paralelism trivial dar este frecvent întâlnit în activitatea practică. **SPMD** diferă de **SIMD** pentru că procesele sunt sincrone numai la începutul și sfârșitul procedurii sau a secțiunii de cod care este duplicată pentru toate procesoarele. Procesoarele execută asincron, în interior, fiecare procedură (sau secțiuni identice de cod), obținând o pseudo operare **SIMD**.

Multiple instrucțiuni, o singură entitate de dată

(MISD - Multiple Instruction Single Data)

În acestă arhitectură, în orice moment considerat, instrucțiunile consecutive ale unui program se găsesc în diferite stadii de execuție prin avansarea în cadrul unei conducte a unităților funcționale, câte o funcție la un moment dat. O secvență de date (Di) este transmisă la un set de procesoare (Pr_i), fiecare procesor fiind controlat de o unitate de control (UC_{01}) și execută o secvență diferită de instrucțiuni. Această arhitectură seamănă cu o structură conductă. Diferența fiind că o structură în conductă aparține la aceiași UC și este controlată de o asingură unitate de control (UC_0).

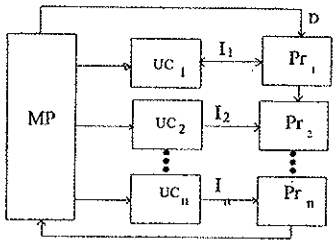


Fig.4. MISD

Instrucțiuni multiple, date multiple

(MIMD - Multiple Instruction Multiple Data)

Arhitectura execută în mod simultan diferite seturi de instrucțiuni pe diverse seturi de date. Arhitectura MIMD conține **n** procesoare care execută simultan diferite secvențe de instrucțiuni pe diferite seturi de date. Acest tip de sistem este numit multiprocesor. Arhitectura MIMD poate suporta seturi de instrucțiuni multiple prin conductă sau prin procesoare complet separate.În cazul când există procesoare complet separate, se poate realiza o subclasificare și în funcție de modul în care sunt conectate memoriile și procesoarele.

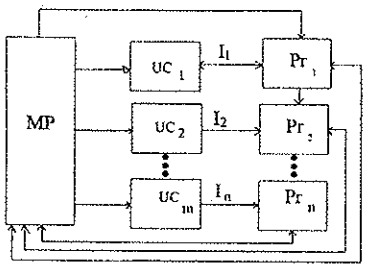


Fig.5. MIMD

În cazul **MIMD** se pot distinge arhitecturi care utilizează: un bus, legături în matrice (crassbar), rețea cu nivele multiple, latice de comutație sau conectoare directă între elementele de procesare (procesoare/memorii). Sistemele de procesare paralelă de tip **MIMD** care utilizează o conectare directă între elemen-

tele de procesare pot fi: hipercuburile, arii de procesare sau cele care au o structură ierarhică. Ariile de procesoare pot fi arii de tip plasă care pot avea un paralelism cu granulă fină ("fine-grain") sau arii celulare, respectiv paralelism cu granulă medie până la granulă mare. Ele sunt caracterizate prin felul în care datele se deplasează pas cu pas, având procesoare care execută operaţii simple asupra datelor pe care le primesc la fiecare pas. Sistemele MIMD includ: structuri de tip arbore, piramidă şi prismă. Exemple de astfel de sisteme: IBM 3090, Cray 2, Alliant FX/8 şi HCUBE. Procesoarele execută concurent programe diferite, folosind diferite seturi de date. În general nu există restricţii la selectarea operaţiei. Programul se poate divide în subprograme (procese, task-uri, etc.) care pot fi rulate concurent pe un număr de procesoare. Este posibil ca în acelaşi multiprocesor, anumite procesoare să execute părţi ale unui program, în timp ce alte procesoare sunt ocupate cu execuţia altui program.

Paradigma **MIMD** este cel mai general model de paralelism. Sincronizarea este atinsă în mod explicit şi local mai mult decât prin mecanismul de sincronizare globală. Modelul este flexibil dar asta înseamnă că software-ul este mult mai sofisticat, deoarece există multe detalii de controlat. Deoarece paradigma **MIMD** este flexibilă există o varietate mare de module de programare care pot ilustra aspectele **MIMD**. Paradigma **MIMD** este utilă când problema de rezolvat oferă task-uri multiple, eterogene şi care se pot executa în acelaşi timp. De asemenea se preferă ca numărul task-urilor să nu fie cunoscut de la început. În continuare se prezintă o aplicaţie care ilustrează acest lucru. Presupunem că se cere elementul maxim dintr-un vector V=(15, 5, 21, 25, 34, 2, 8, 6, 10). Funcţia Max trebuie să fie implementată pe calculatorul serie comparând cel mai mare element cu fiecare element întâlnit; găsind elementul maxim dintre două elemente se continuă compararea ş.a.m.d. Această operaţie implică **n** comparări unde **n** = lungimea vectorului. În cazul **MIMD** împărţim lucrul între diverse procesoare. Versiunea paralelă este realizată în două faze (fig.6).

$\mathbf{F_1}$ Primul procesor împarte vectorul în doi subvectori şi îi plasează la două procesoare. Fiecare din cele două procesoare fac acelaşi lucru; împart subvectorii în două părţi şi le transmit la alte două procesoare. Acest lucru continuă până când procesoarele finale primesc un subvector cu una sau două componente.

$\mathbf{F_2}$ Procesoarele (exemplu nivelele din arbore) aşteaptă pentru un Max local, ele selectează elementul maxim şi transmit la procesoarele de la care a primit subvectorul. Această operaţie se repetă de fiecare procesor până se obţine maximul dorit.

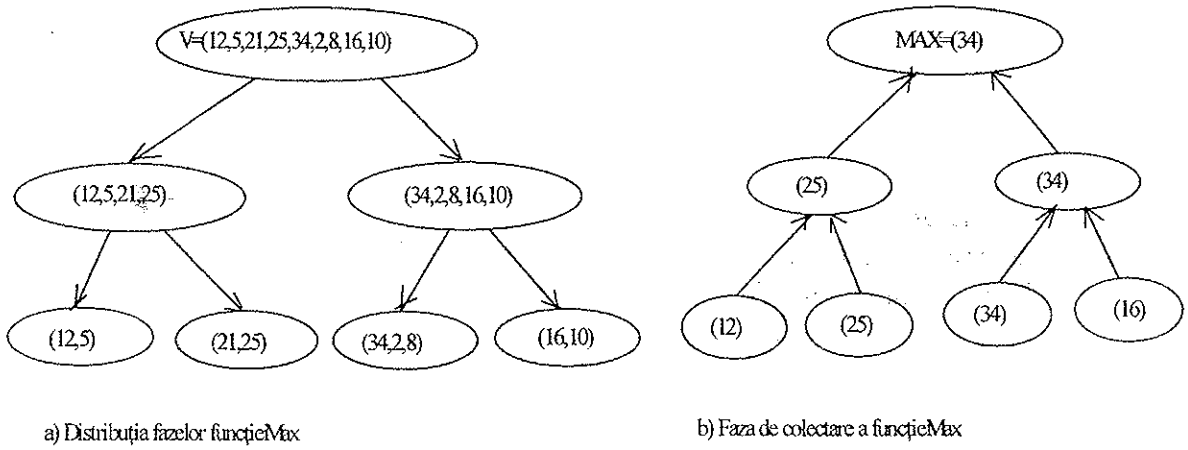
Din această prelucrare se observă că:

a) Fiecare procesor operează independent, afară de faptul când aşteaptă (un sincronism ocazional); procesoarele execută lucrări scurte; procesorul care a primit un singur element al vectorului, termină înaintea celui care a primit două elemente; procesoarele pot efectua la un moment dat diferite operaţii, ca: aşteptare, comparare, expediere a rezultatelor.

b) Se observă că soluţia implică un număr necunoscut de task-uri când începe programul de execuţie paralelă şi deci se utilizează iniţial un număr necunoscut de procesoare; nu se cunosc nivelele în cadrul arborelui până când vectorul nu este divizat de un număr de ori până apare una sau două componente la fiecare procesor; desigur nivelele arborelui pot fi cunoscute dacă se cunoaşte lungimea **n** a vectorului, dar în general **n** nu se cunoaşte până nu începe rularea programului.

Acest exemplu evidenţiază două trăsături esenţiale ale lui **MIMD**: mecanismul de centralizare sau mecanismul de sincronizare globală; generalitatea task-

urilor eterogene operând simultan, chiar dacă se execută operații diferite pe date diferite și în segmente diferite de timp. MIMD este suficient de generală pentru a satisface paradigma reduction/data-flow. De asemenea, SIMD este suficient de acoperitoare, pentru că se poate simula comportarea SIMD prin introducerea unor restricții la MIMD, folosind o programare atentă. În forma complexă o mașină este compusă din procesoare SISD și orice procesor este capabil de a suporta **n** task-uri de la aplicații diferite în același timp. Într-adevăr, multe sisteme cu memorie partajată și multiprocesoare diferite suportă **n** task-uri UNIX pe fiecare procesor (exemplu: Sequent și Encore).



a) Distribuția fazelor funcțieMax

b) Faza de colectare a funcțieMax

Fig.6. Implementarea funcției Max cu ajutorul paradigmei MIMD

Bibliografie

A.L. DeCegama - *The Technology of parallel processing,* Prentice - Hall International, inc. 1995

W.W. Wadge - *The data flow programming Language,* New York, Academic Press, 1995

G.F. Plister - *The IBM Research Parallel Procesor Prototype (RP3): Introduction and Arhitecture,* 1992

S.Y. Kung - *Warefront Array Processors - Concept to Implementation,* IEEE Computer, July 1989

D.D. Cripps - *A parallel Graph Reduction Machine,* New York, Springer Verlag, 1989.