

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

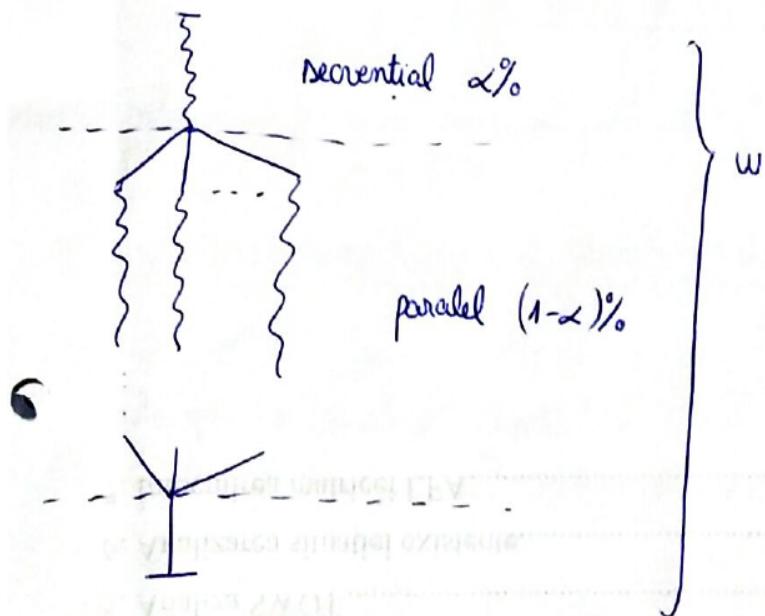
R_L = rata de execuție pe structura monoprocesor

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

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

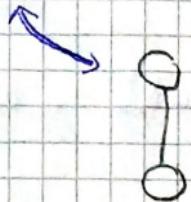
Aceste subgrafe ar putea să aibă mai multe sarcini ?

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

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

↑
Microoperatie terminală

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

→ $\|\mu_{0Tj}\| =$ 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.

$$1 < |\mu_{0Tjk}| \leq N_{\mu_{0T}}$$

↑ numărul de sarcini terminale

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

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

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

↑ estimare a numărului minim al nr. de alocare

OBSERVATTE!

Dacă procesările nu sunt distincte, definim și astfel:

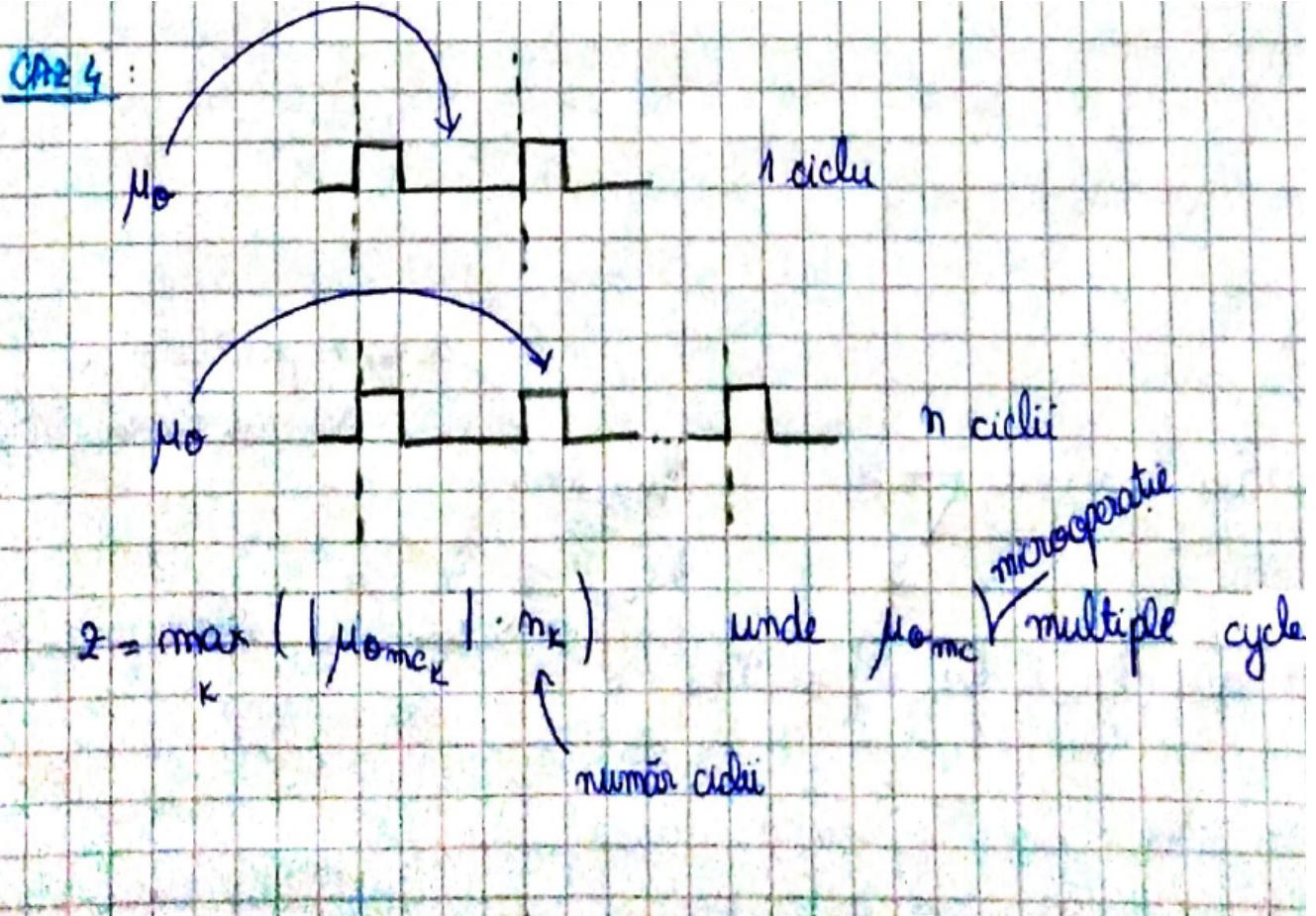
$$g = \max_i \left(\max_j \left(\max \left(\frac{|SGM_{ij}|}{|P_i|}, h(SGM_{ij}) \right) + \min_k \| \mu_{T_{jk}} \| \right) \right)$$

↑ ↗
 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

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

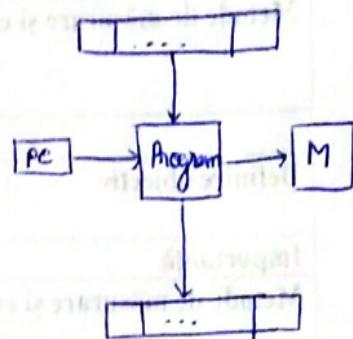
multime finită a stărilor

alfabetul benzii

alfabetul benzii de intrare

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

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

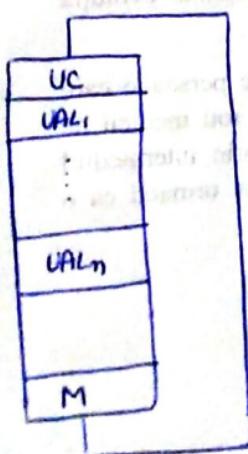


Pipeline

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

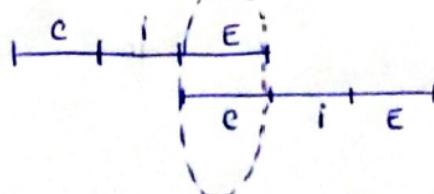
Paralelismul se poate realiza la nivel de \rightarrow preleucrare aritmetică

\rightarrow citire, interpretare, executare instrucțiunii



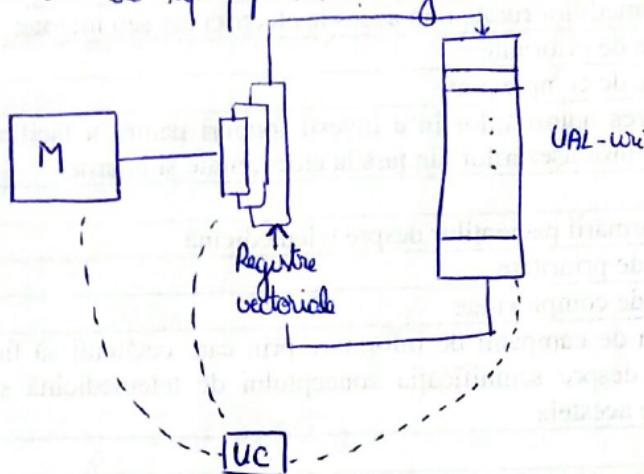
UC are mai multe faze: UCl_1, \dots, UCl_m

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

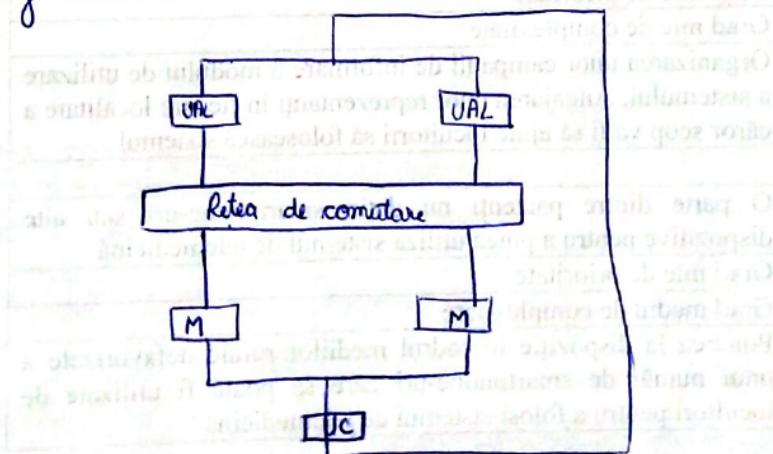
- există un set de instrucțiuni care tratează vectorul ca operand simplu
- există o UAL de tip pipeline și registre vectori (fiecare UAL se ocupă de un registru)



- 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

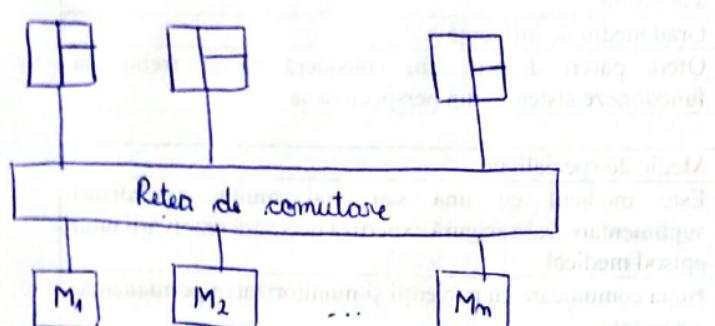
PROCESARE MASIVĂ DE DATE

- 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.
- Încadrează la NUMA

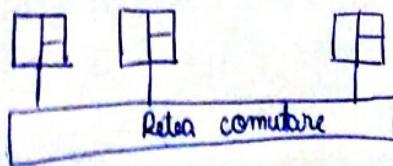


PRAM - STRUCTURĂ PARALELĂ CU ACCES ALEATOR

- extensie a RAM
- comunicatie intre procesare si memorie fara introducere de interzisire
- memorie comună disponibila tuturor programelor
- 4 modalitati de acces la memoria
 - EREW
 - CREW
 - ER CW
 - CRCW

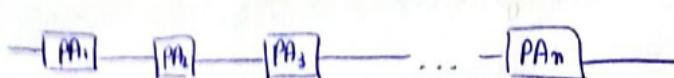
TRANSFER PRIN MESAJE

- Nu se mai partajeaza memoria
- Memoria este distribuita fiecarui procesor a.i. fiecare dispune de propriul program si propria memorie de date.
- Interconexiunea se realizeaza prin schimbari de mesaje prin intermediul unei retele de comutare mesaje
- Exemplu : cluster (nu putem sa vedem memoria reuniata, dar folosim send si receive)



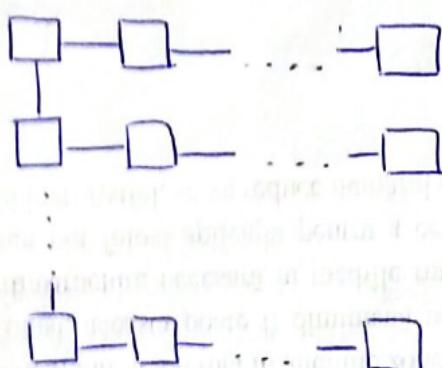
PROCESARE SISTOLICĂ

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



PA = procesor aritmetic

Pipeline liniar

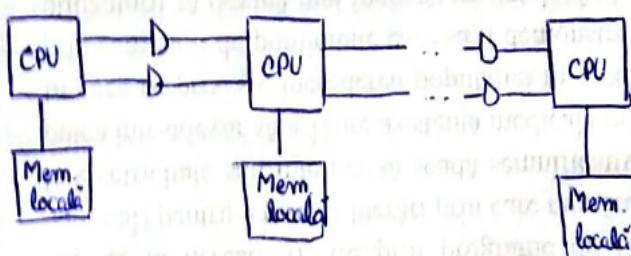


Pipeline matriceal

- se caracterizează prin interconexiuni simple ce permite integrarea VLSI

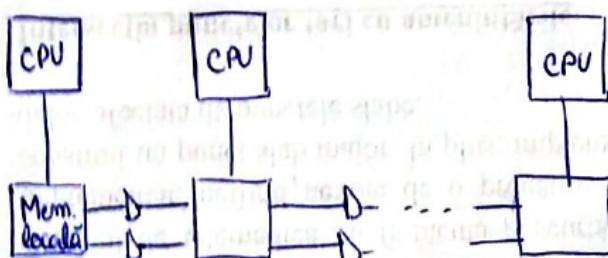
Metode de comunicare în sistemele sistolice:

①



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

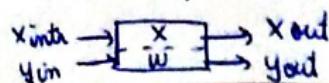
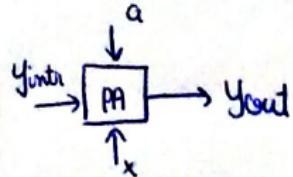
(2)



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

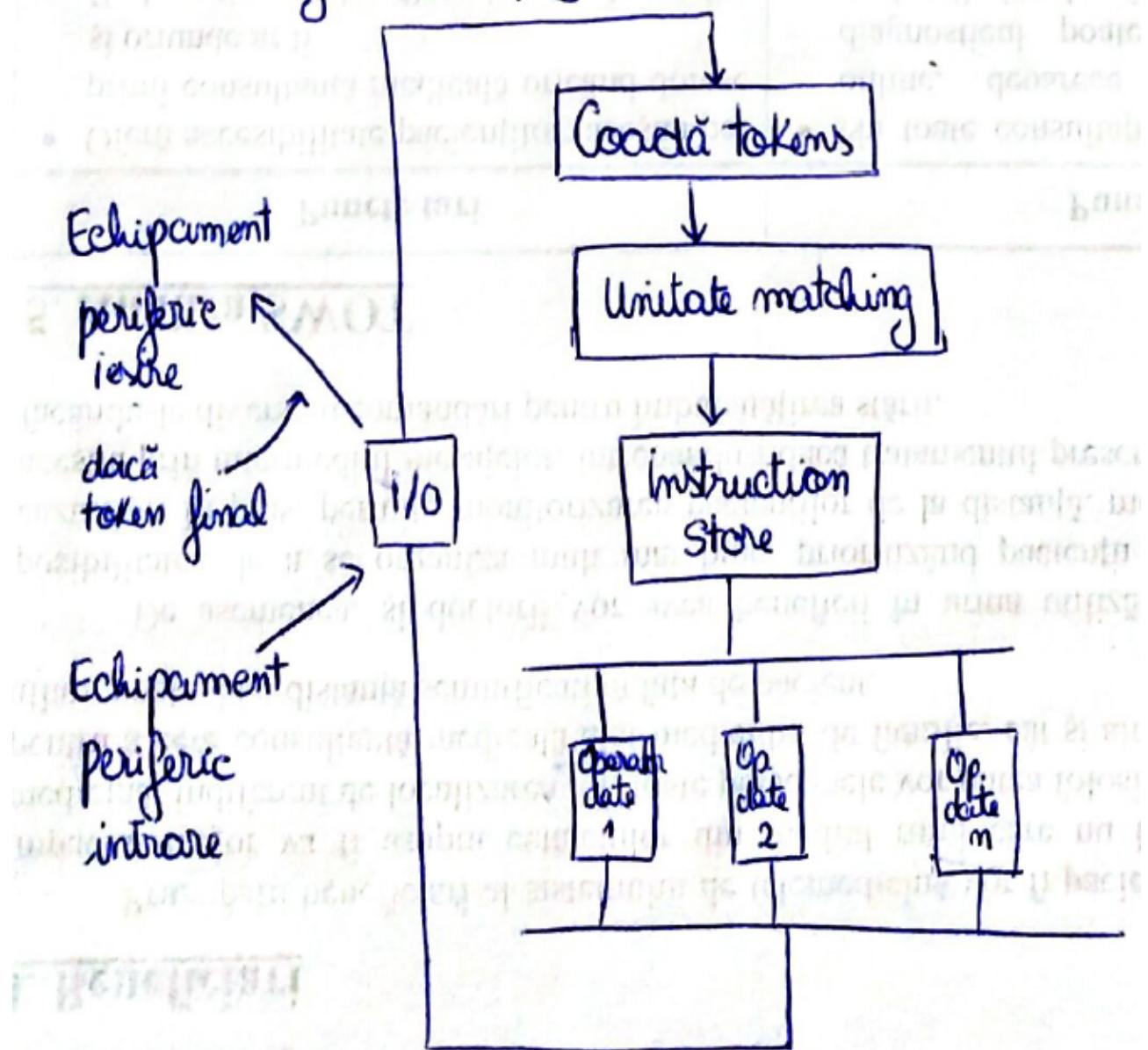


DATA FLOW

- 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 securante 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 procesele care erau independente.

5. Pasii generali ai algoritmului de repartizare microoperatii in microinstructiuni complete

$\mu SB = (\mu B(\mu), <)$ microsublocul pe care trebuie să il partăionam 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> $\mu_{PT_j} = \mu_{PT};$ $\mu_{OD_j} = \mu_{OD} \text{ curent};$ $\{\mu_{IC}\}_j = \{\mu_{IC}\} \setminus \mu_{IC_j}$
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> $\mu_{OD} = \mu_{OD_j}$ <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

$$V(M_i, \alpha'_1) = V(M_i, \alpha'_1 \bar{S} S_i) = V(M_i, \alpha'_1) \downarrow \text{din ipoteză} = V(M_i, \alpha'_2) = V(M_i, \alpha'_2 \bar{S} S_i)$$

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

• $M_i \in R_{S_i}$

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

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

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

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

$$\text{cost}(cc(\mu_0)) = [\log_2(|cc(\mu_0)| + 1)]$$

↑
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

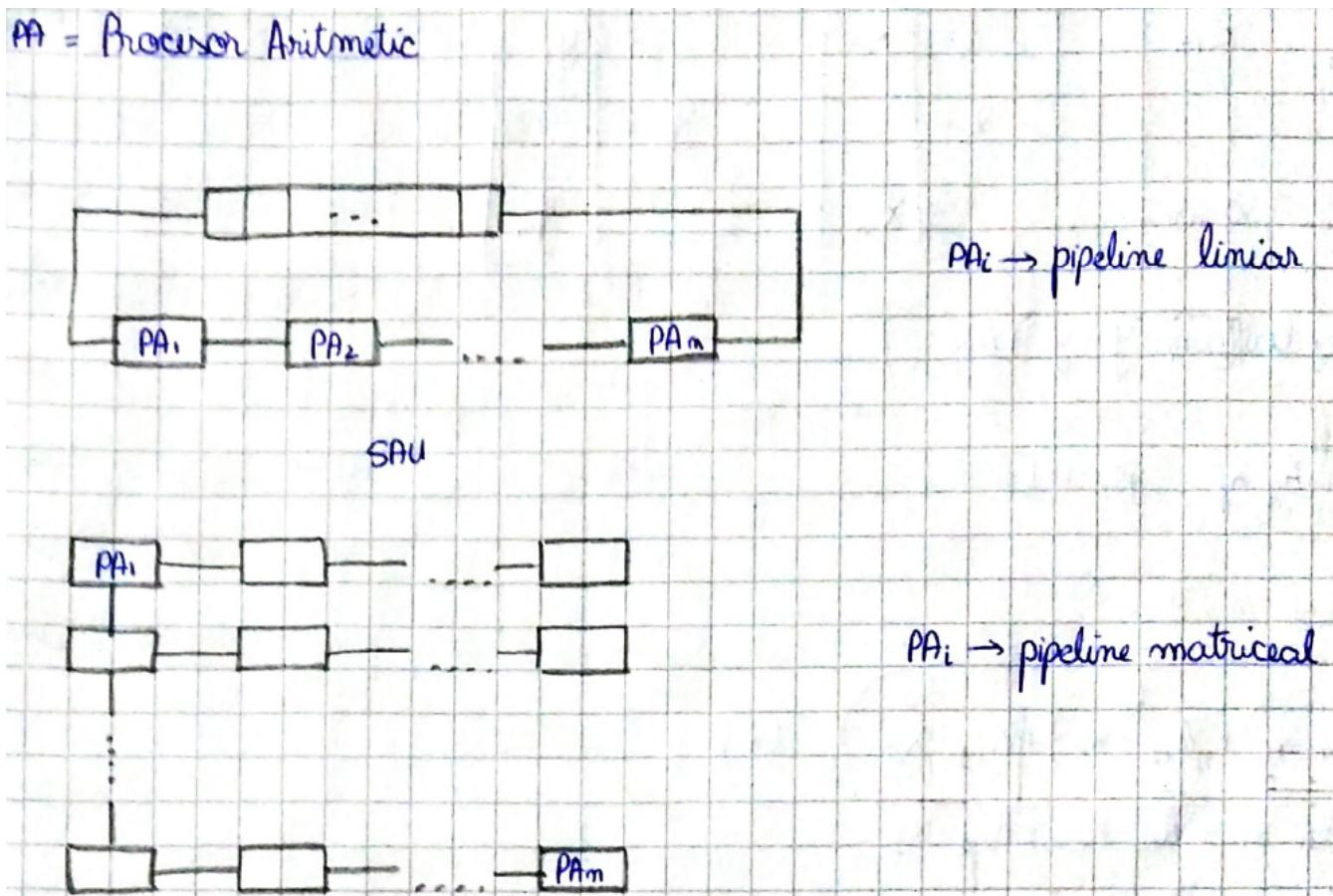
Roadrunner - BladeCenter Cluster IBM

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

Jaguar-Cray XT4 QuadCore 2.1Ghz - CrayInc.

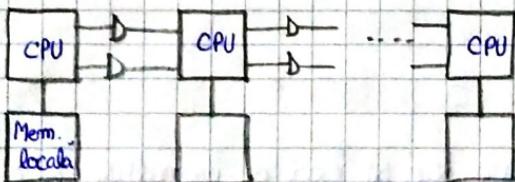
11. Modele sistolic (memorie partajată cu transfer de mesaje)

PA = Procesor Aritmetic



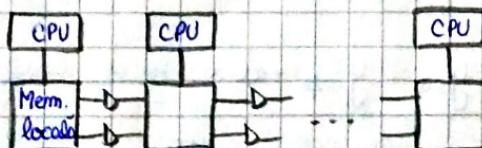
METODE DE COMUNICARE ÎN SISTEMELE SISTOLICE:

(1)



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

(2)

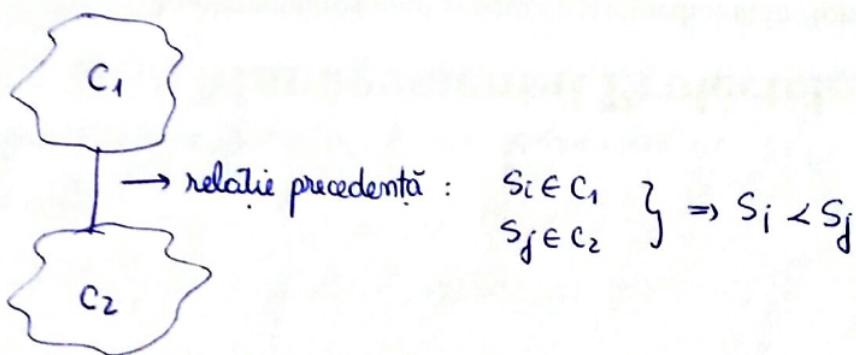


Buclorul conține și pun rezultatul în memorie. În acest moment există un acces direct de memorie care mută datele de ieșire în altă zonă de memorie ca fiind date de intrare.

\Rightarrow necesită implementarea 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 .



$$C = (C_1, C_2)$$

$$\begin{aligned} C_1 &= (S_1, \alpha_1) \text{ unde } \alpha_1 = a_1 \dots a_{2m} \text{ și } S_1 = \{s_{11}, \dots, s_{1m}\} \\ C_2 &= (S_2, \alpha_2) \text{ unde } \alpha_2 = b_1 \dots b_{2m} \text{ și } S_2 = \{s_{21}, \dots, s_{2m}\} \end{aligned}$$

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

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.

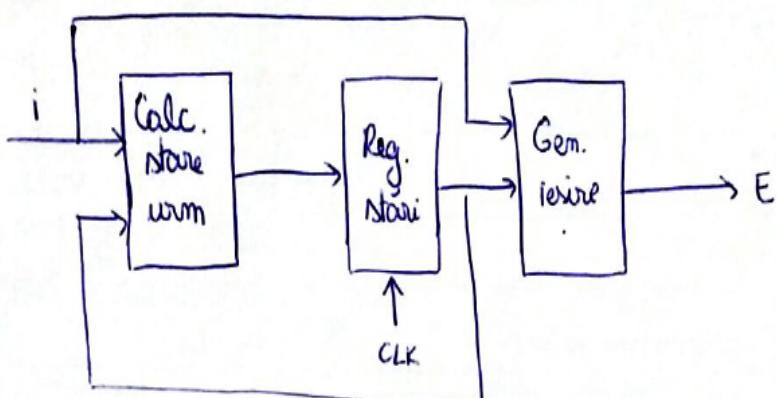
$$C' = (S, \alpha')$$

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

14. Structura unei unități de comandă microprogramate

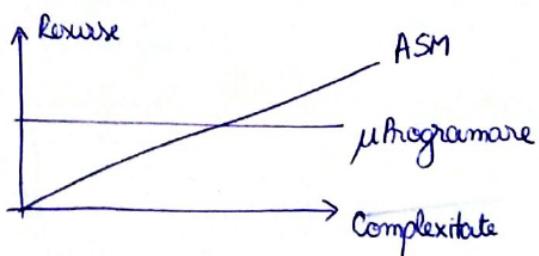
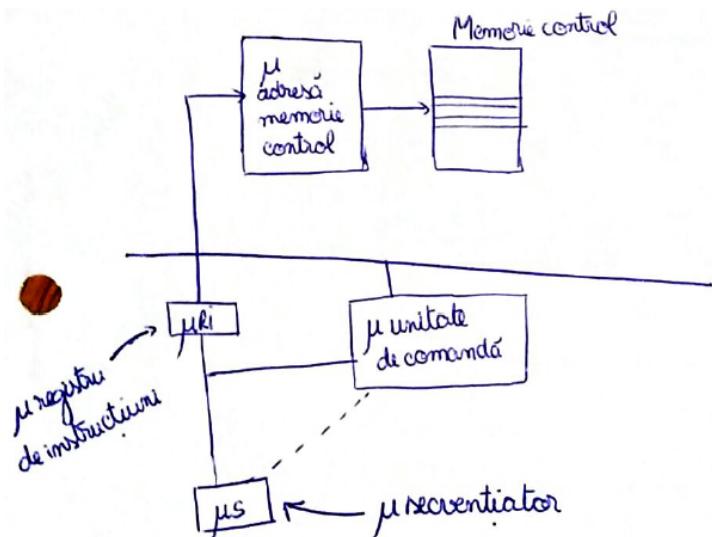
Puteți fi implementată în 2 moduri:

- * 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 execuțiază instrucțiuni pe mai multe fluxuri date
 - **MOPS** (milioane operații pe secundă)
 - ↑ nu ține seama de natura operațiilor sau lungimea curvenilor
 - **MFLOPS** (milioane de operații în virgulă 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

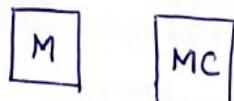
17. Modalitati de implementare a memoriei de control

Poate fi privită din 2 perspective

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

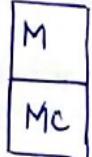
Perspectiva 1 : Relația dintre MC și M

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



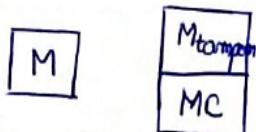
$$V_{MC} \gg V_M$$

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



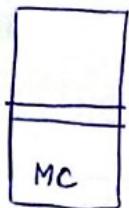
Memoria trebuie să fie suficient de rapidă ca să poată fi folosită ca Mc și deșul de ieftină ca să fie folosită ca M.

- 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



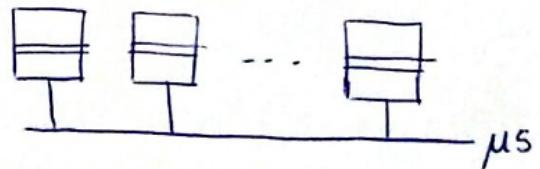
- un curănt din memorie $\Rightarrow \mu$ instrucțiune
- citire μ instrucțiune \Rightarrow un singur acces la MC

- 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:

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



- MC cu organizare pe blocuri

Există 2 tipuri de adresa

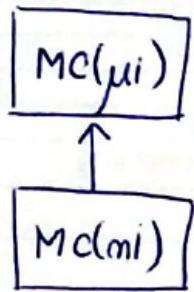
- adrese de μ instrucțiuni din același bloc cu μ instrucțiunea curentă
- adrese de blocuri

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

- MC divizată

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

- 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

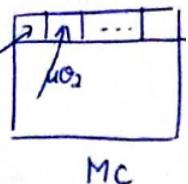
Există 6 tipuri de implementații:

• 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

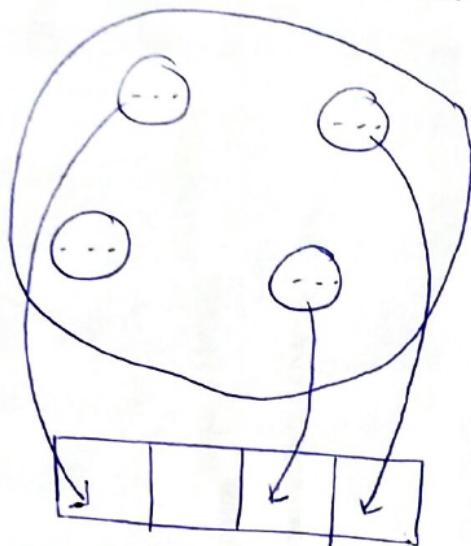
• CODIFICARE ORIZONTALĂ

- 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

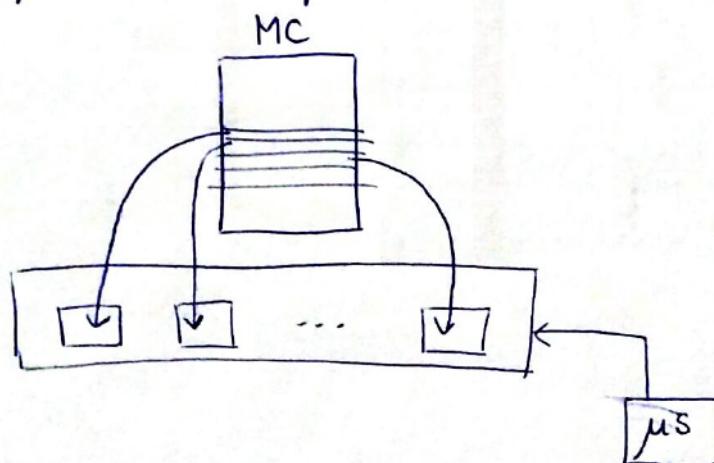
- combina flexibilitatea și paralelismul de la codificarea orizontală cu eficiența codificării verticale
- Nobilește operațiile care nu se pot executa în paralel.
- grupurile cu operațiile care se exclud reciproc \Rightarrow codificare verticală + le punem în același câmp
- grupurile cu operațiile care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



- Este necesar un decodificator pentru fiecare câmp

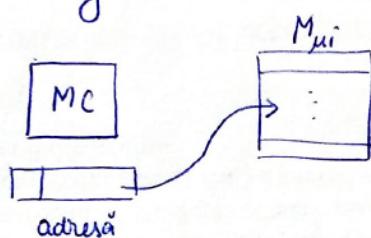
• CONFIICARE CU CONTROL RESIDUAL

- 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 cînd se realizează aceeași operatie în mod repetat



• CONFIICARE CU CONTROL PRIN ADRESE

- formă simplificată a conceptului de nanoprogramare
- fiecare μi 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.