

APP – teorie

1. Excludere mutuală bazat pe XCHG

Instrucțiunea XCHG(r,m) interschimbă, ca o operație atomică, conținutul registrului r cu celula de memorie m (semafor). □ deci r este o variabilă locală a sarcinii Si și este inițializată cu 0, iar □ m este o variabilă globală, comună tuturor sarcinilor care este inițializată cu 1.

Ambele variabile pot lua valorile 0, 1. Algoritmul pentru Si va fi:

<inceput sarcina>;

Repetă

XCHG(ri ,m);

până_când ri;

<secțiune critica>;

XCHG(ri ,m);

<rest sarcina> ;

Numai procesul care găsește m=1 va intra în secțiunea critică și va trece în zero. La ieșire din secțiunea critică va returna pe m la valoarea 1 pentru a permite celorlalte sarcini să intre în secțiunea critică. (m=1 este o convenție, în unele sisteme se consideră m=0) . Când o sarcina Si intră în secțiunea critică, ri=1.

2. algoritmul optim de partitie a unui microprogram descris prin graf de dependențe de date vs algoritmul euristic de partitie

In cadrul algoritmului optim de partitie a unui microprogram, descris prin graf de dependențe de date, se va realiza partitia optima prin generarea din setul de microoperatii uODk a tuturor uIC posibile, pentru fiecare dintre ele se va analiza influenta pe care o are asupra generarii setului de microoperatii disponibile urmator, uODk + 1. In cazul algoritmului euristic de partitie, el este bazat pe ordonare a microoperatiilor din setul disponibil uODk in functie de succesorii in graful de dependente de date. In cadrul

algoritmului heuristic nu se va genera insa o partitie optima, dar va fi mult mai rapida in unele situatii. partitie heuristica = bazata pe ordonarea operatiilor in functie de succesiuni -> complexitate liniara algoritm optim de partitie = NP completa

3. Descrieți principiul de funcționare a sistemelor de tip Data Flow . Comentați componentele principale ale arhitecturii.

Sistemele de tip Data Flow au un model ce este bazat pe fluxul de date, in cadrul caruia executia unei instructiuni de lucru depinde de disponibilitatea operandului. In cadrul acestor sisteme nu avem Program Counter. Specifica efectuarea unor operatii imediat ce vom avea operandii disponibili si din aceasta cauza nu avem nevoie de contorul de program. De asemenea, nu este nevoie de adrese pentru memorarea variabilelor iar operatia se efectueaza numai atunci cand token-ul este prezent pe ambele intrari ale operandului. Datele se stocheaza in cadrul instructiunilor, adica vom mentine un token cu valoare binara in cadrul instructiunii pentru fiecare data. O unitate specializata va verifica disponibilitatea datelor. Ca avantaje avem faptul ca avem un paralelism ridicat si o viteza de executie mai mare. De asemenea este usor de implementat partea de comunicatie si sincronizare, insa vom avea overhead mare pentru control, iar manipularea structurilor de date va fi dificila. Avem Token = {Data, Tag Destinatie, marker} si Match = {Tag, Destinatie}. Avem Df1, Df2, ..., Dfn ce reprezinta retea de masini de baza ce sunt interconectate printr-o retea de comutare; Token Queue cu care pe masura ce vom avea token-uri, ele vor fi adaugate in coada Matching Unit - elementul de lucru are un token pe fiecare ramura; Instruction store - ia elementele de lucru si le pune pe o unitate de procesare

4. Exemple de calculatoare paralele

(aici la ce se referă? Taxonomia Flynn cu extinderea pe tipuri de arhitecturi paralele?)

5. Teorema de suficienta + demonstratie

Sistemele de sarcini formate din sarcini mutual neinterferente sunt determinante

Justificare (prin inducție)

- Pentru un sistem de sarcini format dintr-o singura sarcina este evident.
- Presupunem că afirmația este adevărată pentru sisteme cu n-1 sarcini, ($\alpha'1$ și $\alpha'2$ sunt secvențe de execuție pentru un sistem cu n-1 sarcini)
- Presupunem sistemul C = (S, \prec fiind obținută din \prec eliminând relația de precedență ce implică pe S. $V(M_i, \alpha'1) = V(M_i, \alpha'2)$ din ipoteza de inducție.
- Deci se poate presupune că valorile din DS sunt aceleasi, atât pentru $\alpha'1$ cât și pentru $\alpha'2$,
- respectiv $F(M_i, \alpha'1) = F(M_i, \alpha'2)$ pentru M_i DS .

- Rezultă astfel că pentru $\alpha'1$ și $\alpha'2$, sarcina S scrie aceeași valoare v pentru orice celulă $M_i RS$. Pentru $M_i RS : V(M_i, \alpha_1) = V(M_i, \alpha'_1 SI SF)$ Lemă = $V(M_i, \alpha'_1)$
 $M_i RS$ și ip. de inducție = $V(M_i, \alpha'_2)$ $M_i RS = V(M_i, \alpha'_2 SI SF)$ Lemă = $V(M_i, \alpha'_2)$
) Pentru $M_i RS$ $V(M_i, \alpha_1) = V(M_i, \alpha_2)$ Pentru $M_i RS : V(M_i, \alpha_1) = V(M_i, \alpha'_1 SI SF)$ Lemă = $(V(M_i, \alpha'_1), v)$ ip. de inducție = $(V(M_i, \alpha'_2), v) = V(M_i, \alpha'_2 SI SF)$
 Lemă = $V(M_i, \alpha_2)$ Pentru $M_i RS$ $V(M_i, \alpha_1) = V(M_i, \alpha_2)$
- Sistemul C este determinat dacă sarcinile sunt mutual neinterferente.

6. Limita inferioara

Limita inferioară = μ , limita inferioară pentru situația în care resursele sunt distincte.

$$\mu = \max \{|\mu_{oP_i}|\} \quad 1 \leq i \leq P$$

Fiind o singur resursa, se va fi controlata secvențial, deci avem $|\mu_{oP_i}|$, fiind $|\mu_{oP_i}|$ microoperatii.

Când resursele nu ar fi fost diferite, am putea considera că

$$\mu = \max \{[|\mu_{oP_i}| / |P_i|]\}$$

În calculul acestei limite, s-a presupus că nu există dependență de date între microoperatiile care controleaza diferite resurse.

Eu cred ca aici nu vrea asta, ci mai degraba să ii explic la ce se referă limita astă inferioara, eventual pe scurt cazurile respective. Si să ii mai explic și de ce vrei să afli limita astă inferioara. Chiar nu cred că vrea formulele și explicații la formule.

7. Sun-Ni

Introduce modelul bazat pe limita de memorie

- $T_s = f_{seq} + g(p)(1 - f_{seq})$
- unde $G(n)$ este creșterea de memorie a unui procesor
- $T_p = f_{seq} + g(p)(1 - f_{seq}) / p$
- $V_p = T_s / T_p = (f_{seq} + g(p)(1 - f_{seq})) / (f_{seq} + g(p)(1 - f_{seq}) / p)$
- Caz 1 : $g(n) = 1$
- $V_p = (f_{seq} + 1 - f_{seq}) / (f_{seq} + (1 - f_{seq}) / p) = 1 / (f_{seq} + (1 - f_{seq}) / p) \sim Amdahl$
- Caz 2 : $g(n) = p$
- $V_p = T_s / T_p = (f_{seq} + p(1 - f_{seq})) / (f_{seq} + p(1 - f_{seq}) / p) = (f_{seq} + p(1 - f_{seq})) / 1 = f_{seq} + p(1 - f_{seq}) \sim Gustafson$
- Caz 3 : $g(p) = m > p$
- $V_p = T_s / T_p = (f_{seq} + m(1 - f_{seq})) / (f_{seq} + m(1 - f_{seq}) / p) = (f_{seq} + m(1 - f_{seq})) / (m/p + (1 - m/p)f_{seq})$

8. Sisteme de clasa determinate, secvența de execuție parțială

9. Clase compatibile. Clase incompatibile

10. Modalitati de codificare microinstructiuni. Avantaje si dezavantaje pentru fiecare

11. Metode de calcul paralel: enumerare + comparatie intre memoria partajata si procesare sistolica

12. Amdhal + Worldon + Gustafson

13. Indicatori paraleli.

Viteza de prelucrare – $V_p = T_1 / T_p$

Eficienta - $E_p = V_p / p$

Redundanta - $R_p = O_p / O_1$

Utilizarea - $U_p = O_p / (p * T_p) \leq 1$

14. Sistem de sarcini determinat. Secvente de executie. Secventa partiala de executie.

Secventa de executie: Alfa = a₁a₂a₃...a_i, a_{i+1}, ... a_{_2n}. a_i = eveniment ce declanseaza sau termina o sarcina, cu respectarea ordinii de precedenta < (< relatie de ordine partiala).

Secventa de executie partiala: orice prefix dintr-o secventa de executie valida.

Sistemul de sarcini este determinat daca indiferent de ordinea de executie a sarcinilor independente (adica pentru orice secventa alfa valida) si indiferent de timpul necesar unei sarcini independente, se produce acelasi rezultat si, in plus, daca acest lucru se respecta pentru orice interpretare f_s a sistemului. (f_s = ceea ce executa fiecare sarcina).

** E corect si ce scrie mai jos, dar e dintr-un curs de mai tarziu. Deci le-as zice pe amandoua.

Un sistem de sarcini C este determinat dacă pentru orice stare inițială s0 dată, $V(M_i, a) = V(M_i, a')$, $1 \leq i \leq m$, pentru toate secvențele de execuție a, \dots, a' din C.

15. Tipuri de microinstructiuni

microinstructiuni operaționale care controlează primitivele funcționale ale unității de execuție a sistemului numeric, asigurând fluxul de informație și acțiunile asupra resurselor;

microinstructiune de ramificație (de salt) care inspectează starea primitivelor funcționale și asigură ramificația în algoritmul de control, constituind suportul pentru implementarea deciziilor.

16. Sisteme microprogramate vs. microprogramabile

17. Algoritmul de partitionare (in 8 pasi).

P1. Initializare

Se initializeaza partitia $\mu_{PT} = \emptyset$, $\mu_{PTA} = \mu_B(\mu_O)$ - tot microsublocul (deci, daca le luam si le facem sevcentral)

$\mu_{OD} = \{\mu_{Oj} \mid \forall \mu_{Oj} \in \mu_B(\mu_O) \text{ nu exista } \mu_{Oj} < \mu_{Oj}\}$ - deci pentru care nu exista predecesori

$\mu_{OND} = \{\mu_{Oj} \mid \mu_{Oj} \in \mu_B(\mu_O) \setminus \mu_{OD}\}$ - operatii nedisponibile

P2. Daca $\mu_{OND} = \emptyset$ si $\mu_{OD} \leq 3$ atunci salt la pas 5

P3. Se genereaza $\{IC\}$ – multimea de microinstructiuni complete

$\{IC\} = \{\mu_{ICj}, \dots, \mu_{ICk}\}$ - din OD curent daca $j \neq k$ (s-au generat mai multe IC) atunci salvare $PT_j = PT \cup OD_j = setul curent OD \{IC\}_j = \{IC\} \setminus IC_j$

P4. $\mu_{PT} = \mu_{PT} \cup \mu_{ICj}$ $\mu_{OD} = \mu_{OD} \setminus \mu_{ICj} \cup \{\mu_{Od}\}_j$ $\mu_{OND} = \mu_{OND} \setminus \{\mu_{Od}\}_j$

daca $\mu_{OD} \neq \emptyset$ si $\mu_{PT} \leq \mu_{PTj} - 1$ atunci salt pas 2

P5. daca $OD = \emptyset$ atunci salt pas 7

P6. Se genereaza μ_{IC} din μ_{OD} curent. $\mu_{PT} = \mu_{PT} \cup \mu_{IC}$; $\mu_{OD} = \mu_{OD} \setminus \mu_{IC}$;

P7. daca $\mu OD \neq \Phi$ si $|\mu PT| < |\mu PTA|$ atunci $\mu PTA = \mu PT$ altfel daca $|\mu PTA| \leq n \mu I$ atunci μPTA este optima STOP.

P8. daca $\{\mu IC\}_j \neq 0$ (cel care a fost salvat la pasul 3) atunci reface $\mu PT = \mu PT_j$; $\mu OD = \mu OD_j$; $j \leftarrow j + 1$; (trec la urmatorul, selecteaza urmatoarea IC din setul generat) $\mu IC = \mu IC_j$ $\mu OND = \{\mu O_j \mid \mu O_j \in \mu B(\mu O) \setminus (\mu PT \cup \mu OD)\}$ salt pas 4 altfel PTA este optima .

20. Fie un procesor care implementează o structura paralela pipeline de citire interpretare executie pentru procesare suprascală , cu patru unități paralele. Presupunând un ciclu de instrucțiuni contine:

- citire care necesită o singură perioadă de ceas
- decodificarea instrucțiunii necesită două perioade de ceas
- execuția instrucțiunii necesită două perioade de ceas Avem o secvență de 3000000 de instrucțiuni. Considerând frevența ceasului în Ghz de 1. Calculați durata de execuție a secvenței de program în milisecunde

$$(1 + 2 + 3 \cdot 10^6 / 4 * 2) * 10^{-9} \approx 1.5 \text{ ms}$$

Considerând că în urma analizei microoperațiilor dintr-un subloc au rezultat microinstrucțiunile complete, mIC1,...,mIC5 care conțin micro[1]operațiile ca în tabelul alăturat:

mIC1 = mo1 mo2 mo3 mo4 mo5 mo6

mIC2 = mo3 mo7 mo8 mo9

mIC3 = mo1 mo2 mo8 mo9 mo10

mIC4 = mo4 mo8 mo11

mIC5 = mo6 mo8

Care este organizarea optimă a câmpurilor din formatul general al microinstrucțiunilor:

a. Câmp 1: (mo1)

Câmp 2: (mo2)

Câmp 3: (mo3)

Câmp 4: (mo4 mo7 mo9 mo10 mo11)

Câmp 5: (mo5)

Câmp 6: (mo6)

Câmp 7: (mo8)

b. Câmp 1: (mo1)

Câmp 2: (mo2)

Câmp 3: (mo3)

Câmp 4: (mo4 mo9 mo10)

Câmp 5: (mo5 mo11)

Câmp 6: (mo6 mo7)

Câmp 7: (mo8)

c. Câmp 1: (mo1)

Câmp 2: (mo2)

Câmp 3: (mo3)

Câmp 4: (mo4 mo9)

Câmp 5: (mo5 mo11)

Câmp 6: (mo6 mo7 mo10)

Câmp 7: (mo8)

Ar fi D, dar dc? -- n-am idee, din ce văd și C este validă, nu văd ce ar fi greșit la ea -> se calculeaza si costul: pt fiecare camp, numeri cate instructiuni are, adaugi 1 (o instructiune noua imaginara, la fiecare camp), si obtii N -> te gandesti de cati biti ai nevoie pt a reprezenta N numere (ex: N = 4 => 2 biti, N = 5 => 3 biti); aduni toti bitii necesari pt toate campurile si obtii **costul**. Se alege varianta cu **costul minim** (exemple: campul 1 de mai jos are 1+1 instructiuni, e suficient 1 bit; campul 5 de mai jos are 3+1 instructiuni, sunt suficienti 2 biti - se va observa ca varianta c are costul **10**, iar d are costul **9**)

d. Câmp 1: (mo1)

Câmp 2: (mo2)

Câmp 3: (mo3)

Câmp 4: (mo4)

Câmp 5: (mo5 mo7 mo10)

Câmp 6: (mo6 mo9 mo11)

Câmp 7: (mo8)

1. Legile Amdahl, Worlton, Gustafson si Sun-Ni
2. Limita inferioara pentru nivele de alocare a sistemelor de sarcini
3. Modele de calcul paralel (comparatii: sistolic, data flow, transfer mesaje)
4. Sistem de sarcini determinat, nedeterminat si secenta partiala de executie
5. Pasii generali ai algoritmului de repartizare microoperatii in microinstructiuni complete
6. Teorema de necesitate cu demonstratie
7. Teorema de suficienta cu demonstratie
8. Clasa de compatibilitate + incompatibilitate + cost + cum se determina
9. Algoritm heuristic + optim de impartire pe niveluri
10. Dati exemple de sisteme cu arhitectura paralela
11. Modele sistolic (memorie partajata cu transfer de mesaje)
12. Concatenarea sistemelor de sarcini
13. Sisteme echivalente
14. Structura unei unitati de comanda microprogramate
15. Indicatori de performanta (prelucrari paralele)
16. Diferenta intre sistemele microprogramate si microprogramabile
17. Modalitati de implementare a memoriei de control
18. Organizarea microoperatiilor
19. Legatura dintre algoritmi paraleli si arhitecturi paralele

1. Legile Amdahl, Worlton, Gustafson si Sun-Ni

Amdahl

Stabileste o limită a creșterii de viteză a structurilor paralele în raport cu structurile monoprocesor.

R_H = rata de execuție pe structura paralelă

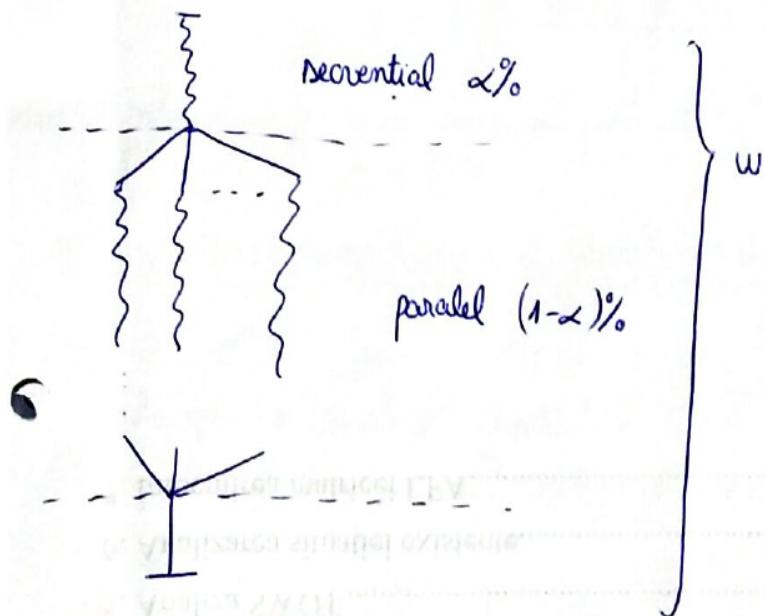
R_L = rata de execuție pe structura monoprocesor

f = procentul nevenitului de instrucțiuni executată în paralel

$1-f$ = procentul —||— nevenitual

Formula de bază: $R = \frac{1}{\frac{f}{R_H} + \frac{1-f}{R_L}}$

Formă redusă a formulei:



Cresterea de viteză :

$$V_p = \frac{T_1}{T_p} = \frac{w}{\alpha w + (1-\alpha)w} = \frac{1}{\alpha + \frac{1-\alpha}{P}}$$

Indiferent de numărul de procesare, creșterea de viteză este limitată de caracteristicile intrinseci ale programului.

! Dimensiunea problemei rămâne constantă cu creșterea dimensiunii sistemului.

! Portiunea resequentială a problemei limitează speed-up-ul total care poate fi atins în ciuda creșterii resurselor.

Worlton

Un program paralel constă din

- o secțiune securitățială
- o secțiune paralelă
- o secțiune de sincronizare

$$\Rightarrow \begin{cases} t_D = \text{temp sincronizare} \\ t_O = \text{temp overhead} \\ t = \text{temp mediu pt. 1 task} \\ N = \text{nr. task-uri} \\ P = \text{nr. proceseare} \end{cases}$$

Timpul pentru un procesor: $T_1 = Nt$

Timpul pentru P proceseare: $T_{NP} = t_D + \left\lceil \frac{N}{P} \right\rceil \cdot (t + t_O)$

Crescerea de viteză:

$$V_p = \frac{T_1}{T_{NP}} = \frac{Nt}{t_s + \left[\frac{N}{P} \right] (t+t_0)} = \frac{1}{\underbrace{\frac{t_s}{Nt}}_{\text{numarizare}} + \underbrace{\frac{1}{N} \left[\frac{N}{P} \right] \left(1 + \frac{t_0}{t} \right)}_{\text{overhead}}}$$

Pentru rezultate optime:

- P divisor al lui N
- $\frac{t_s}{N \cdot t}$ cât mai mic
 - t_s cât mai mic
 - mărirea intervalului între numarizări
- $\frac{t_0}{t}$ cât mai mic
 - t_0 cât mai mic
 - creșterea granularității task-worker

GUSTAFSON

← Dimensiunea problemei scăzăză
până la o limită fixă a
timpului

- A arătat că oramenii nu sunt interesați de rezolvarea unei probleme în cel mai scurt timp (Amdahl), ci mai degrabă de rezolvarea problemei celei mai mari într-un timp rezonabil
- Folosește m procesare pentru rezolvarea unui task

$$w' = \alpha w + (1-\alpha)wm \quad \text{unde} \quad \begin{cases} \alpha = \text{procentul de execuție secesională} \\ 1-\alpha = \text{procentul de execuție paralelă} \end{cases}$$

Gestarea de viteză:

$$V_p = \frac{T_1}{T_m} = \frac{\alpha w + (1-\alpha)wm}{\alpha w + (1-\alpha)wm} = \frac{\alpha w + (1-\alpha)wm}{\alpha w + (1-\alpha)w} = \frac{\alpha + (1-\alpha)m}{\alpha + (1-\alpha)} = \alpha + (1-\alpha)m$$

- Sugerează că e benefică construirea unui sistem paralel mare, pt. că speed-up-ul poate crește liniar cu dimensiunea sistemului dacă se menține același timp de execuție.

SUN & NI

- dimensiunea problemei neobrază până la o limită netă de dimensiunea memoriei sistemului
- generalizare pentru Amdahl și Gustafson
 - când $G(m) = 1 \Rightarrow$ Amdahl
 - când $G(m) = m \Rightarrow$ Gustafson

$$w = \underbrace{\alpha w}_{\text{componenta}} + (1-\alpha) w \cdot G(m)$$

cărțigul pe care îl avem prin adăugarea memoriei

Găsirea de viteză:

$$V_p = \frac{T_i}{T_m} = \frac{\alpha w + (1-\alpha) w \cdot G(m)}{\alpha w + (1-\alpha) w \cdot G(m) / m} = \frac{\alpha + (1-\alpha) G(m)}{\alpha + (1-\alpha) G(m) / m}$$

Caz A : nu adăugăm memorie suplimentară $\Rightarrow G(m) = 1$

$$V_p = \frac{\alpha + (1-\alpha)}{\alpha + \frac{1-\alpha}{m}} = \frac{1}{\alpha + \frac{1-\alpha}{m}} = \text{Amdahl}$$

Caz B : $G(m) = m$

$$V_p = \frac{\alpha + (1-\alpha)m}{\alpha + \frac{(1-\alpha)m}{m}} = \frac{\alpha + (1-\alpha)m}{\alpha + 1-\alpha} = \frac{\alpha + (1-\alpha)m}{1} = \text{Gustafson}$$

2. Limita inferioara pentru nivele de alocare a sistemelor de sarcini

CALCULUL LIMITEI INFERIOARE PENTRU PROCESUL DE ALOCARE

Microsubbloc

$$\mu_{SB} = (\mu_B(\mu_0), \alpha)$$

$$\mu_B = \{\mu_0, \dots, \mu_{IB}\}$$

$$P = \{P_1, \dots, P_{IP}\} \leftarrow \text{elemente de execuție (procesare)}$$

μ_0 : acionează asupra lui P_j .

Limita inferioară a nivelelor de alocare

CF2.1 Calculăm înălțimea grafului de dependență de date h

Calea cea mai lungă în G de la nodul initial către nodul final

CF2.2 : Calculăm p.

Tinem cont de distribuția pe procesare

\Rightarrow Împărțim graful : $G = \{G_1, \dots, G_{IP}\} \Leftrightarrow$ identificăm sub-grafurile din graful initial pentru corespunzătoare fiecărui procesor

$$G_i = \{\mu_0 \mid \mu_0 \text{ controlă } P_i / \text{se execută pe } P_i\}$$

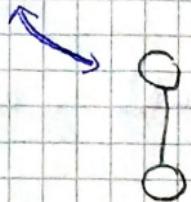
Aceste subgrafe ar putea să aibă mai multe sarcini ?

Definim $\text{SGM}_{ij} \subset G_i$ cu proprietatea că: $\forall \mu_0 \in \text{SGM}_{ij}$, atunci

↓
subgraf maximal

există $\sigma \mu_{0k}$ astfel încât:

$\mu_0 < \mu_{0k}$ SAU $\mu_{0k} < \mu_0$



$G_i = \bigcup \text{SGM}_{ij}$ ← combinare paralelă

$\mu_{0r} \Rightarrow$ nu există μ_{0i} astfel încât $\mu_{0r} < \mu_{0i}$

↑
Microoperatie terminală

$\mu_{0rj} \Rightarrow$ microoperatie terminală în SGM_{ij}

→ $\|\mu_{0rj}\| =$ poziția sarcinii terminale în graful G
număr

Pentru fiecare G_i , calculăm ρ_i (limita de nivel pentru fiecare subgraf)

După, urcăm să calculăm ρ folosind ρ_i și sarcinile terminale din fiecare subgraf.

$$1 < |\mu_{0rjk}| \leq N_{\mu_{0r}}$$

↑ numărul de sarcini terminale

$$\rho_i = \max(\rho_{ij})$$

$$\rho_{ij} = |\text{SGM}_{ij}| + \min_k \|\mu_{0rjk}\|$$

$$\rho = \max_i \left(\max_j \left(\text{SGM}_{ij} + \min_k (\|\mu_{0rjk}\|) \right) \right)$$

↑ estimare a numărului minim al nr. de alocare

OBSERVATTE!

Dacă procesările nu sunt distincte, definim și astfel:

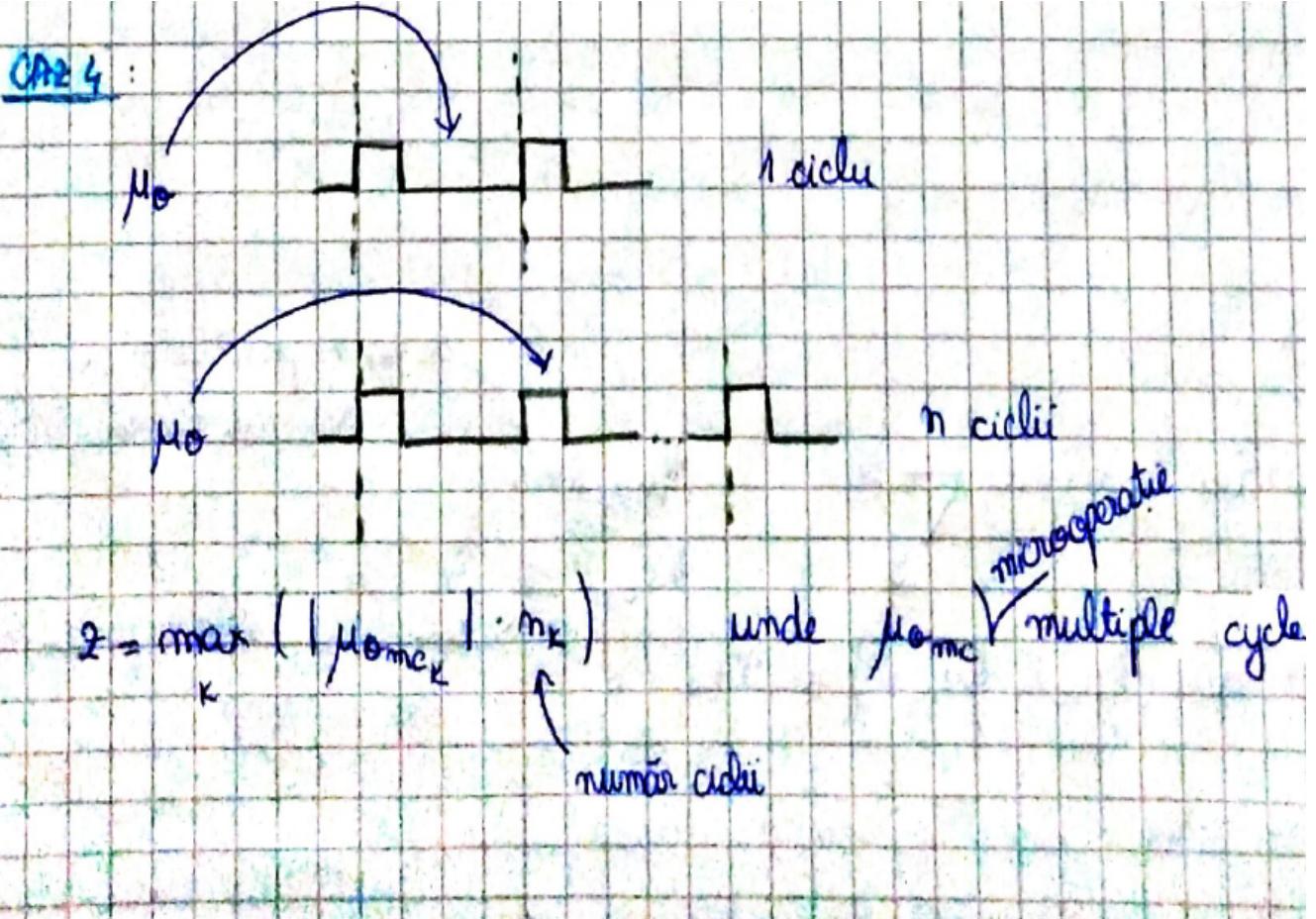
$$g = \max_i \left(\max_j \left(\max \left(\frac{|SGM_{ij}|}{|P_i|}, h(SGM_{ij}) \right) + \min_k \| \mu_{T_{jk}} \| \right) \right)$$

↑ ↗
 înălțimea
 subgrafului
 maximal ij din cauză
 dependenței de date

CA2 3 :

- nu considerăm dependente de date

$$\mu = \max_i \left(\frac{\mu_B(\mu_0)_{P_i}}{|P_i|} \right)$$



CONCLuzie

Limita inferioară este: $\max(\lambda, p, \mu, z) = L_i$

3. Modele de calcul paralel (comparatii: sistolic, data flow, transfer mesaje)

MODELUL RAM (RANDOM ACCESS MACHINE)

- Acest model este similar cu modelul mașinii Turing.
- Un astfel de model este utilizat pentru dezvoltarea de algoritmi și analiza complexității acestora, constituind punctul de plecare pentru modelele parallele.
- Are
 - o bandă de intrare
 - o bandă de ieșire
 - program
- program counter \Rightarrow folosit pentru execuția programului

$$\text{RAM} = \{ S, \Gamma, \Sigma, f, s_0, z \}$$

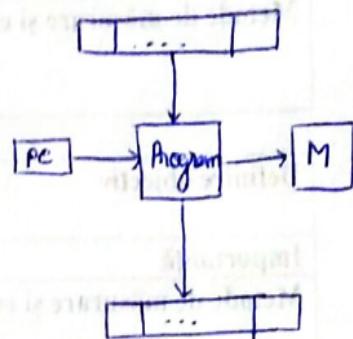
multime finită a stărilor

alfabetul benzii

alfabetul benzii de intrare

multimea stărilor finale ($z \subseteq S$)
stare initială ($s_0 \in S$)

funcție de tranziție: $f: S \times \Gamma \rightarrow S \times \Gamma$

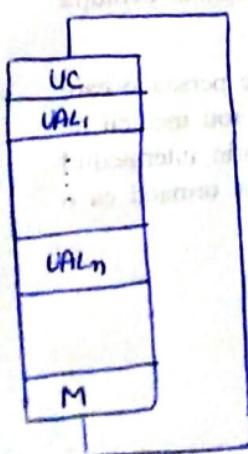
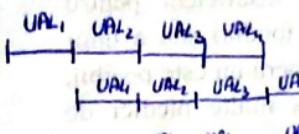


Pipeline

Bresupune divizarea unui task T în subtask-uri T_1, T_2, \dots, T_k și de a le atribui unor elemente de procesare.

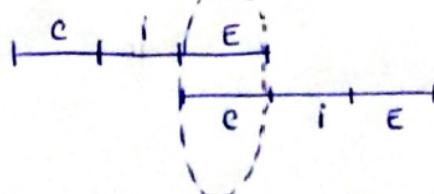
Paralelismul se poate realiza la nivel de \rightarrow preleucrare aritmetică

\rightarrow citire, interpretare, executare instrucțiunii



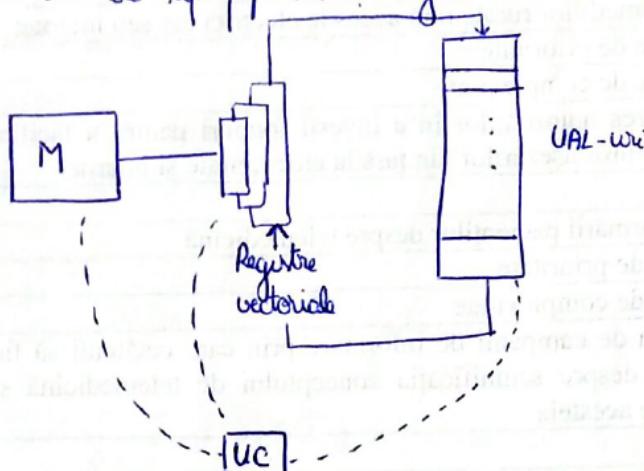
UC are mai multe faze: UCl_1, \dots, UCl_m

Orice UC se ocupă de citirea, interpretarea și executarea instrucțiunilor maxime. După ce începe să execute, se poate apăsa să citească o altă instrucție.



PROCESOARE DE VECTORI

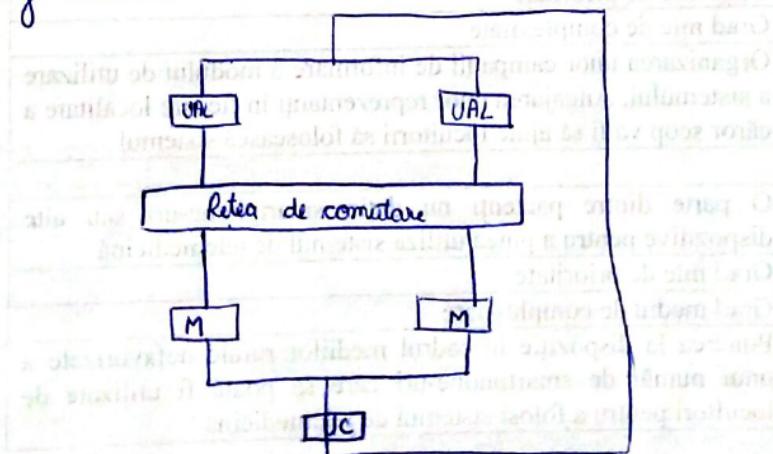
- există un set de instrucțiuni care tratează vectorul ca operand simplu
- există o UAL de tip pipeline și registre vectori (fiecare UAL se ocupă de un registru)



- poate fi asociată cu SIMD, dar spre deosebire de SIMD:
 - nu avem multiple unități de procesare
 - este o masină reentrantă, dar are mai multe UAL-uri

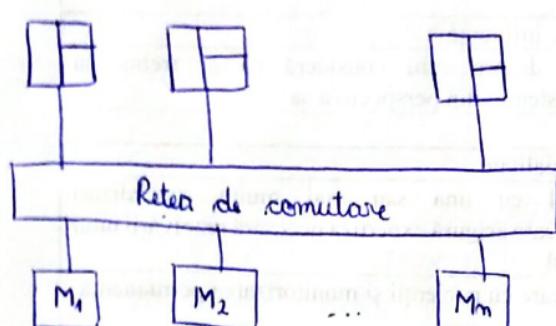
PROCESARE MASIVĂ DE DATE

- Se referă la calculatoare parallele sincrone, care constau din UAL multiple, supervizate de o singură UC și care efectuează aceeași operatie la un moment dat.
- UAL efectuează aceeași operatie asupra unor date diferite
- poate fi asociat cu un SIMD extins



MEMORIE PARTAJATĂ

- Fiecare procesor conține propria unitate de prelucrare, propria memorie și propria unitate de comandă
- Se comunică prin intermediul unei rețele de comutare cu module de memorie partajată.
- Nu încadrează la NUMA

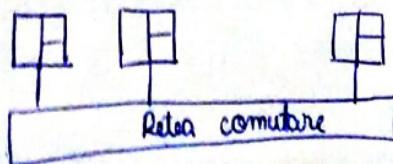


PRAM - STRUCTURĂ PARALELĂ CU ACCES ALEATOR

- extensie a RAM
- comunicatie intre procesare si memorie fara introducere de intarziere
- memorie comună disponibilă tuturor programelor
- 4 modalități de acces la memoria PRAM:
 - EREW
 - CREW
 - ERCW
 - CRCW

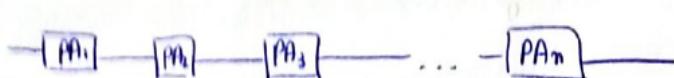
TRANSFER PRIN MESAJE

- Nu se mai partajează memoria
- Memoria este distribuită fiecărui procesor a.i. fiecare dispune de propriul program și propria memorie de date.
- Interconexiunea se realizează prin schimburile de mesaje prin intermediul unei rețele de comutare mesaje
- Exemplu: cluster (nu putem să vedem memoria reuniată, dar folosim send și receive)



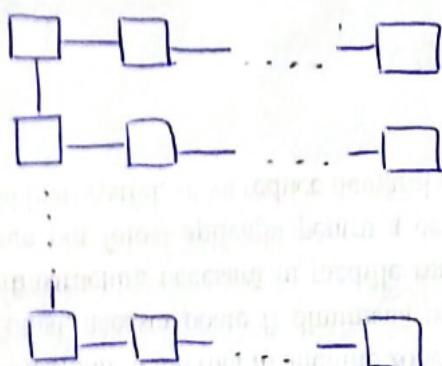
PROCESARE SISTOLICĂ

- constă în elemente de procesare identice aranjate într-o structură pipeline



PA=procesor aritmetic

Pipeline liniar

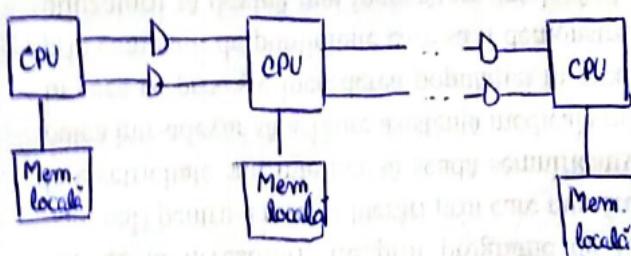


Pipeline matriceal

- se caracterizează prin interconexiuni simple ce permite integrarea VLSI

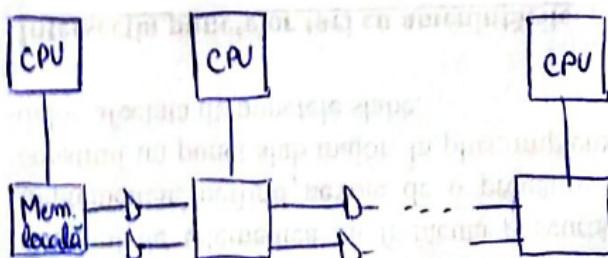
Metode de comunicare în sistemele sistolice:

①



După prelucrare, punem într-un buffer, iar următorul năște să ia de acolo.

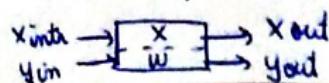
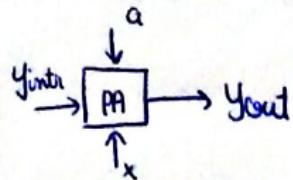
(2)



Prelucră ceva și pun rezultatul în memorie. În acest moment există un acces direct de memorie care mută datele de ieșire în altă zonă de memorie ca fiind date de intrare \Rightarrow necesită implementarea unei DMA

Ex: $y = y_0 + Ax$

- ABORDARE 1: Sequential $\Rightarrow O(n^2)$
- ABORDARE 2: Pipeline sequential \Rightarrow Cost: $m \cdot PA$
 $\Rightarrow O(m)$
- ABORDARE 3: Pipeline matriceal \Rightarrow Cost: $n^2 \cdot PA$

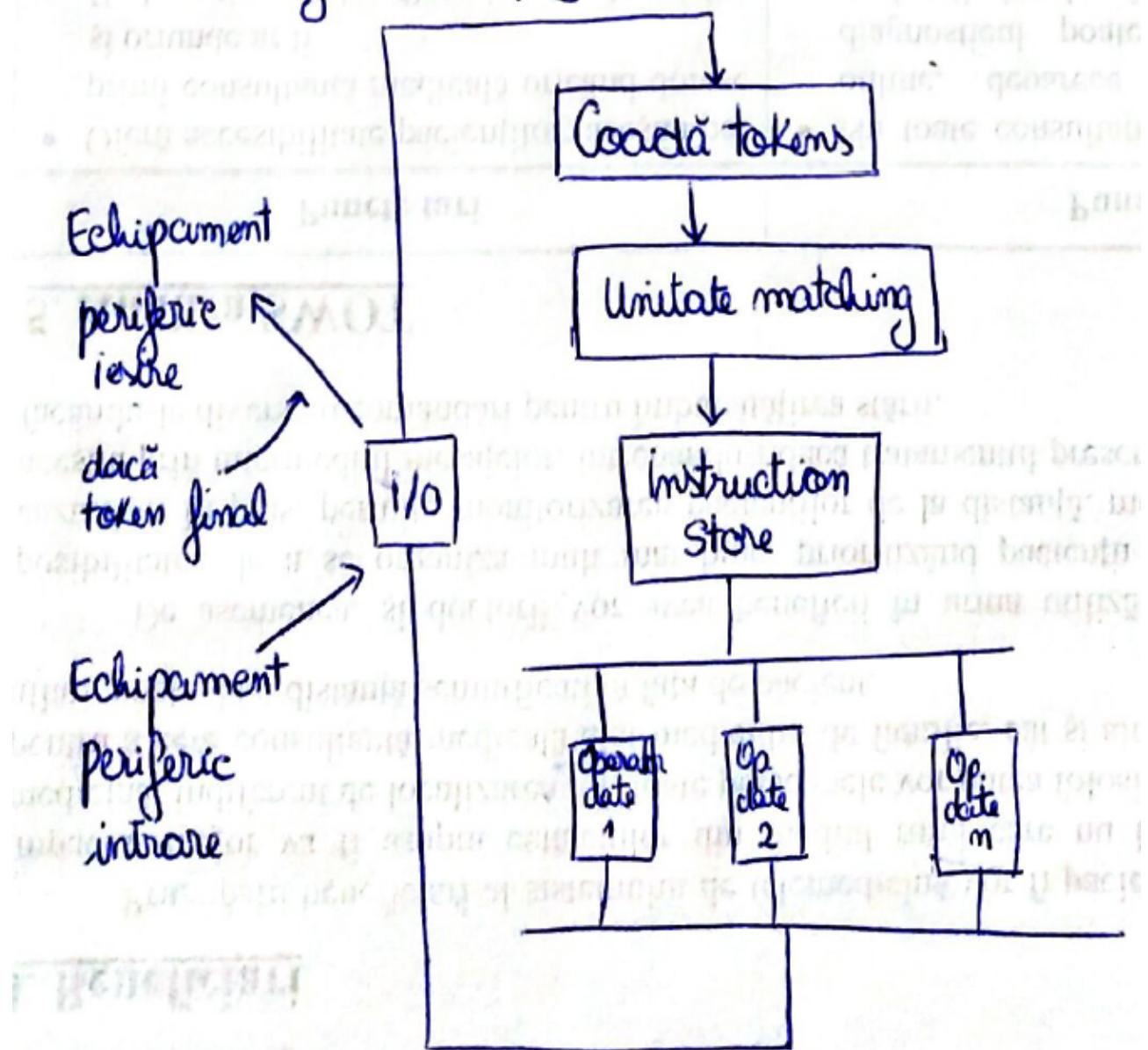


DATA FLOW

- Specifică efectuarea unei operații imediat ce operandii devin disponibili.
⇒ Nu elimină necesitatea existenței contorului de program
- nu necesită adrese pentru memorarea variabilelor
- operația nu efectuează numai când token-ul este prezent pe ambele intrări ale operandului
- datele nu se stochează în cadrul instrucțiunilor (nu menține un token cu valoare binară în cadrul instrucțiunii pentru fiecare dată)
- disponibilitatea datelor este verificată de o unitate specializată, dedicată
- Avantaje
 - paralelism ridicat
 - viteză ridicată de execuție
 - ușor de implementat comunicarea și sincronizarea
- Deavantaje
 - overhead mare pentru control
 - manipulare dificilă a structurilor de date

- Token = { Data, Tag, Destinatie, marker }

Match = { Tag, Destinatie }



4. Sistem de sarcini determinat, nedeterminat si secventa partiala de executie

• SISTEM DE SARCINI

Fie $S = \{s_1, \dots, s_n\}$ o multime de sarcini, iar \prec o relatie de ordine paritală, nereflexivă, denumită relație de precedență.

Perechea $C = (S, \prec)$ este un sistem de sarcini.

- $s_i \prec s_j \Leftrightarrow$ Pt. a intia s_i execută s_j , trebuie să se fi terminat s_i
- $s_i \nmid s_j$ sunt **independente** cind \prec dintre ele este \emptyset , adică $s_j \neq s_i$

• SECVENTĂ DE EXECUȚIE

O secvență de execuție a unui sistem cu m sarcini $C = (S, \prec)$ este orice sir α , $\alpha = a_1, a_2, \dots, a_{2m}$ de evenimente de inițiere și terminare a proceselor cu respectarea relațiilor de precedență impuse de \prec .

$\bar{s} \rightarrow$ inițiere sarcină

$\underline{s} \rightarrow$ terminare sarcină

Proprietăți:

- α nu e unic
- pentru $\forall s \in S$, $\bar{s} \nmid \underline{s}$ apăr o singură dată în α
- dacă $a_i = \bar{s} \nmid a_j = \underline{s}$, atunci $i < j$
- dacă $a_i = \underline{s_k} \nmid a_j = \bar{s_p}$ cu $s_k \prec s_p$, atunci $i < j$

• SEQUENȚĂ DE EXECUȚIE PARȚIALĂ

Este orice prefix al unei secrete de execuție.

$$\alpha_p = a_1 \dots a_p, p < 2n$$

• SPATIUL STĂRILOR

$$T = \Delta_0 \Delta_1 \dots \Delta_{2n}$$

↑ stare initială

• PROPRIETATEA DE DETERMINARE / NEDETERMINARE

Un sistem de sarcini este considerat **determinat** dacă executându-se în paralel, respectând relațiile de precedență, indiferent de durata și ordinea de execuție a unor sarcini independente, rezultatul este același.

Un sistem de sarcini este considerat **nedeterminat** dacă rezultatele produce de sarcini independente depind de ordinea în care acestea se execută.

Nedeterminarea se poate rezolva introducând o relație de precedență între procese care erau independente.

5. Pașii generali ai algoritmului de repartizare microoperatii în microinstructiuni complete

$\mu SB = (\mu B(\mu), <)$ microsublocul pe care trebuie să îl partiziționăm pe niveluri (seturi de microinstructiuni complete)

$\mu PT =$ partitia care se va genera

$\mu PTA =$ partitia de microinstructiuni anterioara, considerata cea mai buna pana in momentul respectiv

$\mu OD =$ setul de micro-operatii disponibile la momentul curent

$\mu OND =$ setul de micro-operatii nedisponibile la momentul curent

$n_{MPL} =$ numarul de micro-instructiuni complete care s-ar putea genera

$\mu IC =$ micro-instructiunea completa generata din μOD

$\{\mu od\} =$ multimea de micro-operatii disponibile rezultate ca urmare a generarii μIC

Pas 1.	<p>Initializare</p> $\mu_{PT} = \Phi$ $\mu_{PTA} = \mu_B(\mu_o)$ $\mu_{OD} = \{ \mu_o \mid \mu_o \in \mu_B(\mu_o) \text{ nu exista } \mu_{oi} \text{ astfel ca } \mu_{oi} < \mu_o \}$ $\mu_{OND} = \mu_B(\mu_o) \setminus \mu_{OD}$
Pas 2.	Daca $\mu_{OND} = \Phi$ si $ \mu_{OD} \leq 3$ atunci pas 5
Pas 3.	<p>Se genereaza</p> $\{\mu_{IC}\} = \text{multimea de } \mu_I \text{ complete } \{\mu_{IC}\} = \{\mu_{IC_j}, \dots, \mu_{IC_k}\}$ <p>Daca $j \neq k$ (s-a generat o singura μ_{IC})</p> <p>Atunci salvare</p> $\mu_{PT_j} = \mu_{PT};$ $\mu_{ODj} = \mu_{OD} \text{ curent};$ $\{\mu_{IC}\}_j = \{\mu_{IC}\} \setminus \mu_{IC_j}$
Pas 4.	$\mu_{PT} = \mu_{PT} + \mu_{IC_j}$ $\mu_{OD} = \mu_{OD} \setminus \mu_{IC_j} \cup \{\mu_{od}\}_j$ $\mu_{OND} = \mu_{OND} \setminus \{\mu_{od}\}_j$ <p>Daca $\mu_{OD} \neq 0$ si $\mu_{PT} \leq \mu_{PT_j} - 1$ atunci salt la Pas 2</p>
Pas 5.	<p>Daca $\mu_{OD} = \Phi$</p> <p>Atunci salt Pas 7.</p>
Pas 6.	<p>Se genereaza</p> <p>μ_{IC} din μ_{OD} curent</p> $\mu_{PT} = \mu_{PT} + \mu_{IC}$ $\mu_{OD} = \mu_{OD} \setminus \mu_{IC}$ <p>Daca $\mu_{OD} \neq \Phi$ si $\mu_{PT} \leq \mu_{PTA} - 1$ atunci salt Pas 4. altfel salt Pas 2.</p>
Pas 7.	<p>Daca $\mu_{PT} \leq \mu_{PTA}$</p> <p>atunci</p> <p>altfel daca $\mu_{PT} \leq nm\mu_I$</p> <p>atunci μ_{PTA} este optima STOP</p>
Pas 8.	<p>Daca $\{\mu_{IC}\}_j = 0$ (cel care a fost salvat la Pas 3.)</p> <p>Atunci reface $\mu_{PT} = \mu_{PT_j}$</p> $\mu_{OD} = \mu_{ODj}$ <p>$j=j+1$ (selecteaza urmatoarea μ_{IC} din setul generat)</p> $\mu_{IC} = \mu_{IC_j}$ $\mu_{OND} = \mu_B(\mu_o) \setminus \mu_{PT} \setminus \mu_{OD}$ <p>salt Pas 4.</p> <p>Altfel μ_{PTA} este optima STOP</p>

6. Teorema de necesitate cu demonstratie

Enunt:

Un sistem de sarcini C este interpretat, dar la care stim domeniile de definitie si de valori este determinat pentru orice interpretare a sarcinilor sale numai daca acestea sunt mutual neinterferente.

Demonstratie:

$$C = (S, \prec)$$

$$S = \{s_1, \dots, s_n\}$$

Demonstram prin reducere la absurd.

$S \neq S'$ sunt independente, dar sunt interferente

Vrem sa vedem daca produc acelasi rezultat.

$$\alpha = \beta_1 \bar{s} \leq \bar{s}' \leq \beta_2$$

$$\alpha' = \beta_1 \bar{s}' \leq \bar{s} \leq \beta_2$$

Consideram celula M_i de memorie care va fi folosita nu de s nu de s'

$$M_i = R_s \cap R_{s'}$$

$$f_s \rightarrow u$$

$$f_{s'} \rightarrow v$$

$$V(M_i, \alpha) = V(M_i, \beta_1 \bar{s} \leq \bar{s}' \leq \beta_2) = V(M_i, \beta_1 \bar{s} \leq \bar{s}' \leq \beta_2) = V(M_i, \beta_1)uv$$

$$V(M_i, \alpha') = V(M_i, \beta_1 \bar{s}' \leq \bar{s} \leq \beta_2) = V(M_i, \beta_1 \bar{s}' \leq \bar{s} \leq \beta_2) = V(M_i, \beta_1)vu$$

Decarce de obicei $u \neq v \Rightarrow$ ne produc rezultate diferite \Rightarrow Trebuie ca $R_s \cap R_{s'} = \emptyset$

$$M_j \in D_s \cap R_{s'}$$

$$M_i \in R_s$$

$$V(M_i, \alpha) = V(M_i, \beta_1 \bar{s} \leq \bar{s}' \leq \beta_2) \xleftarrow{s' \text{ nu produce pt. ca } \notin R_{s'}} V(M_i, \beta_1 \bar{s}) = V(M_i, \beta_1)u$$

$$V(M_i, \alpha') = V(M_i, \beta_1 \bar{s}' \leq \bar{s} \leq \beta_2) = V(M_i, \beta_1 \bar{s}' \leq \bar{s} \leq \beta_2) = V(M_i, \beta_1)v$$

$$\Rightarrow D_s \cap R_{s'} = \emptyset$$

$$\text{Analog } D_{s'} \cap R_s = \emptyset$$

\Rightarrow Din toate cele 3 \Rightarrow sistem neinterferent

7. Teorema de suficientă cu demonstratie

Enunț :

Sistemele de sarcini formate din sarcini mutual neinterferente sunt determinat

Demonstratie (prin inducție)

$$C = (S, \prec)$$

$$S = \{S_1, \dots, S_m\}$$

• PAS 1 : $m=1$ (A)

• PAS 2 : Pp. că afirmația este adevărată pentru n sarcini

$$V(M_i, \alpha'_1) = V(M_i, \alpha'_2)$$

valoare lui M_i

în urma secenței α'_1

• PAS 3 : Demonstrăm pentru m

Fie α'_1, α'_2 secențe de execuție pt. sistemul C' cu $m-1$ sarcini

$$C' = (S \setminus S_i, \alpha')$$

$$\alpha'_1 = \alpha'_1 \bar{S} S_i$$

$$\alpha'_2 = \alpha'_2 \bar{S} S_i$$

• $M_i \notin R_{S_i}$

domeniul de valori

$$V(M_i, \alpha'_1) = V(M_i, \alpha'_1 \bar{S} S_i) = V(M_i, \alpha'_1) \downarrow \text{din ipoteză}$$

$$= V(M_i, \alpha'_2) \quad \text{nu produce nimic pt. că } M_i \notin R_{S_i}$$

• $M_i \in R_{S_i}$

$$V(M_i, \alpha'_1) = V(M_i, \alpha'_1 \bar{S} S_i) = V(M_i, \alpha'_1) \cup = V(M_i, \alpha'_2) \cup = V(M_i, \alpha'_2 \bar{S} S_i)$$

$$= V(M_i, \alpha'_2)$$

$$\Rightarrow V(M_i, \alpha'_1) = V(M_i, \alpha'_2)$$

8. Clasa de compatibilitate + incompatibilitate + cost + cum se determină

• CLASA DE COMPATIBILITATE : CC

Două operații μ_0 și μ_1 sunt compatibile dacă pentru orice k ($k \in [1, |μPT|]$) are loc următoarea relație :

dacă $\mu_{0k} \in μ_{ik}$, atunci $\mu_{1k} \notin μ_{ik}$ $\Leftrightarrow \mu_{0k} \text{ și } \mu_{1k} \text{ nu se execută nesimultan }$ sau paralel

$$cc(\mu_0) = \{\mu_0 \mid \forall \mu_0 \text{ și } \mu_1 \text{ sunt compatibile}\}$$

Notă: în care oricare 2 μ_0 sunt compatibile între ele

• CLASĂ DE COMPATIBILITATE MAXIMĂ : MCC

Este acea clasă de compatibilitate la care nu se mai poate adăuga nicio instrucțiune fără a pierde compatibilitatea.

$$MCC(\mu_0) = \{\mu_0 \mid \forall \mu_0 \notin MCC(\mu_0), \exists \mu_1 \in MCC(\mu_0) \text{ a.i. } \mu_{0i} \in μ_{ic} \text{ și } \mu_{1j} \in μ_{ic}\}$$

• COST CLASĂ DE COMPATIBILITATE

- este nr. biti necesari pentru a codifica toate μ_0 din acea clasă

$$\text{cost}(cc(\mu_0)) = \lceil \log_2(|cc(\mu_0)| + 1) \rceil$$

↑
pentru NOP

$$\text{Cost}(\text{Structura } μ_{ic}) = \sum \text{cost}(cc_i(\mu_0))$$

• CLASA DE INCOMPATIBILITATE : IC

Două μ_0 sunt incompatibile dacă \exists cel puțin o $μ_{ic}$ a.i. $\mu_{0i} \in μ_{ic}$ și $\mu_{1j} \in μ_{ic}$.

• CLASA DE COMPATIBILITATE MAXIMĂ ASOCIAȚĂ UNIEI MIC : (AMCC asociată lui MIC)

Pt. că IC, clasele de compatibilitate maximă ce conțin un element din IE ne numesc AMCC.

$$AMCC = \{MCC \mid \forall \mu_0 \in MIC_m, \exists \mu_0 \in MCC \text{ și } \mu_0 \in IC_m\}$$

Proprietăți:

- Pt. $\forall \text{Mic}$, reuniunea MCC asociate acoperă întreg blocul de μ_0 .
- Pt. $\forall \text{Mic}$, reuniunea a K MCC, $K < |\text{Mic}|$, nu acoperă întreg blocul de μ_0 .
- O partitie a setului de μ_0 în $g+h$ cămpuri are cost mai mare decât partitie cu g cămpuri.

9. Algoritm heuristic + optim de împartire pe niveluri

PAS 1

$$\mu_{PT} = \emptyset$$

PAS 2

Definim setul de $\mu_{oD} \times \mu_{oND}$

$$\mu_{oD} = \{\mu_0 \mid \mu_0 \in \mu_B(\mu_0) \times \exists \mu_{oi} < \mu_0 \text{ să nu fi fost într-o } \mu_i\}$$

$$\mu_{oND} = \{\mu_0 \mid \mu_0 \in \mu_B(\mu_0), \mu_B(\mu_0) \setminus \mu_{oD}\} \text{ la nivelul } K\}$$

PAS 3

Din μ_{oD} generăm μ_i 'e

$$\{\mu_{ic}\} = \{\mu_{ic_1}, \dots, \mu_{ic_K}\}$$

Se alege μ_i cu nr. maxim succesiuni $\Rightarrow \mu_{ich}$

$$\mu_{PT} = \mu_{PT} \cup \mu_{ich}$$

$$\mu_{oD} = \mu_{oD} \setminus \{\mu_{ich}\} \cup \{\mu_{od}\}_h$$

$$\mu_{oND} = \mu_{oND} \setminus \{\mu_{od}\}_h$$

PAS 4

Dacă $\mu_{oND} \neq \emptyset$, atunci salt la 3

Așa fel dacă $\mu_{oD} \neq \emptyset$ atunci salt la 3

altfel μ_{PT} finală

10. Dată exemple de sisteme cu arhitectură paralela

IBM's Blue Gene/P massively parallel supercomputer.

Blue Gene/L - eServer BlueGene Solution IBM

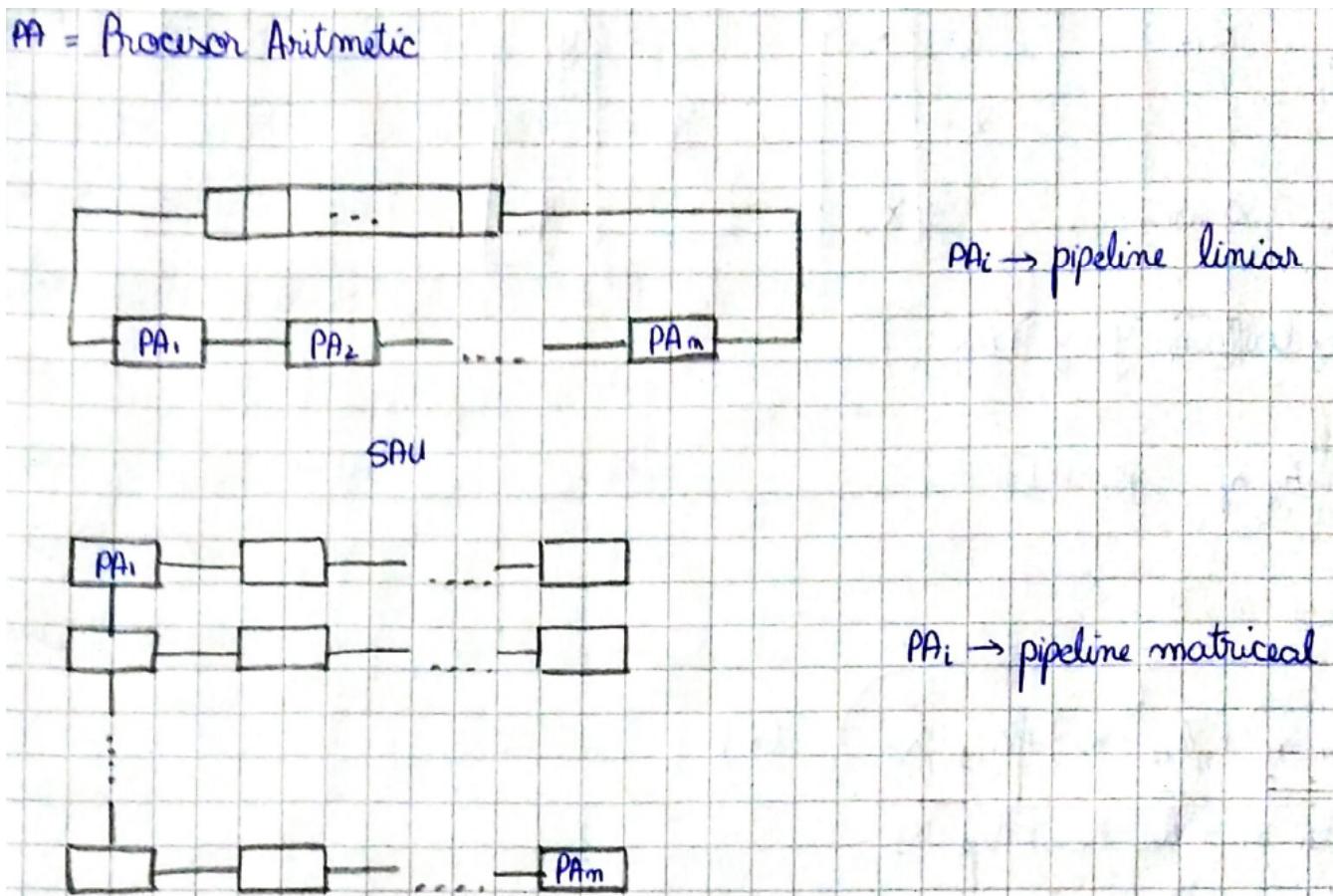
Roadrunner - BladeCenter Cluster IBM

Ranger - SunBlade - opteron quad 2Ghz infiniband- Sun Microsystems

Jaguar-Cray XT4 QuadCore 2.1Ghz - CrayInc.

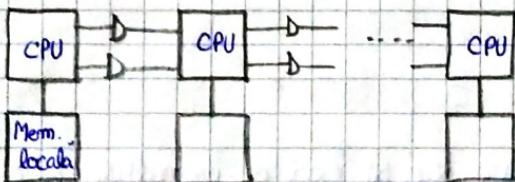
11. Modele sistolic (memorie partajată cu transfer de mesaje)

PA = Procesor Aritmetic



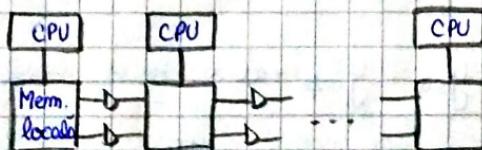
METODE DE COMUNICARE ÎN SISTEMELE SISTOLICE:

(1)



După prelucrare, punem într-un buffer, iar următorul năpâie să preia de acolo.

(2)

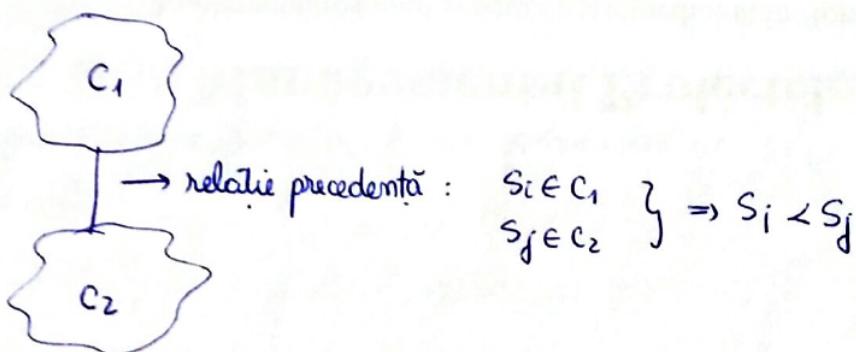


Bridușorul sărac nu poate rezultaști în memorie. În acest moment există un acces direct de memorie care mută datele de ieșire în altă zonă de memorie ca fiind date de intrare.

\Rightarrow necesară implementarea unui DMA în sistem

12. Concatenarea sistemelor de sarcini

Două sisteme de sarcini C_1 și C_2 pot fi concatenate ($C = (C_1, C_2)$), graful asociat obținându-se prin introducerea unei are de la nodul terminal al lui C_1 , la nodul initial al lui C_2 .



$$C = (C_1, C_2)$$

$$C_1 = (S_1, \alpha_1) \text{ unde } \alpha_1 = a_1, \dots, a_{2m} \text{ și } S_1 = \{s_{11}, \dots, s_{1m}\}$$

$$C_2 = (S_2, \alpha_2) \text{ unde } \alpha_2 = b_1, \dots, b_{2m} \text{ și } S_2 = \{s_{21}, \dots, s_{2m}\}$$

$$\Rightarrow \alpha = a_1, \dots, a_{2m}, b_1, \dots, b_{2m}$$

13. Sisteme echivalente

Două sisteme cu aceeași multime de sarcini sunt echivalente dacă sunt determinante și dacă pentru aceeași stare initială produc aceeași succență de valori.

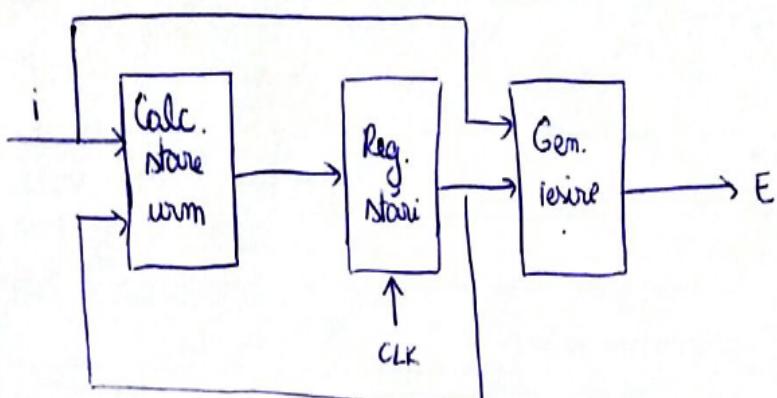
$$C' = (S, \alpha')$$

$$C = (S, \alpha)$$

14. Structura unei unități de comandă microprogramate

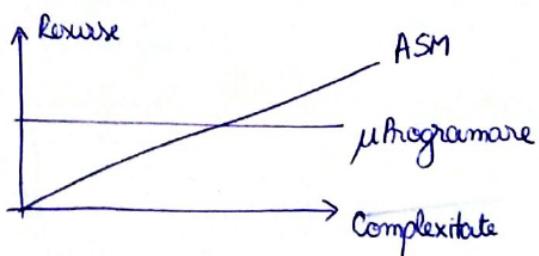
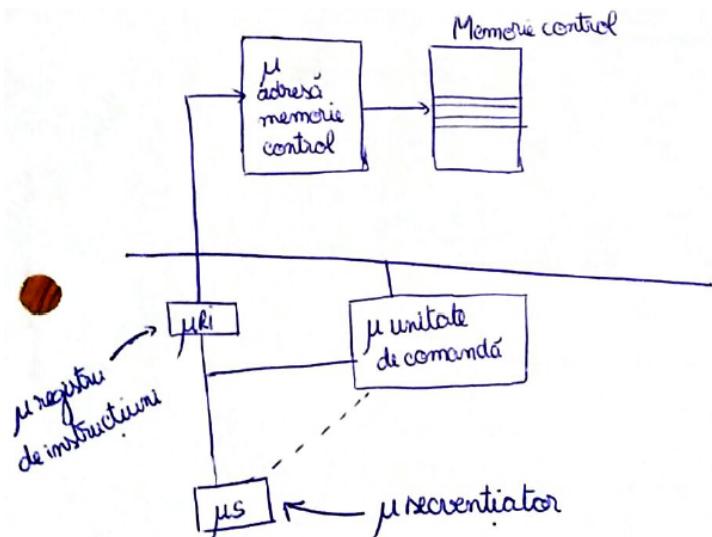
Puteți fi implementată în 2 moduri:

- * Algorithm State Machine



- * Microprogramare:

Stările sunt reprezentate ca celule de memorie, iar trecerea de la o stare la alta este făcută de o unitate mică de comandă.



15. Indicatori de performanță (prelucrari paralele)

- **RATA DE EXECUȚIE:** măsură rata de obținere a unor rezultate în unitatea de timp
 - Unități de măsură
 - MIPS (milioane instrucțiuni pe secundă)
 - ↖ inadecvat pt. o mașină SIMD care execuță aceeași instrucțiune pe mai multe fluxuri date
 - MOPS (milioane operații pe secundă)
 - ↖ nu ține seama de natura operațiilor sau lungimea curvenilor
 - MFLOPS (milioane de operații în virgula mobilă pe secundă)
 - ↖ adecvată numai pentru aplicații numerice
 - MLips (milioane de instrucțiuni logice pe secundă)
 - ↖ adecvată pentru aplicații de IA

• VITESĂ DE PRELUCRARE

$$V_p = \frac{T_1}{T_p}, \text{ unde } \begin{cases} T_1 = \text{timpul necesar pt. prelucrare cu 1 procesor} \\ T_p = \frac{\text{---}}{p \text{ procese}} \end{cases}$$

↑ raportul dintre prelucrarea sequentială și cea paralelă

rezultă în evidență creșterea în viteză datorită paralelismului

$$V_p \geq 1$$

$$\underline{\text{ideal}} : T_1 = p \cdot T_p$$

• EFICIENȚA : raportul dintre viteză de prelucrare și număr procesare

$$E_p = \frac{V_p}{p} = \frac{T_1}{p \cdot T_p} \leq 1$$

• REDUNDANȚA

$$R_p = \frac{Q_p}{Q_1} \text{ unde } Q_p = \text{nr. operații efectuate în timp}$$

↑ reflectă timpul pierdut cu overload-ul

• UTILIZARE

$$U_p = \frac{Q_p}{p \cdot T_p} \leq 1$$

↑ raportul dintre numărul de operații necesare efectuării calculului cu p procesare și numărul de operații care ar fi putut fi efectuate cu p procesare în timpul T_p

16. Diferenta intre sistemele microprogramate si microprogramabile

Sisteme microprogramate → modalitatea de implementare a uc, ~~fără~~ fără a oferi resurse hardware și suportul de programe pt. accesul utilizatorului la nivelul jprogramului

Sisteme microprogramabile → oferă atât resurse hardware, cât și facilitățile software pentru accesul utilizatorului la nivelul jinstructionii

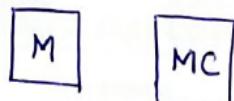
17. Modalitati de implementare a memoriei de control

Poate fi privită din 2 perspective

- relația dintre memoria de control (Mc) și memoria principală (M)
- structura Mc

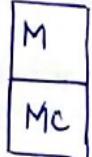
Perspectiva 1 : Relația dintre MC și M

- Mc separată de M, atât din punct de vedere logic, cât și fizic



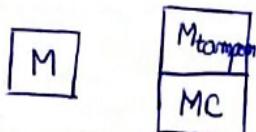
$$V_{MC} \gg V_M$$

- Mc implementată în același spațiu fizic și de adresare ca M



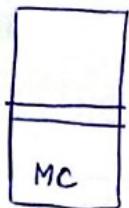
Memoria trebuie să fie suficient de rapidă ca să poată fi folosită ca Mc și deosebit de ieftină ca să fie folosită ca M.

- Mc este implementată separat de M, dar este încărcată din aceasta prin inter-mediu unei memorii tampon.



Perspectiva 2 : Structura MC

- MC cu μ-instrucțiuni pe curănt



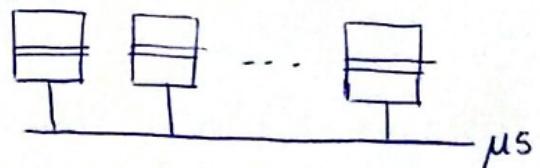
- un curănt din memorie $\Rightarrow \mu\text{-instructiune}$
- citire μ-instrucțiune \Rightarrow un singur acces la MC

- MC cu organizare pe pagini:

Pt. un grup de μ-instrucțiuni se face o singură citire, iar sevenicatorul se deplasează la fiecare din ele

Dezavantaje:

- microinstructiuni mai lungi
- organizarea în memorie de către compilator este mai grea



- MC cu organizare pe blocuri

Există 2 tipuri de adresa

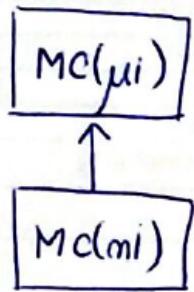
- adrese de μ-instrucțiuni din același bloc cu μ-instrucțiunea curentă
- adrese de blocuri

- micșorarea lungimii instrucțiunii
- timp suplimentar cu comutarea adreselor de blocuri

- MC divizată

- folosește 2 unități de memorie
 - memorie de μ-instrucțiuni (M_i)
⇒ păstrează toate μi distincte
 - memorie de adrese de μ-instrucțiuni (MA)
⇒ păstrează programul prin reținerea adreselor de μi
- lungime curănt $MA < \text{lungime curănt } M_i$
- pentru execuția unei μi sunt necesare 2 adresaři: una la MA și una la M_i

- MC structurată pe 2 niveluri
 - flexibilitate mare
 - μ_i instrucțiunea maximă este interpretată de un set de m_i din $MC(\mu_i)$. La rândul ei, fiecare μ_i este interpretată de o multime de m_i din $MC(m_i)$



18. Organizarea microoperatiilor

Trebuie să tinem cont de următoarele lucruri:

- gradul de paralelism pe care vrem să il atingem
- structura maximă de bază
- gradul de optimizare a lungimii curțințului de control

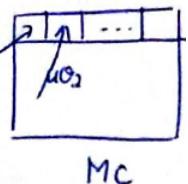
Există 6 tipuri de implementații:

• CONIFICARE VERTICALĂ / MAXIMALĂ

- fiecare microinstrucțiune specifică o microoperatie
- se rulează sequential
- curțant de control are lungimea minimă în acest caz: $[\log_2 (m+1)]$
 - \uparrow avantaj
- dezavantaje:
 - eliminarea controlului paralel asupra resurselor
 - lipsă de flexibilitate în introducerea de noi microoperatii

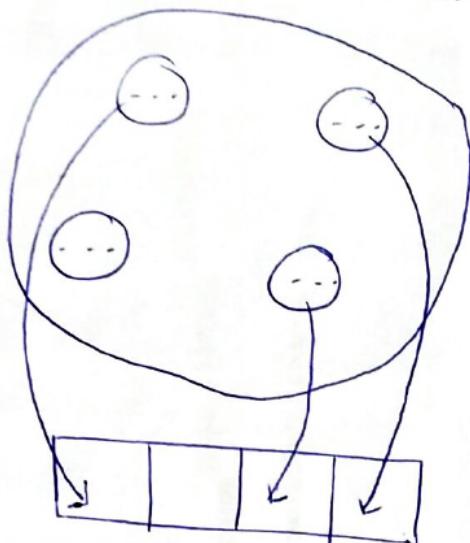
• CODIFICARE ORIZONTALĂ

- fiecare operatie este pusă în corespondență cu un bit din cadrul curantului de control
- avantaj: asigură paralelism maxim și are o flexibilitate mare
- dezavantaj: lungime mare a memoriei de control



• CODIFICARE MINIMALĂ / OPTIMĂ

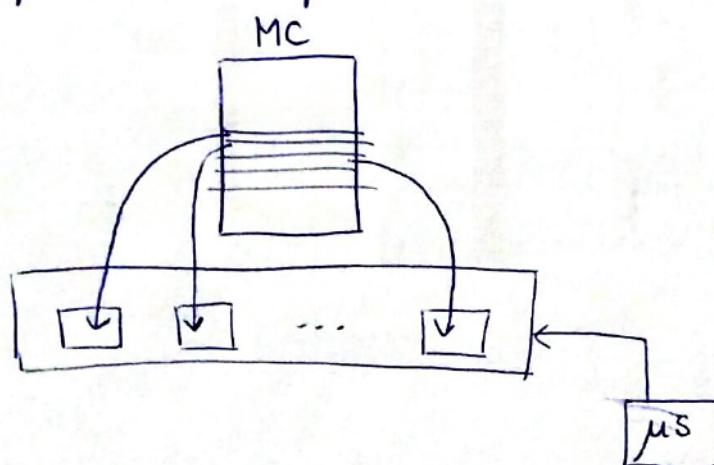
- combina flexibilitatea și paralelismul de la codificarea orizontală cu eficiența codificării verticale
- Nobilește operațiile care nu se pot executa în paralel.
- grupurile cu operațiile care se exclud reciproc \Rightarrow codificare verticală + le punem în același câmp
- grupurile cu operațiile care nu se exclud reciproc \Rightarrow codificare orizontală + le punem în compuri diferite



- Este necesar un decodificator pentru fiecare câmp

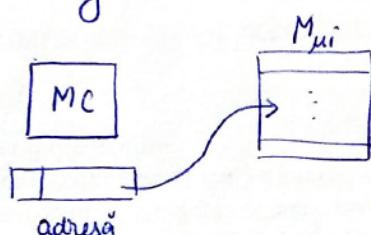
• CONFIICARE CU CONTROL RESIDUAL

- curîntul de control nu controlează direct resursele, ci prin intermediul reg. de control residual
- microinstructiunile pot să modifice valoarea uneia sau a mai multor registre
- avantaj: asigură o economie a MC atunci când se realizează aceeași operatie în mod repetat



• CONFIICARE CU CONTROL PRIN ADRESE

- formă simplificată a conceptului de nanoprogramare
- fiecare jui e memorată o singură dată
- dezavantaj: necesită 2 accese la memoria în cadrul unui ciclu de microinstructiune
- avantaj: economie de memorie



• CONFIICARE MIXTĂ

- microinstructiunea este împărțită în câmpuri
- putem avea în fiecare câmp
 - microoperării
 - câmpuri
 - control al câmpurilor
 - adresă

19. Legatura dintre algoritmi paraleli si arhitecturi paralele

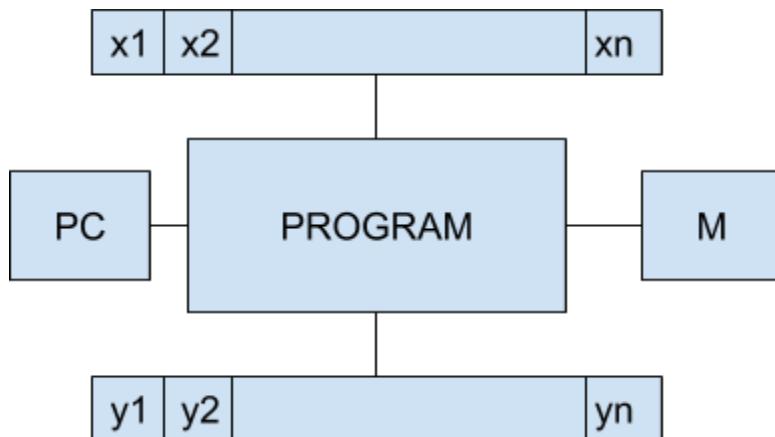
Algoritm paralel	Arhitectură paralelă
<ul style="list-style-type: none">• Granularitate de prelucrare Unde mă opresc? Job / Task / proces?	<ul style="list-style-type: none">• Complexitate procesor
<ul style="list-style-type: none">• Control concurrent Se referă la schema de selecție a modulilor pentru execuție care trebuie să satisfacă dependență de date	<ul style="list-style-type: none">• Mod de operare
<ul style="list-style-type: none">• Structurarea datelor Structuri date, mecanisme de acces la date, etc.	<ul style="list-style-type: none">• Structurarea memoriei
<ul style="list-style-type: none">• Geometria comunicatiei Se referă la sablonul de interacțiune între module	<ul style="list-style-type: none">• Retele de comutare Ex: CrossBar, Delta
<ul style="list-style-type: none">• Complexitate algoritm	<ul style="list-style-type: none">• Număr procesare, dimensiune memorie etc.

I. d

Modele de calcul paralel

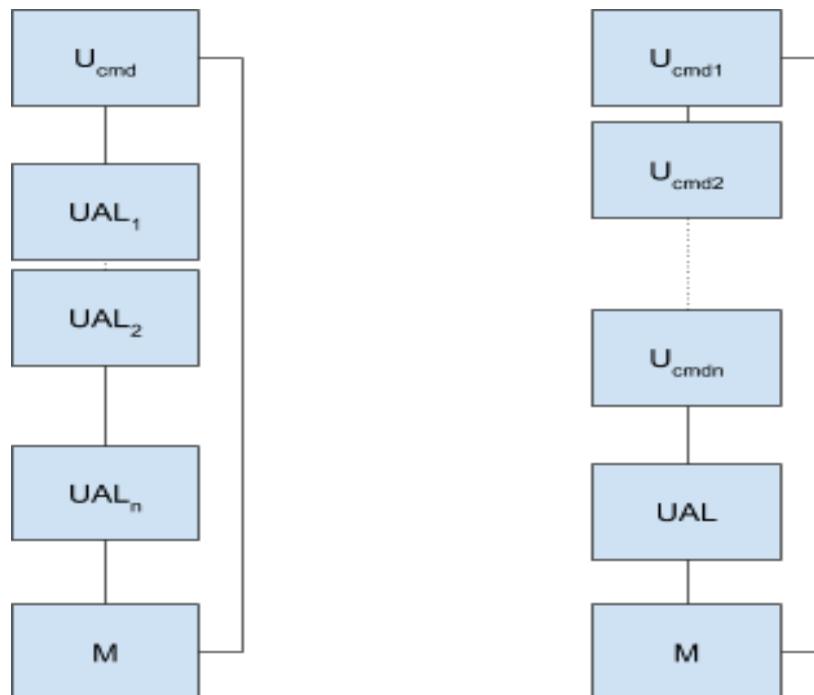
A. RAM (Random Access Machine)

- Masina de baza monoprocesor
- Similar cu modelul Turing
- Analiza complex algoritmilor si dezvoltarea lor
- Punctul de plecare pentru modelele paralele
- La fiecare moment de timp se executa cate o instructiune



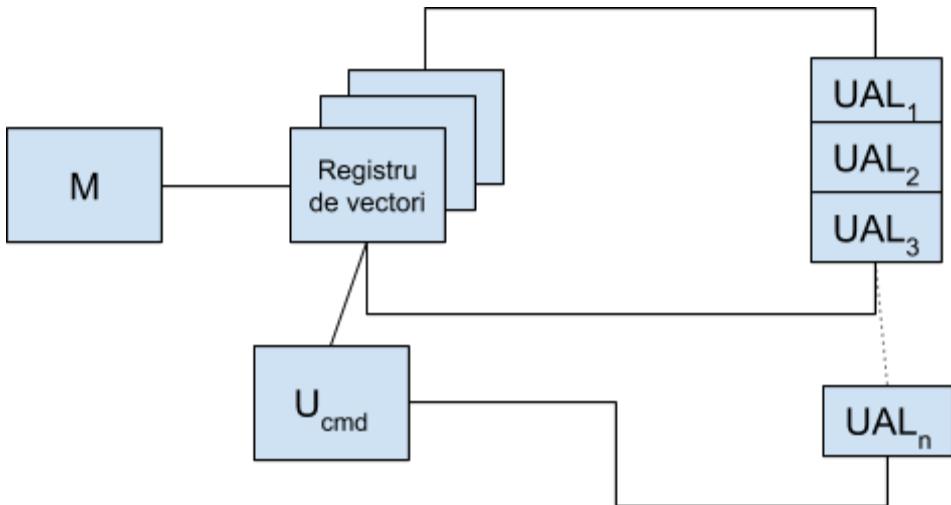
B. Pipeline

- La nivel unitate de comanda / la nivel unitate de prelucrare
- Imparte un task T in subtaskuri $T_1 \dots T_n$. Fiecare subtask e atribuit unui element de procesare.



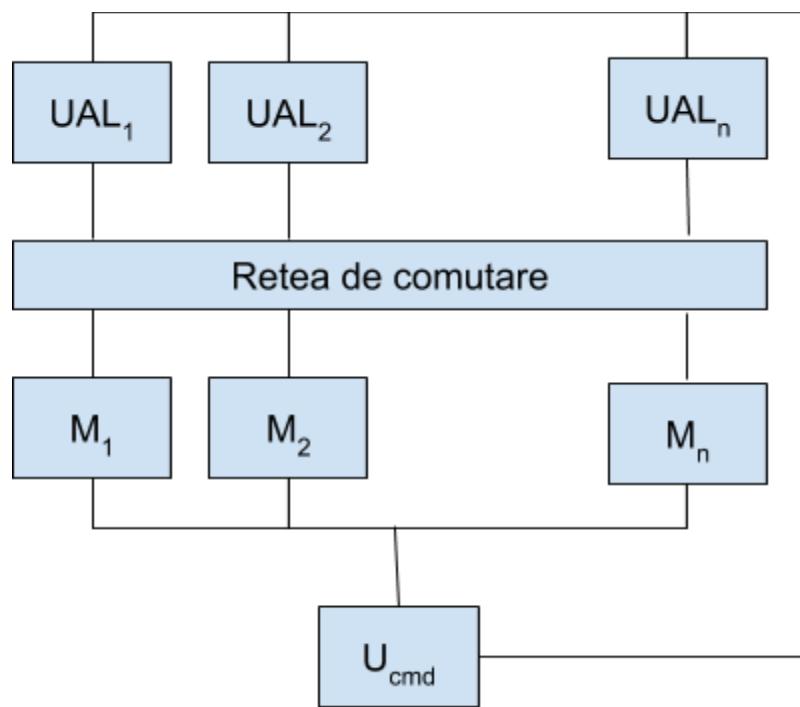
C. Procesare de vectori

- Set de instructiuni care trateaza vectorul ca un operand simplu
- Sisteme monoprocesor care au ca extensie procesarea de vectori
- SIMD



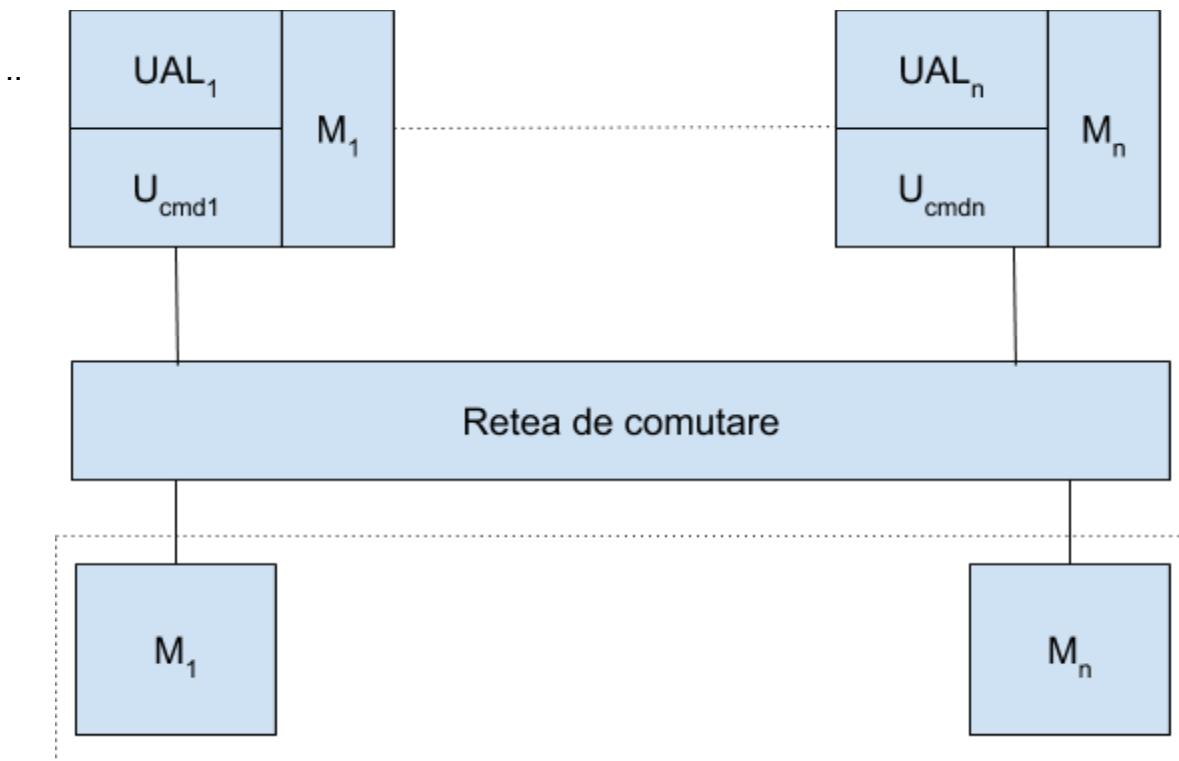
D. Procesare de "masive de date" (MPP)

- Calculatoare care constau din unitati aritmetice multiple supervizate de o singura unitate de control, care efectueaza aceleasi op la un mom dat
- Asemanator cu SIMD + UMA



E. Multiprocesoare cu memorie partajata

- Fiecare procesor are propriile UAL, Ucmd si M
- Comunicarea se face prin intermediul unei retele de comutare, cu module de memorie partajata
- Reteaua de comutare permite atat schimbul de informatii cat si accesul la fiecare modul al memoriei partajate
- **MIMD + NUMA (acces neuniform la mem. Partajată vs mem. locală)**

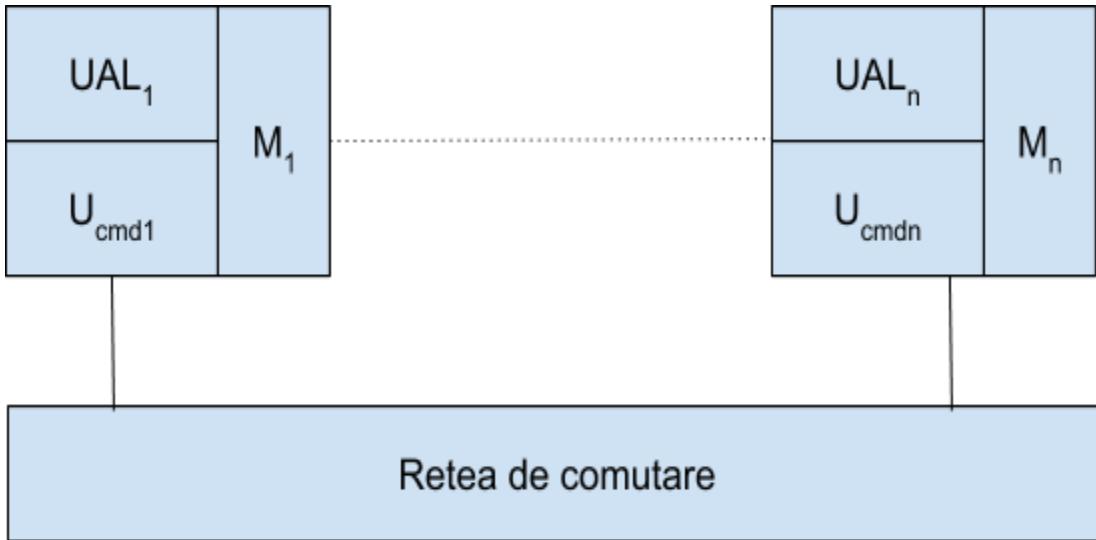


F. Modelul PRAM (Parallel Random Access Machine)

- Programele sunt diferite, comunicarea intre P si M este considerata ca se face sincron, dar secvential daca se face la acelasi submodul de memorie
- Memoria comună e disponibila tuturor proceselor
- 4 posib de acces la memorie: EREW, ERCW, CREW, CRCW

G. Multiprocesor cu transfer de mesaje

- Fiecare procesor dispune de elemente de procesare
- Interactiunea se face prin mesaje, nu acces direct



H. Procesarea sistolica

- Elemente de procesare identice, asezate in pipeline liniar/matriceal
- Utilizat pentru operații pe vectori, matrice
- Comunicarea între PA se face: la nivel de CPU (comunicare directă între CPU-uri) sau prin DMA (comunicare indirectă între CPU-uri, prin DMA)

I. Modelul data flow

- Specifica efectul unor operatii imediat ce operanzii sunt disponibili
-> elimina necesitatea existentei contorului de program
- pentru a realiza aceasta, fiecarui operand i se asociaza un token
prin care se specifica daca e disponibil pentru prelucrare
- Nu necesita adrese pt memorare
- Nu se execută instr., ci se asociază fiecărei date o stare de disponibilitate -> execuție asincronă ce începe când datele sunt disponibile
- Viteză ridicată & potențial ridicat de paralelism
- Ușor de integrat dpdv al sincronizării & comunicării
- Pot exista timpi mari de așteptare, overhead mare de control, iar prelucrarea structurilor de date (vectori, matrice etc) e foarte dificilă

II. Subiecte examen

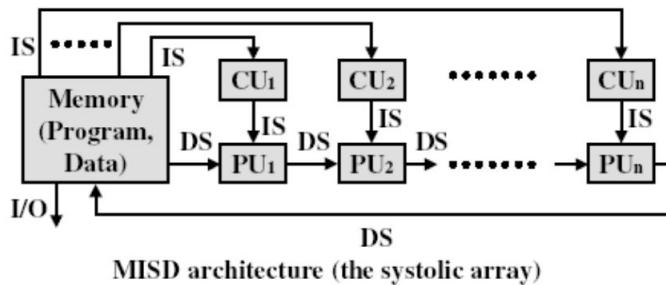
- 1. Modele de calcul paralel. Comparatie data flow-transmitere de mesaje(3)**
 - La data flow nu sunt necesare adrese pentru memorare, la transmitere de mesaje fiecare procesor are elementele proprii de procesare(UAL, U_{cmd}, M)
 - La data flow comunicatia si sincronizarea sunt usor de integrat, in schimb la transmiterea de mesaje au un cost ridicat din cauza timpului in plus necesar trimiterii de mesaje
 - Ambele au un potential ridicat de paralelism

- 2. Modele de calcul paralel. Comparatie intre modelul cu memorie partajata si modelul sistolic.**
 - La modelul sistolic unitatile de procesare sunt asezate in pipeline(liniar sau matriceal), in schimb la cel cu memorie partajata fiecare proces are propriile unitati de procesare(UAL, U_{cmd},M) si de asemenea au si o memorie comună
 - Timpul de acces la memoria comună in cazul memoriei partajate este mai mare deoarece mai multe procese pot incerca simultan sa o acceseze, dar doar unul poate intra intr-un ciclu de ceas. La pipeline este impartita in mai multe module, astfel se pot accesa mai multe module simultan.
 - Sistolic - MISD, partajata: MIMD

MISD (Multiple Instruction Streams Over A Single Data Streams)

❖ MISD

- ✓ Processor arrays, systolic arrays
- ✓ Special purpose computations



MISD architecture (the systolic array)

3. Limita inferioara (4) - cred ca asta vrea... dar este extrem de scris...

LI = limita inferioara de nr. de niveluri pe care pot organiza graful

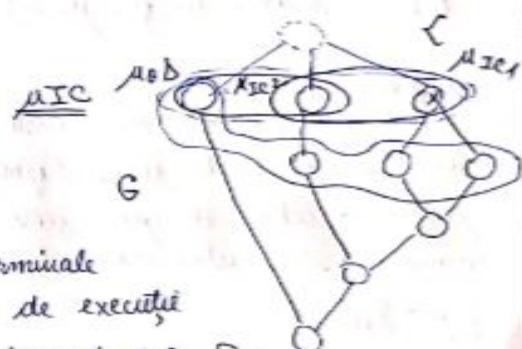
I $\rightarrow h = \text{subtirea grafului } G$

$$\underline{II} \rightarrow g = \max_i \left\{ \max_j \left\{ |SGM_{ij}| + \min_k (\|\mu_{OTjk}\|) \right\} \right\};$$

$1 \leq i \leq |\mu_{OT}|$ nr. de msp. terminale

$1 \leq j \leq |\mathcal{P}_i|$ nr. de elem. de execuție

$1 \leq k \leq |G_i|$ grafurile cu elem. de exec. \mathcal{P}_i



Pentru fiecare \mathcal{P}_i se definește p_i ,

$$p_i = \max_j \left\{ \max \left(\frac{|SGM_{ij}|}{|\mathcal{P}_i|}, h(SGM_{ij}) + \|\mu_{OT}\|_j \right) \right\}$$

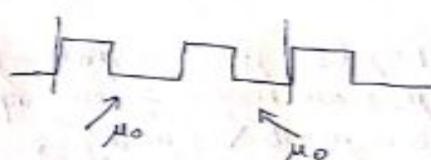
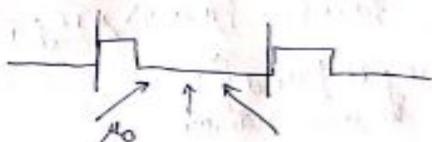
$$g = \max_i (p_i)$$

III $\rightarrow \mathcal{P}_i$ ne sunt distințe

$$\mu = \max_i \left(\frac{(\mu_B(\mu_0) \mathcal{P}_i)}{|\mathcal{P}_i|} \right)$$

$$\mu = \max_i \left(\mu_B(\mu_0) \mathcal{P}_i \right),$$

$$1 \leq i \leq |\mathcal{P}_i|$$



$\overline{N} \rightarrow \overline{Z} = \text{nr. de cicluri ai instr.} \times \text{nr. de cicluri pt m op.}$

$$Z = n_i \cdot |\mu_{0m_i}|$$

$L_I = \max(h, g, \mu, Z)$: nr. minim de instr. posibil al unui blocul de op: $m \cdot \mu I$

4. Gustafson

Gustafson

w - workflow

α - procent sequential

$1-\alpha$ - procent paralel

$$w' = \alpha w + (1-\alpha) * n * w$$

$$T_1 = \alpha w + (1-\alpha) * n * w$$

$$T_n = \alpha w + (1-\alpha) * w$$

$$S_n = \frac{\alpha w + (1-\alpha) * n * w}{\alpha w + (1-\alpha) * w}$$

4'. Sun-Ni

Sun-Ni's Law (or Sun and Ni's Law), is a memory-bounded speedup model which states that as computing power increases the corresponding increase in problem size is constrained by the system's memory capacity.

w^* = the scaled workload under a memory space constraint

w - workload-ul normal

α - procent sequential

$G(m)$ - the function that reflects the parallel workload increase factor as the memory capacity increases m times

$$w^* = \alpha w + (1-\alpha) G(n) w$$

$$S^{**} = \frac{\alpha w + (1-\alpha) G(n) w}{\alpha w + \frac{(1-\alpha) G(n) w}{n}} = \frac{\alpha + (1-\alpha) \frac{G(n)}{n}}{\alpha + \frac{(1-\alpha) G(n)}{n}}$$

Pentru subiecte de comparatie: Amdahl's law states that the sequential portion of the problem (algorithm) limits the total speedup that can be achieved as system resources increase. Gustafson's law suggests that it is beneficial to build a large-scale parallel system as the speedup can grow linearly with the system size if the problem size is scaled up to maintain a fixed execution time.^[4] Yet as memory access latency often becomes the dominant factor in an application's execution time,^[5] applications may not scale up to meet the time bound constraint.^{[1][2]} Sun-Ni's Law, instead of constraining the problem size by time, constrains the problem by the memory capacity of the system, or in other words bounds based on memory. Sun-Ni's Law is a generalization of Amdahl's Law and Gustafson's Law. When the memory-bounded function $G(M)=1$, it resolves to Amdahl's law, when the memory-bounded function $G(M)=m$, the number of processors, it resolves to Gustafson's Law.

5. Amdahl vs Worlton

Amdahl - 2k18: Formula prin care se stabileste o limita a cresterii de viteza in structurile paralele in raport cu cele secentiale

Worlton - tcare aproximeaza operarea unui multiprocesor

Worlton

- t_s - timpul de sincronizare
- t_o - timpul de overhead
- t - timpul mediu de executie al unui task
- p - nr de procesoare
- N - nr de taskuri
- T_1 - $N*t$
- T_{Np} - $t_s + (N/p)*(t+t_o)$

$$VNp = \frac{T_1}{T_{Np}} = \frac{1}{\frac{ts}{Nt} + \frac{1}{N} * [\frac{N}{p}] (1 + \frac{t_o}{t})}$$

$\frac{ts}{Nt}$ - reducerea timpului de sincronizare + marirea timpului intre sincronizari (efectul sincronizarii)

$\frac{N}{p}$ - cresterea numarului de procesoare, nr de taskuri multiplul nr de procesoare

$\frac{t_o}{t}$ - cresterea lui t (granularitatea taskurilor) - efectul de overhead

6. Legea lui Amdahl

R_H - rata de executie pe p procesoare

R_L - rata de executie pe 1 procesor

f - procentul de paralelism

$1-f$ - procentul de secentialitate

$$R(f) = \frac{1}{\frac{f}{R_h} + \frac{1-f}{R_l}}$$

$$R(f) = \frac{1}{(1-f) + \frac{f}{N}} \quad N - \text{nr de procesoare.}$$

- 7. Compatibilitate si incompatibilitate(2)** - sunt si in cursurile de andrei.cisco - dar trebuie descifrate(cursul 9 Pag 1), dar o sa o fac mai tarziu - acolo imi par foarte vag enuntate.

Def. 1

Fie $mPt=\{mIC_1, mIC_2, \dots, mIC_{[mPt]}\}$ partitia unui microsubbloc in microinstructiuni complete si $MB(mO)=\{mO_1, mO_2, \dots, mO_{[MB]}\}$ setul de microoperatii distincte din cadrul microsubblocului.

Două microoperatii mO_i și mO_j sunt **compatibile** dacă pentru orice k , $1 \leq k \leq [mPT]$, dacă $mO_i \in mIC_k$ atunci $mO_j \notin mIC_k$.

Compatibilitatea între două microoperatii trebuie privită în sensul că cele două microoperatii nu sunt specificate (nu sunt active) niciodată împreună în cadrul unei microinstructiuni din microbloc. Controlul resurselor sistemului micropogramat nu necesită niciodată efectuarea în paralel a celor două microoperatii.

Def. 2

Două microoperatii $mO_i \in MB(mO)$, $mO_j \in MB(mO)$ sunt **incompatibile** dacă există cel puțin o microinstructiune completă mIC astfel încât $mO_i \in IC_k$ și $mO_j \in mIC_k$.

- 8. Clase de compatibilitate, incompatibilitate - aici nu gasesc incompatibilitate - la fel ca mai sus se gasesc in cursul 9**

O clasă de compatibilitate $CC(mO)$ este un set (subset) al mulțimii $MB(mO)$ în care oricare două microoperatii sunt compatibile între ele.

$$CC(mO)=\{mO \mid \text{pt orice } mO_i, mO_j \in CC(mO) \text{ avem } mO_i \text{ compatibilă cu } mO_j\}$$

- O

singura definitie... nici nu ai ce scrie la ea

- 9. Costul clasei de compatibilitate - la fel se gaseste ca mai sus si asta se gaseste in cursul 9 pag1(jos)-2 - o sa o scriu mai tarziu si pe asta din cursul 9**

Costul de implementare a unei clase de compatibilitate (măsurat în numărul de biți necesari pentru codificare) este dat de implementarea codificării verticale a microoperațiilor ce compun clasa.

$$\text{Cost } CC_i = \lceil \log_2(|CC_i|+1) \rceil$$

iar costul total de implementare al cuvântului de control

$$\text{Cost } CC = S \sum_{i=1}^k \lceil \log_2(|CC_i|+1) \rceil$$

unde k este numărul de clase de compatibilitate.

10. Sisteme microprogramate si microprogramabile(3)

Sistemele microprogramate se referă la modalitatea de implementare a unității de comandă fără a oferi resursele hardware și suportul de programe pentru accesul utilizatorului la nivelul programului.

Sistemele microprogramabile oferă atât resursele hardware ca și facilitățile software pentru accesul la nivelul microinstrucțiunilor

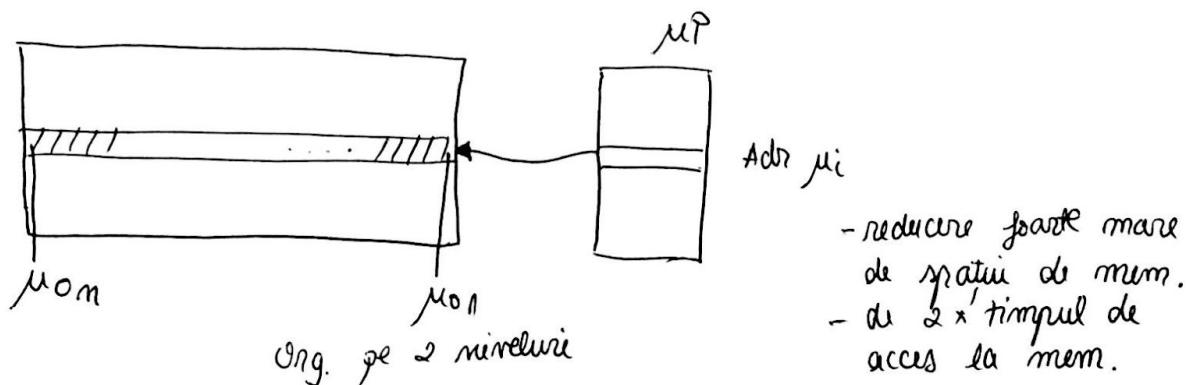
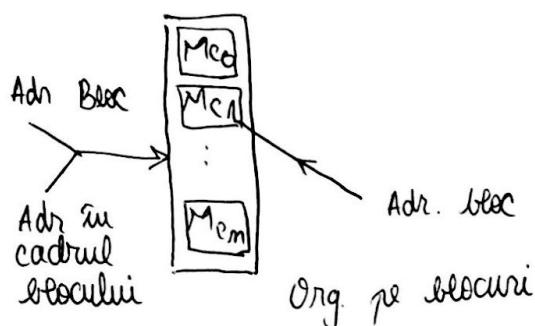
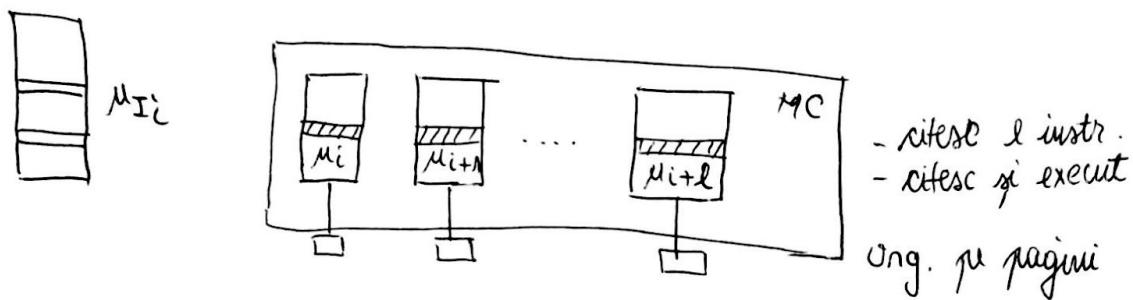
11. Organizarea memoriei de control

Organizarea memoriei de control în funcție de membru principală:

1. M și Mc sunt entități separate
2. M și Mc sunt aceeași entitate
3. M și Mc sunt entități separate, dar sunt încarcate în aceeași entitate

Organizarea memoriei de control:

1. Pagini
2. Blocuri
3. 2 nivele



12. Algoritmi de partitionare - vezi urmatoarele 2 exercitii

13. Algoritmul de partitionare (in 8 pasi)

Alg. de partitie optimă

- ① $\mu_{PT} = \emptyset$; $\mu_{PTA} = \mu_B(\mu_0)$; se consideră că se exec. secvențial
 $\mu_{OD} = \{\mu_0\} \cup \mu_{OI} \in \mu_B(\mu_0) \Rightarrow \exists \mu_{OI} < \mu_0\}$
 $\mu_{OND} = \mu_B(\mu_0) \setminus \mu_{OD} = \{\mu_0 | \mu_0 \in \mu_B(\mu_0) \setminus \mu_{OD}\}$
- ② Dacă $\mu_{OND} = \emptyset$ și $|\mu_{OD}| \leq 3 \Rightarrow ⑤$
- ③ Se generează $\{\mu_{IC}\} = \{\mu_{IC_j}, \mu_{IC_{j+1}}, \dots, \mu_{IC_k}\}$
Dacă $j \neq k$, atunci salvează CONTEXT: $\mu_{PT,j}^{**} = \mu_{PT}$; $\mu_{OD,j} = \mu_{OD}$;
 $\{\mu_{IC}\}_j = \{\mu_{IC_j}, \dots, \mu_{IC_k}\}$
- ④ $\mu_{PT} = \mu_{PT} \cup \mu_{IC_j}$; $\mu_{OD} = \mu_{OD} \setminus \mu_{IC_j} \cup \{\mu_{OD}\}^{**}$
obs: μ_{OD} nu conține pe cele dependente de μ_{OD} anterior
 $\mu_{OND} = \mu_{OND} \setminus \{\mu_{OD}\}^{**}$
Dacă $|\mu_{OD}| \neq 0$ și $|\mu_{PT}| \leq |\mu_{PT}|_{j+1}$, atunci $\Rightarrow ②$
- ⑤ Dacă $\mu_{OD} = \emptyset$, atunci $\Rightarrow ⑦$

- ⑥ Se generează μ_{IC} din μ_{OD} : $\mu_{PT} = \mu_{PT} \cup \mu_{IC}$; $\mu_{OD} = \mu_{OD} \setminus \mu_{IC}$
 dacă $\mu_{OD} \neq 0$ și $|\mu_{PT}| < |\mu_{PTA_j}| - 1 \Rightarrow$ ④
 altfel, \Rightarrow ②
- ⑦ dacă $|\mu_{PT}| < |\mu_{PTA}|$, atunci $\mu_{PTA} = \mu_{PT}$.
 altfel, dacă $|\mu_{PTA}| \leq m \cdot \mu_I$ $\Rightarrow \mu_{PTA}$ este optimă $\Rightarrow \underline{\text{STOP}}$
- ⑧ dacă $\{\mu_{IC_j}\} \neq \emptyset$, atunci refac CONTEXT: $\mu_{PT} = \mu_{PT_j}$; $\mu_{OD} = \mu_{OD_j}$; $j = j + 1$; $\mu_{IC} = \mu_{IC_j}$; $\mu_{OND} = \{\mu_0 | \mu_0 \in \mu_{OND} \setminus (\mu_{PT} \cup \mu_{OD})\}$
 \Rightarrow ④
 altfel, μ_{PTA} este optimă $\Rightarrow \underline{\text{STOP}}$

14. Algoritm de partitionare a UI folosind uO - asta pare cea mai aproape ca foloseste microoperatii

Algoritmul evolutiv

1. Pas 1. Initializare

$$\left\{ \begin{array}{l} \mu_{PT} \neq \emptyset, \mu_{OD} = \{\mu_0 | \mu_0 \text{ este } \neq \mu_1 \dots \mu_n\} \\ \mu_{OD} = \mu_B(\mu_0) \setminus \mu_{OD} \end{array} \right.$$

$p_{succ}(\mu_0)$ = nr de succesiuni cu operatorii respectiv

$$p_{succ}(\mu_{jC}) = \sum_{\mu_0 \in \mu_{jC}} p_{succ}(\mu_0)$$

2) doară μ_{OD} atențional pe s

3) generază setul $\{\mu_{jC}\} = \{\mu_{jC_1}, \mu_{jC_2}, \dots, \mu_{jC_{k-1}}$
desc.

- considerăm prioritățile acordată multimei după ^{-49%}
numărul de succesiuni cu μ_{jC}

* $p_{succ}(\mu_{jC_j}) \geq p_{succ}(\mu_{jC_{j+1}}) \geq \dots \geq p_{succ}(\mu_{jC_k})$

$$\mu_{PT} = \mu_{PT} \cup \mu_{jC_j}$$

$$\mu_{OD} = \mu_{OD} \setminus \mu_{jC_j} \setminus \{\mu_{od}\}_j$$

$$\mu_{OD} = \mu_{OD} \setminus \{\mu_{od}\}_j$$

salt 2

4. Generază $\{\mu_{jC}\} = \{\mu_{jC_1}, \dots, \mu_{jC_k}\}$ în
ordinea după un criteriu

$$\mu_{PT} = \mu_{PT} \cup \mu_{jC_k}$$

$$\mu_{OD} = \mu_{OD} \setminus \mu_{jC_k}$$

// salt pe s - comentat

doară $|\mu_{OD}| \neq 0$ atunci salt 3

altfel μ_{PT} este cea aproape optimă.

15. Teorema de suficientă(5)

Teorema de suficientă

Sistemul de sarcini format din sarcini mutual neinterferente este determinat.

***** daca cere si:

Teorema de necesitate:

Fie dat un sistem de sarcini neinterpretat, dar cu $DS \neq \emptyset$, sistemul C este determinat pt \forall interpretare a sarcinilor sale \Leftrightarrow sarcinile sunt neinterferente

Sisteme echivalente de sarcini:

Doua sisteme sunt echivalente \Leftrightarrow sunt determinate si produc aceleasi sechete de valori pentru o stare initiala data

=

T. suficientă

Sarcini neinterferente $\Rightarrow C = (\mathcal{S}, \alpha)$ sunt determinante

1. $\mathcal{S} = \{s\}$ o singură sarcină $\Rightarrow C$ este det.

2. P.d. adică $\mathcal{S}' = (s_1, \dots, s_{m-1}) \quad \{ \quad C' = (\mathcal{S}', \alpha') \text{ este det.}$

3. Dăm să $C = (\mathcal{S}, \alpha)$, unde $\mathcal{S} = \{s_1, \dots, s_m\}$ este det.

C' există $\alpha'_1 \neq \alpha'_2 \quad V(M_i, \alpha'_1) = V(M_i, \alpha'_2), \alpha'_1 = s_1, \dots, n$

$C = (\mathcal{S}, \alpha) \quad S \quad \alpha_1 = \alpha'_1 \overline{S} \quad \alpha_2 = \alpha'_2 \overline{S}$

DS are acelasi valori $\alpha'_1 \neq \alpha'_2 \quad M_i \in DS \quad f_S : \mathcal{S} \rightarrow RS$

$M_f \in RS$

$M_1 \notin RS$

$$\begin{aligned} V(M_i, \alpha_1) &= V(M_i, \alpha'_1 \overline{S}) = \\ &= V(M_i, \alpha'_1) = \\ &= V(M_i, \alpha'_2) = \\ &= V(M_i, \alpha_2 \overline{S}) = \\ &= V(M_i, \alpha_2) \\ \Rightarrow V(M_i, \alpha_1) &= V(M_i, \alpha_2) \end{aligned}$$

$$\left| \begin{array}{l} M_i \in RS \\ V(M_i, \alpha_1) = V(M_i, \alpha'_1 \overline{S}) \\ = V(M_i, \alpha'_1) \approx \\ = V(M_i, \alpha'_2) \approx \\ = V(M_i, \alpha'_2 \overline{S}) \\ = V(M_i, \alpha_2) \\ \Rightarrow V(M_i, \alpha_1) = V(M_i, \alpha_2) \end{array} \right.$$

16. Indicatori paraleli(3)

- a. Rata de executie - producerea unui rezultat in unitatea de timp

- i. MIPS - milioane de instructiuni/sec
- ii. MOPS - milioane de operatii/sec
- iii. MFLOPS - milioane de op in virgula mobila/sec
- iv. LIPS(MLIPS, GLIPS) - milioane de inferente logice/sec
- b. Viteza de prelucrare
 $V_p = T_1/T_p >> 1$ (t 1 proc/ t p proc) -> raport intre prelucrarea paralela si cea secventiala
 $p*T_p >> T_1 \rightarrow$ se consuma timp cu sincronizarea, comunicarea, overhead creat de interactiunea intre procese
Caz ideal $p*T_p = T_1$
- c. Eficienta de prelucrare
 $E_p = V_p/p = T_1/(p*T_p) < 1$
- d. Redundanta
 $R_p = O_p/O_1$ (nr op p proc/nr op 1 proc) -> t pierdut de overhead
- e. Utilizarea
 $U_p = O_p/(p*T_p) < 1$

17. Sistem de sarcini determinat. Secvente de executie. Secventa partiala de executie(2)

- a. Un sistem de sarcini determinat este acel sistem care se executa in paralel si care coopereaza intre sarcini pentru implementarea functiilor, care produce acelasi rezultat indiferent de rata de executie a unei sarcini independente si indiferent de ordinea in care se executa sarcinile paralele.
- b. Secvente de executie
 $\alpha = \alpha_1 \alpha_2 \dots \alpha_{2n}$ $\alpha_i = S^-_i$ sau $\alpha_i = S^-_i$
 S^- sau S^- apar o data in α
 1. $a_j = S^-_i$ $a_k = S^-_j \Rightarrow j < k$ (initierea e intotdeauna inaintea terminarii)
 2. $a_l = S^-_m$ $a_p = S^-_o$ $S_m < S_o \Rightarrow l < p$

Mai multe secvente de executie, o alegem pe cea cu cele mai putine niveluri

O secvență de execuție a unui sistem de n procese, $C = (P, <)$ este orice sir $\alpha = a_1 a_2 \dots a_{2n}$ de evenimente de inițiere și terminare a proceselor cu respectarea relațiilor de precedență impuse de $<$.

c. Secvențe parțiale

$$\alpha_p = a_1 a_2 \dots a_k \quad k < 2n$$

α_p - secvența parțială \rightarrow indică ce sarcini sunt active la un moment dat
 S_i este activă dacă există a_j , $a_j = S_i^+$, $j \leq k$ și nu există $a_l = S_l^-$, $l \leq k$

O secvență de execuție parțială este orice prefix al unei secvențe de execuție.

18. Sistem de sarcini determinat

Un sistem format din procese care se execută în paralel și cooperează pentru realizarea unor operații logice și de calcul, și produce același rezultat indiferent de durata de execuție a fiecărui proces independent, sau de ordinea în care acestea se execută, formează un sistem de procese funcțional, totaechival asincron (independent de viteza de execuție) sau sistem determinat.

Un sistem de procese este *nedeterminat* dacă rezultatele produse de procese independente depind de ordinea în care acestea se execută.

În Figura 5.2 se arată un exemplu de sistem nedeterminat:

P₁: R₁ ← BUSFN (M; DCD (ADR))

P₂: M * DCD (ADR) ← R₂

P₁ - citește din locația de memorie de la adresa ADR;
P₂ - scrie în locația de memorie de la adresa ADR.

Nedeterminarea se poate rezolva introducând o relație de precedență adecvată între procesele care erau independente.

Pentru a stabili condițiile necesare și suficiente ca un sistem să fie determinat, vom considera un model simplificat în care sistemul fizic este privit ca o mulțime ordonată de locații de memorie:

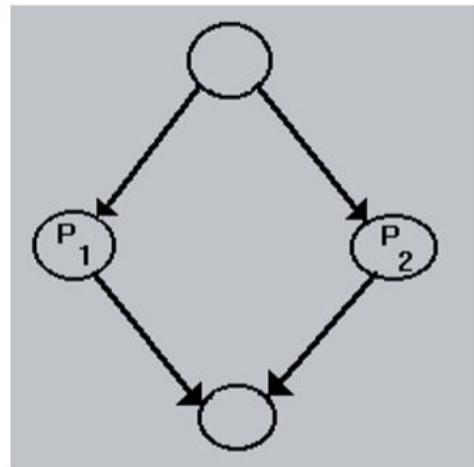


Figura 5.2. Exemplu de sistem nedeterminat

$$\mathbf{M} = (M_1, M_2, \dots, M_m)$$

ce conțin orice valoare dintr-o mulțime de valori V .

Stările sistemului vor fi definite de valorile care se găsesc în memorie la un moment dat:

De exemplu dacă:

$$\alpha = a_1 a_2 \dots a_{2n} \text{ și}$$

$$\sigma = s_0 s_1 \dots s_{2n}$$

reprezintă o secvență de execuție, respectiv secvența de stări corespunzătoare, iar $M_i[k]$ reprezintă valoarea din celula M_i imediat după evenimentul a_k ; starea sistemului va fi:

$$s_k = [M_1[k], M_2[k], \dots, M_m[k]]$$

Mulțimea tuturor stărilor poate fi definită astfel:

$$\Sigma = V^m \quad V^m = [V \times V \times \dots \times V], \text{ produs cartezian.}$$

Pentru a formaliza efectul execuției unui proces asupra celulelor de memorie vom considera că fiecărui proces P i se asociază funcția:

$$f_p : V^d \longrightarrow V^r$$

unde:

$$\begin{array}{ll} d = & |D_p| \quad \text{cardinalul domeniului valorilor de intrare, } D_p; \\ r = & |R_p| \quad \text{cardinalul domeniului valorilor de ieșire, } R_p. \end{array}$$

Act
Go t

Pentru o stare inițială s_0 și o secvență de execuție α , secvența corespunzătoare de stări:

$\sigma = s_0 s_1 \dots s_{2n}$ este definită în felul următor :

$$\begin{array}{ll} \text{Fie } D_p = (M_{x1}, M_{x2}, \dots, M_{xd}) & \text{domeniul valorilor de intrare;} \\ R_p = (M_{y1}, M_{y2}, \dots, M_{yr}) & \text{domeniul valorilor de ieșire;} \end{array}$$

1°. dacă $a_{k+1} = \bar{P}$, atunci $M_i[k+1] = M_i[k], \quad 1 \leq i \leq m;$

2°. dacă $a_{k+1} = P$, și $a_l = \bar{P}, \quad 1 \leq l \leq k$, atunci stările locațiilor de memorie din domeniul de valori al lui P la momentul $k+1$ sunt:

$$[M_{y1}[k+1], M_{y2}[k+1], \dots, M_{yr}[k+1]] = f_p[M_{x1}[l], M_{x2}[l], \dots, M_{xd}[l]]$$

și

$$M_i[k+1] = M_i[k] \quad \text{pentru } (\forall) M_i \notin R_p$$

Altfel spus, dacă a_{k+1} este un eveniment de inițiere nu apare o schimbare a stării, adică $s_{k+1} = s_k$.

Dacă însă a_{k+1} este un eveniment de terminare a lui P, $s_{k+1} \neq s_k$ doar în domeniul R_s , noile valori din R_s fiind determinate de f_p pe baza valorilor din domeniul de definiție din momentul imediat precedent inițierii lui P.

Secvența de stări $\sigma = s_0s_1.....s_{2n}$ ce rezultă dintr-o secvență de execuție, poate fi reprezentată sub forma unui tablou de dimensiuni $m * (2n+1)$ având pe linii celulele de *memorie M*, iar pe coloane stările.

19. Organizarea microoperatiilor. Avantaje + dezavantaje.

- a. Organizarea verticală
 - i. Avantaj: codificare minima (dimensiune)
 - ii. Dezavantaj:
 - 1. eliminarea controlului paralel asupra resurselor
 - 2. inflexibilitatea dezvoltării sau completării sistemului în ceea ce privește introducerea de noi microoperații.
- b. Organizare orizontală
 - i. Avantaj: paralelism maxim + flexibilitate mare
 - ii. Dimensiune prea mare
- c. Codificare minimală
 - i. Combina flexibilitatea și paralelismul potențial oferite de codificarea orizontală cu eficiența codificării verticale
- d. Codificare minimală pe 2 niveluri
 - i. Mai puțin eficientă decât cea minimală
- e. Codificare cu control rezidual
 - i. asigură o economie de memorie de control atunci când unele primitive funcționale realizează aceeași operație în mod repetat sau când un set de microoperații este activ o perioadă mare de timp, iar alte seturi de microoperații se modifică.
- f. Codificarea cu control prin adrese
 - i. Dezavantaj: necesită două accese la memorie în cadrul unui ciclu de microinstrucțiune
 - ii. Avantaj: se realizează o economie importantă de memorie.

20. Exemple de calculatoare parallele

IBM's Blue Gene/P massively parallel supercomputer.

Blue Gene/L - eServer BlueGene Solution IBM

Roadrunner - BladeCenter Cluster IBM

Ranger - SunBlade - opteron quad 2Ghz infiniband- Sun Microsystems

Jaguar-Cray XT4 QuadCore 2.1Ghz - CrayInc.

21. Tipuri de microinstructiuni

Pentru unitatea de comandă micropogramată a calculatorului didactic vom defini două tipuri de microinstructiuni și anume:

- Microinstructiune operatională, care specifică controlul primitivelor funcționale ale unității de execuție
- microinstructiune de ramificație, care permite testarea stării primitivelor funcționale și asigură ramificația în microprogram .

22. Algoritmi de impartire pe niveluri(2)

23. Legatura dintre algoritmi paraleli si arhitecturi paralele

ALGORITMI	ARHTECTURA
Granularitatea modulului de procesare(task, proces, thread, etc)	Complexitatea procesorului + comunicatia intre procesoare
Control concurrent	Analiza modului de operare procesor(mono, pipeline, SIMD, MIMD, etc)
Mecanismul datelor <ul style="list-style-type: none">- Structurare- Modalitate de acces	Structurarea memoriei
Geometria comunicatiei <ul style="list-style-type: none">- statica/dinamica- simetrica/asimetrica	Retea de comutare <ul style="list-style-type: none">- Crossbar- Delta $a^n \times b^n$- Trunchi k
Complexitate algoritm	Nr de procesoare / dimensiune memorie / performanta retea de comutare

24. Un subiect la alegere(3)

25. Descrieti arhitectura din laborator - aici nu vrea cam ceea ce vrea si la urmatoarea intrebare?

intel xeon quad core

8 RAM

12 MB cache

2.5 GHz

26. Descrieti structura sistemului pe care ati lucrat la laborator(2)

- a. Si aici ce vrea mai exact? Procesor, memorie RAM, placa video? Si daca da, stie cineva care sunt?

National Center for Information Technology NCIT - s-a infiintat in 2001 atunci cand a aparut primul cluster al facultatii, **CoLaborator**. In 2006, infrastructura a fost extinsa ducand la aparitia celui de-al doilea cluster mult mai puternic decat CoLaborator, **NCIT**. Cele 2 clustere functioneaza ca un intreg datorita vitezelor foarte mari dintre ele. (se planuieste ca aceasta viteza sa ajunga la 10Gb Ethernet)

Clusterul NCIT al facultății este accesibil prin front-end processor la adresa fep.grid.pub.ro folosind protocolul SSH.

Infrastructura de cloud din cadrul clusterului NCIT este bazata pe solutia opensource Openstack. Aceasta este o solutie de IaaS (Infrastructure as a Service).

In prezent, exista 6 arhitecturi diferite disponibile in cluster si anume:

- Intel Xeon Quad 64b
- Intel Xeon Nehalem 64b
- AMD Opteron 64b
- IBM Cell BE EDp 32/64b
- IBM Power7 64b
- NVidia Tesla M2070 32/64b

Tool-uri folosite in cluster pentru dezvoltare:

- Sun Studio
- Open MP
- Open MPI

Pentru debugging folosim:

- TotalView Debugger
- Sun Studio

Pentru profiling si analize de performanta:

- Sun Studio Performance Tools
- Intel VTune

Configurație (hosturi și caracteristici)

<code>ibm-quad.q</code>	Intel Xeon, E5405, 2 GHz, IBM HS21, 28 nodes
<code>ibm-nehalem.q</code>	Intel Xeon, E5630, 2.53 GHz, IBM HS22, 4 nodes
<code>ibm-opteron.q</code>	AMD Opteron, IBM LS22, 14 nodes
<code>ibm-cell-qs22.q</code>	BECell Broadband, IBM QS22, 4 nodes
<code>ibm-dp.q</code>	2 x 12-core Intel Xeon X5650 + 2 Nvidia Tesla, iDataPlex dx360 M3 Server