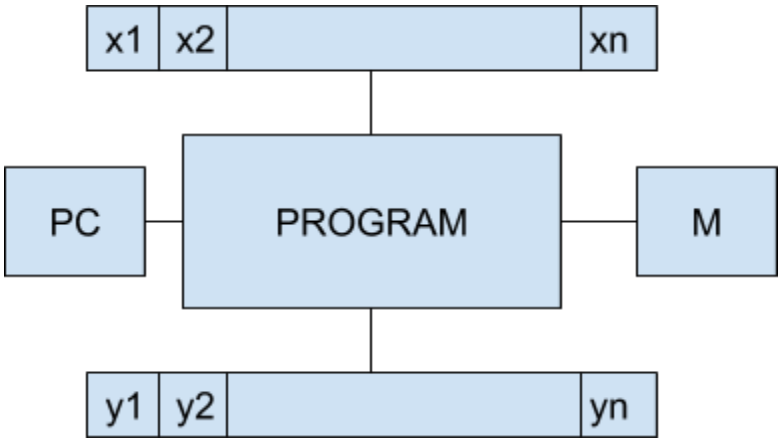
I. d

Modele de calcul paralel

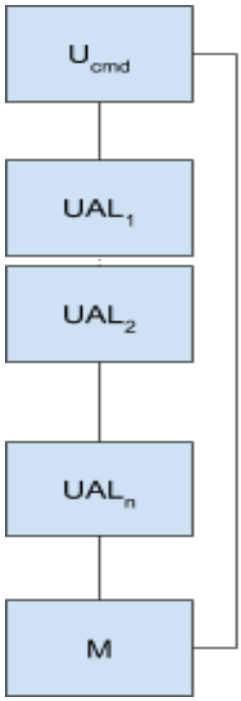
A. RAM (Random Access Machine)

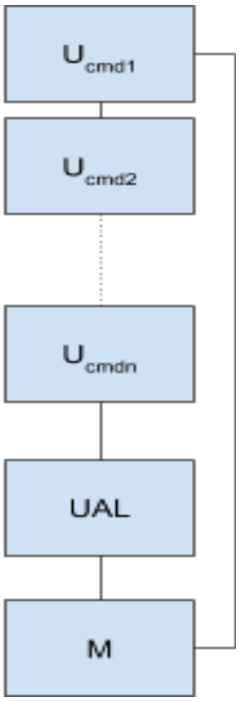
- Masina de baza monoprocesor
- Similar cu modelul Turing
- Analiza complex algoritmilor si dezvoltarea lor
- Punctul de plecare pentru modelele paralele
- La fiecare moment de timp se executa cate o instructiune



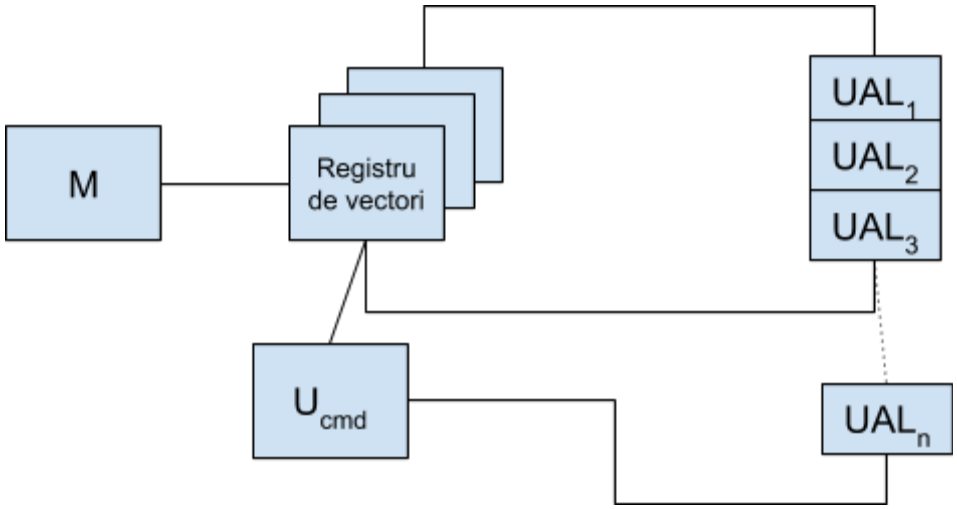
B. Pipeline

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

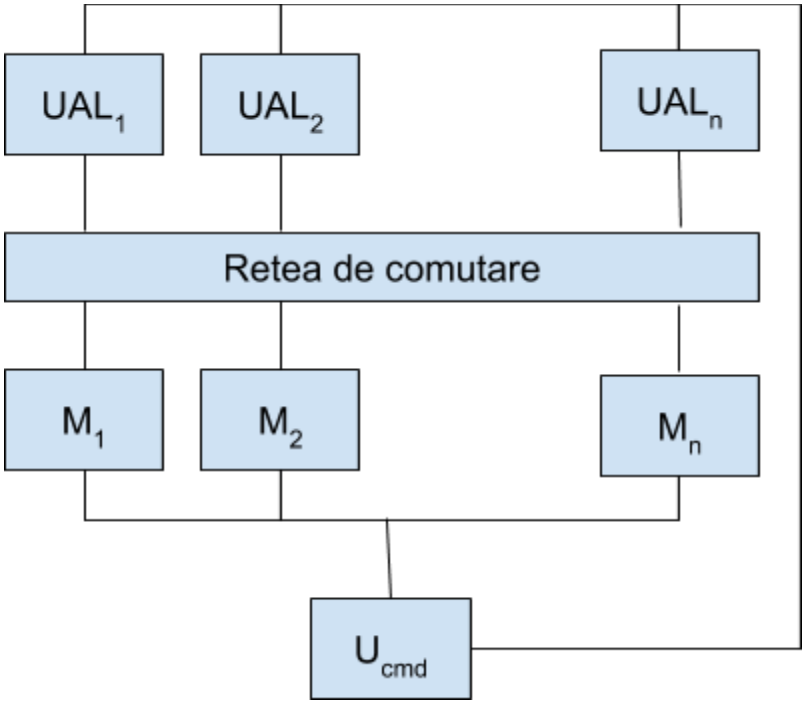




C. Procesare de vectori
   - Set de instructiuni care trateaza vectorul ca un operand simplu
   - Sisteme monoprocesor care au ca extensie procesarea de vectori
   - SIMD

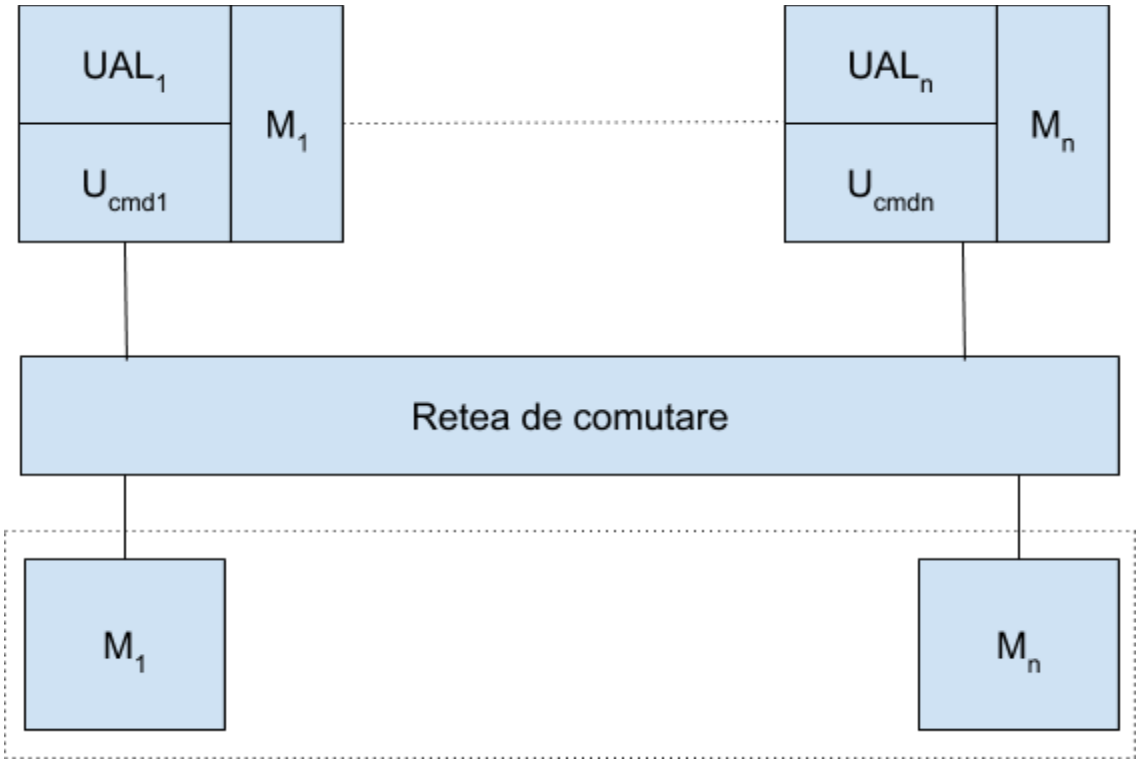


D. Procesare de “masive de date” (MPP)
   - Calculatoare care constau din unitati aritmetice multiple supervizate de o singura unitate de control, care efectueaza aceleasi op la un mom dat
   - Asemanator cu SIMD + UMA



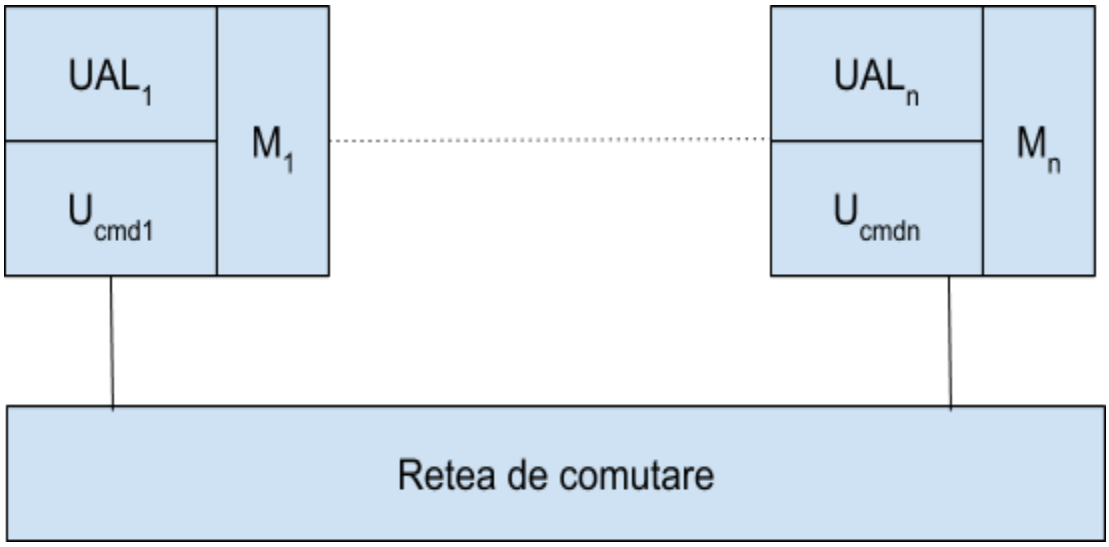
E. Multiprocesoare cu memorie partajata
   - Fiecare procesor are propriile UAL, Ucmd si M
   - Comunicarea se face prin intermediul unei retele de comutare, cu module de memorie partajata
   - Reteaua de comutare permite atat schimbul de informatii cat si accesul la fiecare modul al memoriei partajate
   - 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 comuna 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 operaţii 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 contorulului 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 -> execuţie asincronă ce începe când datele sunt disponibile
   - Viteză ridicată & potenţial ridicat de paralelism
   - Uşor de integrat dpdv al sincronizării & comunicării
   - Pot exista timpi mari de aşteptare, overhead mare de control, iar prelucrarea structurilor de date (vectori, matrice etc) e foarte dificilă

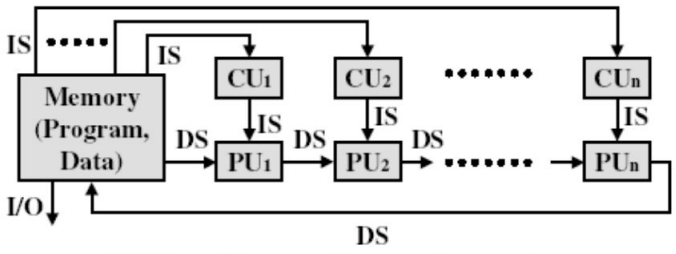
II. Subiecte examen

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, Ucmd,M) si de asemenea au si o memorie comuna
    - Timpul de acces la memoria comuna in cazul memoriei partajate este mai mare deoarece mai multe procese pot incerca simultan sa o acceseze, dar doar unul poate 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

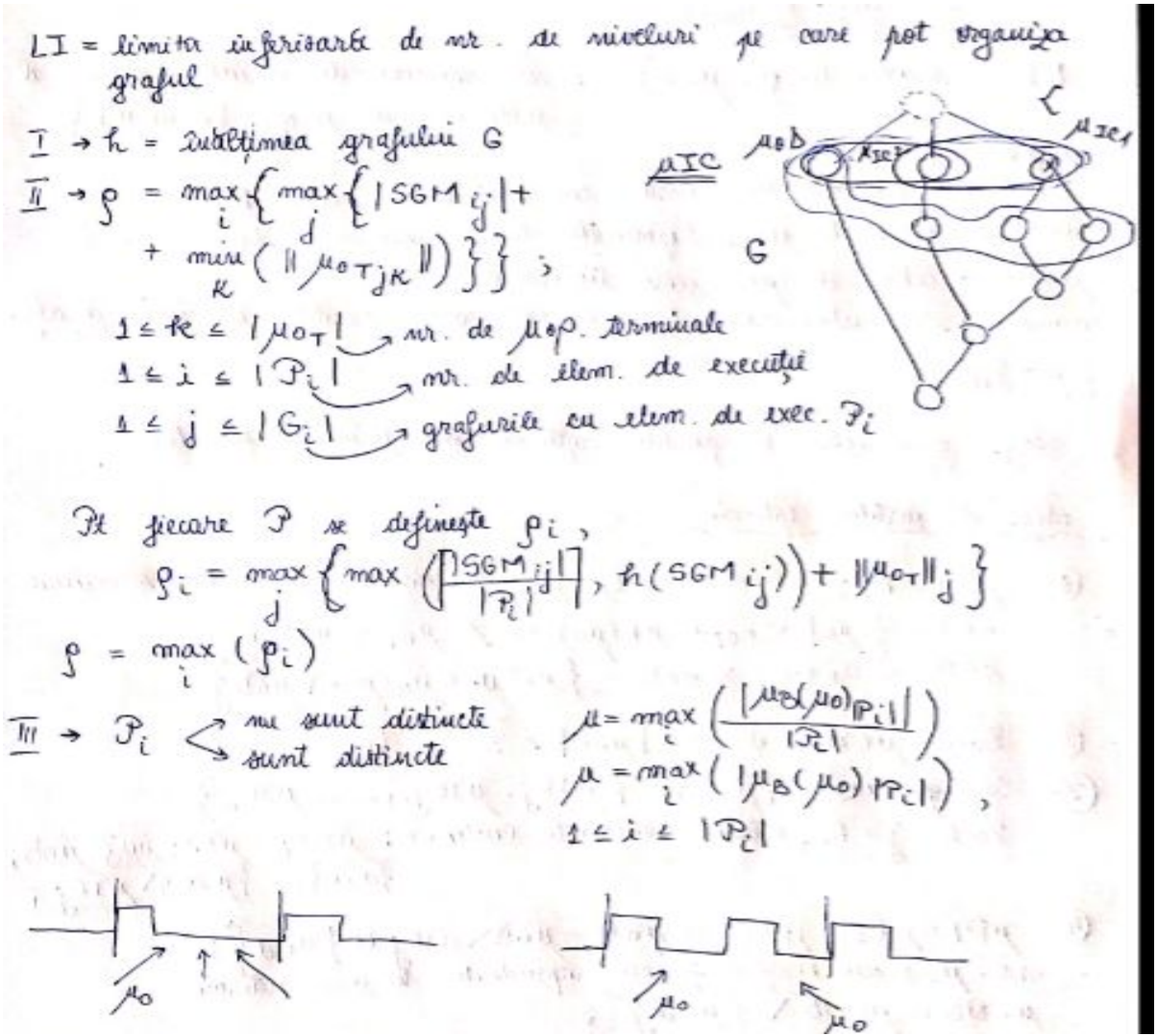
MISD (Multiple Instruction Streams Over A Single Data Streams)

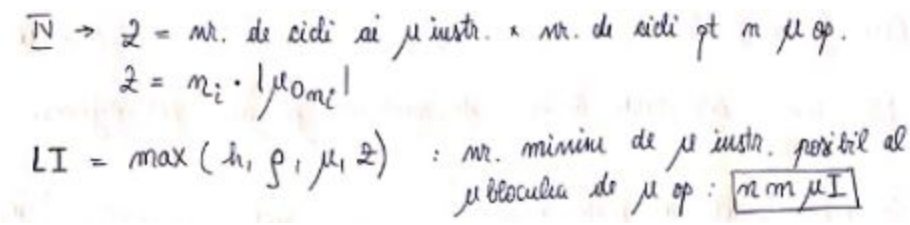
- ❖MISD
  - ✓Processor arrays, systolic arrays
  - ✓Special purpose computations



MISD architecture (the systolic array)

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





4. Gustafson

Gustafson

w - workflow

α - procent secvential

1- α - 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 secvential

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

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

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

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 secventiale

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)

$$VNp = \frac{T1}{Tnp} = \frac{1}{\frac{ts}{Nt}+\frac{1}{N}*[\frac{N}{p}](1+\frac{t0}{t})}$$

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

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

$\frac{t0}{t}$ - 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 secventialitate

$$R(f) = \frac{1}{\frac{f}{Rh}+\frac{1-f}{Rl}}$$

$$R(f) = \frac{1}{(1-f)+\frac{f}{N}}$$ N- 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 $mPt=\{mIC_1,mIC_2,...,mIC_{[mPt]}\}$ partiția unui microsubbloc în microinstrucțiuni complete și $MB(mO)=\{mO_1,mO_2,..., mO_{[MB]}\}$ setul de microoperații distincte din cadrul microsubblocului.

Două microoperații 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 microoperații trebuie privită în sensul că cele două microoperații nu sunt specificate (nu sunt active) niciodată împreună în cadrul unei microinstrucțiuni din microbloc. Controlul resurselor sistemului microprogramat nu necesită niciodată efectuarea în paralel a celor două microoperații.

Def. 2

Două microoperații $mO_i \in MB(mO)$, $mO_j \in MB(mO)$ sunt **incompatibile** dacă există cel puțin o microinstrucțiune 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 microoperații sunt compatibile între ele.

$CC(mO)=\{mO \mid$ pt orice $mO_i, mO_j \in CC(mO)$ avem mO_i 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.

$$\text{Cost } CC_i = [\log_2(|CC_i|+1)]$$

iar costul total de implementare al cuvântului de control

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

unde k este numărul de clase de compatibilitate.

10. Sisteme microprogramate si microprogramabile(3)

Sistemele microprogramate se refera la modalitatea de implementare a unitatii de comanda fara a oferi resursele hardware si suportul de programe pentru accesul utilizatorului la nivelul programului.

Sistemele microprogramabile ofera atat resursele hardware cat si facilitatile software pentru accesul la nivelul microinstructiunilor

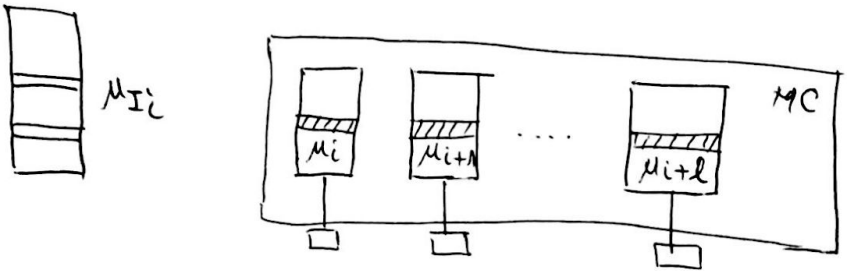
11. Organizarea memoriei de control

Org mem de control in functie de mem principala:

1. M si Mc sunt entitati separate
2. M si Mc sunt aceeasi entitate
3. M si Mc sunt entitati separate, dar sunt incarcate in aceeasi entitate

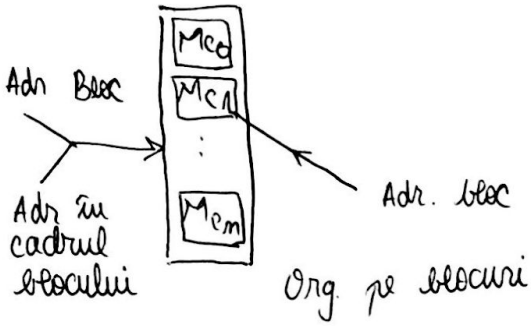
Org mem de control:

1. Pagini
2. Blocuri
3. 2 nivele

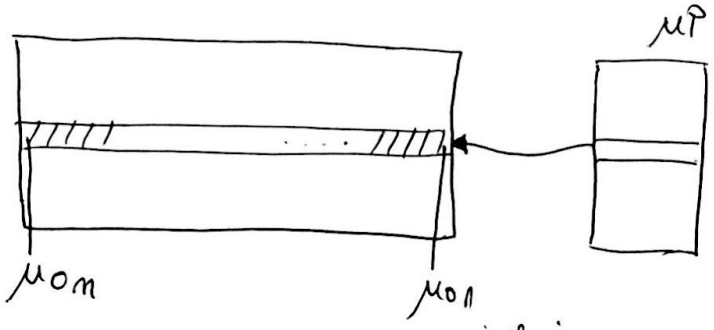


- citesc 1 instr.
- citesc și execut

Org. pe pagini



Org. pe blocuri



Org. pe 2 niveluri

- reducere foarte mare de spațiu de mem.
- de 2 x timpul de acces la mem.

-

12. Algoritmi de partitionare - vezi urmatoarele 2 exercitii

13. Algoritmul de partitionare (in 8 pasi)

Alg. de partiție optimă

① $\mu PT = \emptyset$; $\mu PTA = \mu_B(\mu_0)$; se consideră că se exec. secvențial

$\mu_0 D = \{\mu_0 \mid \forall \mu_{0i} \in \mu_B(\mu_0) \Rightarrow \nexists\ \mu_{0i} < \mu_0\}$;

$\mu_0 ND = \mu_B(\mu_0) \setminus \mu_0 D = \{\mu_0 \mid \mu_0 \in \mu_B(\mu_0) \setminus \mu_0 D\}$

② Dacă $\mu_0 ND = \emptyset$ și $|\mu_0 D| \leq 3 \Rightarrow$ ⑤

③ Se generează $\{\mu IC\} = \{\mu IC_j, \mu IC_{j+1}, \dots, \mu IC_k\}$

Dacă $j \neq k$, atunci salvează CONTEXT : $\mu PT_j = \mu PT$; $\mu_0 j = \mu_0 D$;

$\{\mu IC\}_j = \{\mu IC\} \setminus \mu IC_j$;

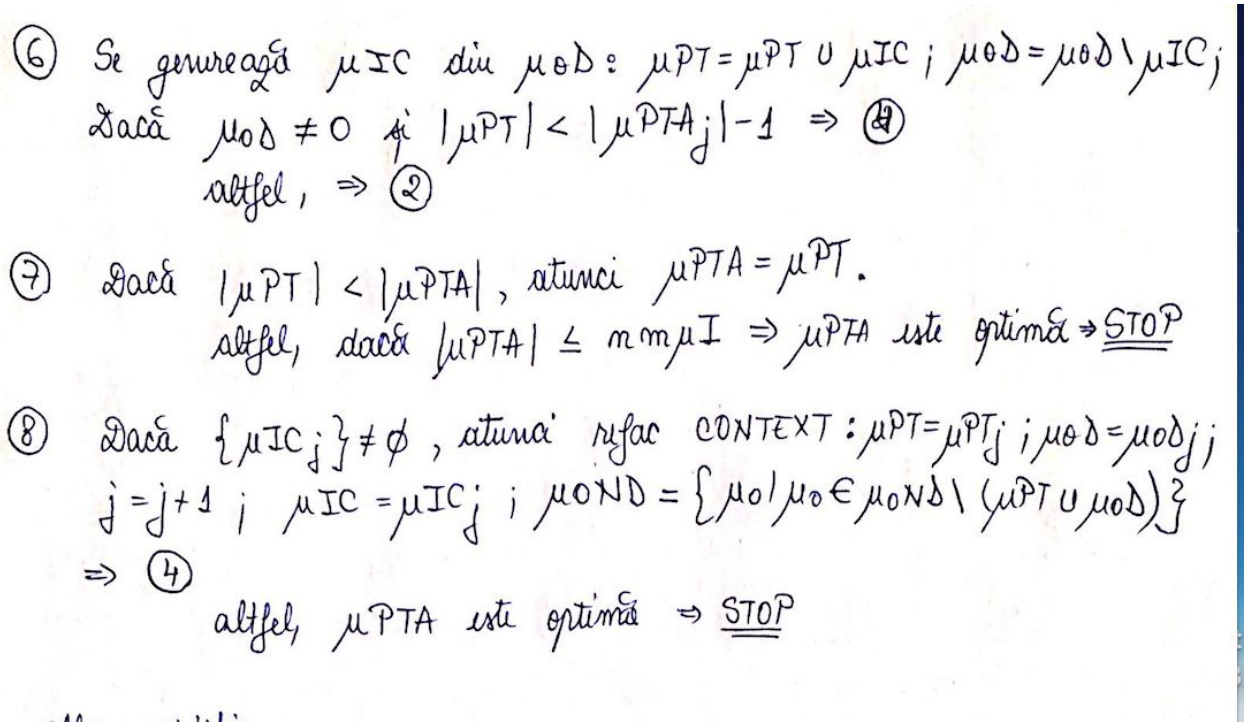
④ $\mu PT = \mu PT \cup \mu IC_j$; $\mu_0 D = \mu_0 D \setminus \mu IC_j \cup \{\mu_0 d\}^*$;

obs: $\mu_0 d$ nu conține și cele dependente de $\mu_0 d$ anterior

$\mu_0 ND = \mu_0 ND \setminus \{\mu_0 d\}^*$;

Dacă $|\mu_0 D| \neq 0$ și $|\mu PT| \leq |\mu PT_j| - 1$, atunci ⇒ ②

⑤ Dacă $\mu_0 D = \emptyset$, atunci ⇒ ⑦



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

// Algoritm euristic

1. Pas 1. Initializare

$$\begin{cases} \mu PT \neq \Phi \text{ , } \mu OD = \{\mu o \mid \mu o \in \mu OS \neq \mu o' < \mu o\} \\ \mu OD = \mu B(\mu o) \setminus \mu OD \end{cases}$$

$p_{succ}(\mu o)$ = nr de succesori ai operatorului respectiv

$p_{succ}(\mu JC) = \sum_{\mu o \in \mu JC} p_{succ}(\mu o)$

2) daca μOMS atunci pas 4

3) genereaza setul $\{\mu JC\} = \{\mu JC_1, \mu JC_{j+1}, \ldots \mu JC_k \ldots$ ordonat.

- consideram sortata aceasta multime dupa numarul de succesori ai μJC
- $p_{succ}(\mu JC_j) \geq p_{succ}(\mu JC_{j+1}) \geq \ldots \geq p_{succ}(\mu JC_k)$

$\mu PT = \mu PT \cup \mu JC_j$

$\mu OS = \mu OS \setminus \mu JC_j \setminus \{\mu od\}_j$

$\mu ONS = \mu ONS \setminus \{\mu od\}_j$

salt 2

4. Genereaza $\{\mu JC\} = \{\mu JC_h \ldots \mu JC_k\}$ le ordonam dupa un criteriu

$\mu PT = \mu PT \cup \mu JC_h$

$\mu OS = \mu OS \setminus \mu JC_h$

// salt pas 2 - comentat

daca $|\mu OS| \neq 0$ atunci salt 4

altfel μPT este cea aproape optima.

15. Teorema de suficienta(5)

Teorema de suficienta

Sistemul de sarcini format din sarcini mutual neinterferente este determinat.

********** **daca cere si:**

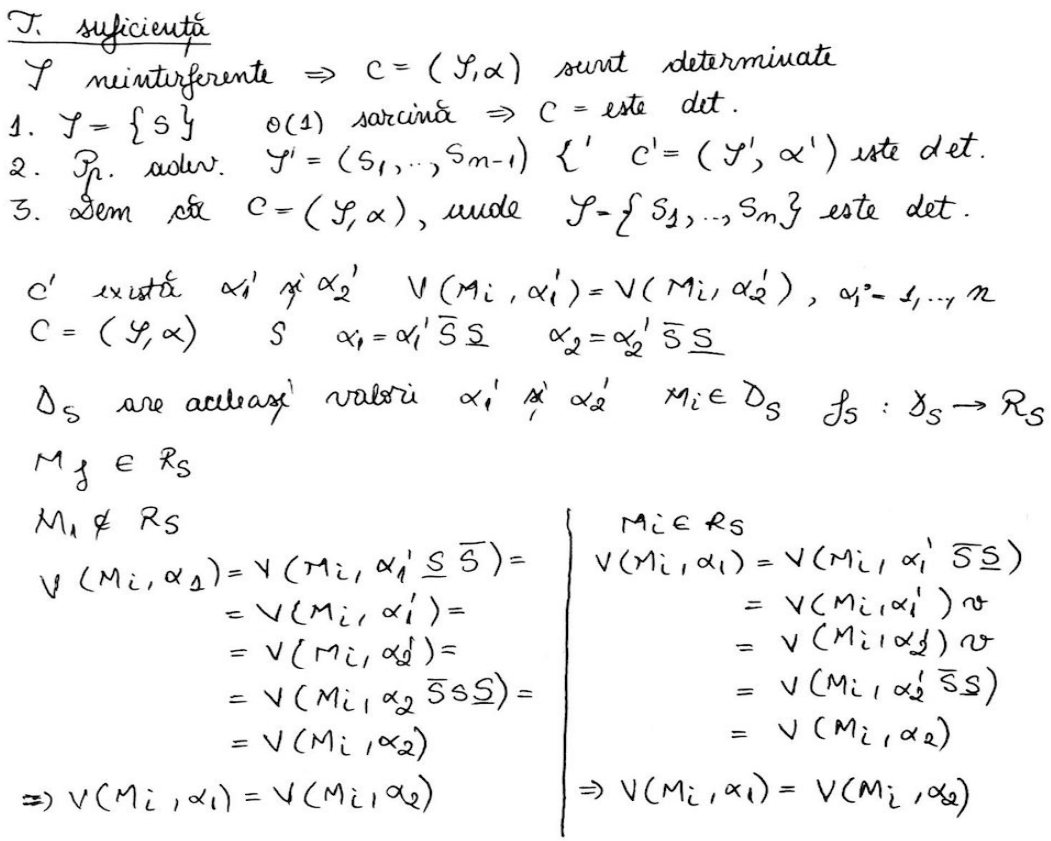
Teorema de necesitate:

Fie dat un sistem de sarcini neinterpretat, dar cu DS si RS≠0, sistemul C este determinat pt ∀ interpretare a sarcinilor sale ⇔ sarcinile sunt neinterferente

Sisteme echivalente de sarcini:

Doua sisteme sunt echivalente ⇔ sunt determinate si produc aceleasi secvente de valori pentru o stare initiala data

=



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

Vp = T1/Tp >> 1 (t 1 proc/ t p proc) -> raport intre prelucrarea paralela si cea secventiala

p*Tp >> T1 -> se consuma timp cu sincronizarea, comunicarea, overhead creat de interactiunea intre procese

Caz ideal p*Tp = T1

c. Eficienta de prelucrare

Ep = Vp/p = T1/(p*Tp) < 1

d. Redundanta

Rp = Op/O1 (nr op p proc/nr op 1 proc) -> t pierdut de overhead

e. Utilizarea

Up = Op/(p*Tp) < 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

$\alpha = \alpha_1\alpha_2...\alpha_{2n}$ $\alpha_i = \bar{S}$ sau $a_i = \underline{S}$

$\bar{S}$ sau $\underline{S}$ apar o data in α

1. $a_j = \bar{S}_i$ $a_k = \underline{S}_j$ => j < k (initierea e intotdeauna inaintea terminarii)
2. $a_l = \underline{S}_m$ $a_p = \bar{S}_o$ $S_m < S_o$ => 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 şir α = a1 a2.....a2n de evenimente de inițiere şi terminare a proceselor cu respectarea relațiilor de precedență impuse de <.

c. Secvente partiale

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

α_p - secventa partiala -> indica ce sarcini sunt active la un mom dat

S_i este activa daca exista a_j, $a_j = \bar{S}_i$, $j \leq k$ si nu exista $a_l = \underline{S}_i$, $l \leq k$

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 şi cooperează pentru realizarea unor operaţii logice şi de calcul, şi produce acelaşi 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:

P_1: $R_1 \leftarrow$ BUSFN (M; DCD (ADR))

P_2: M * DCD (ADR) $\leftarrow R_2$

P_1 - citeşte din locaţia de memorie de la adresa ADR;
P_2 - scrie în locaţia de memorie de la adresa ADR.

Nedeterminarea se poate rezolva introducând o relaţie de precedenţă adecvată între procesele care erau independente.

Pentru a stabili condiţiile necesare şi suficiente ca un sistem să fie determinat, vom considera un model simplificat în care sistemul fizic este privit ca o mulţime ordonată de locaţii de memorie:

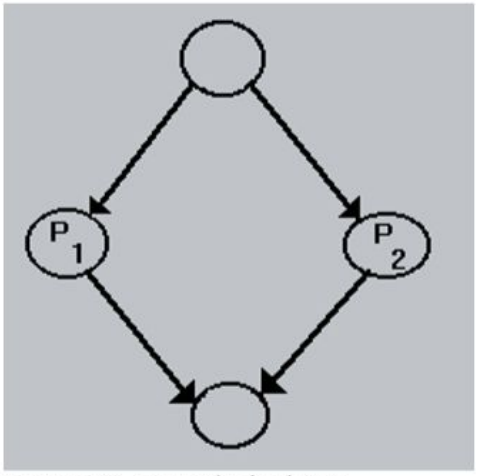


Figura 5.2. Exemplu de sistem nedeterminat

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

ce conţin orice valoare dintr-o mulţime de valori **V**.

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

De exemplu dacă:

$\alpha = a_1 a_2 ..... a_{2n}$ şi

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

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

$$s_k = [M_1[k], M_2[k], ..., M_m[k]]$$

Mulţimea tuturor stărilor poate fi definită astfel:

$\mathbf{\Sigma} = \mathbf{V}^m$ $\qquad$ $\mathbf{V}^m =$ $\qquad$ $[\mathbf{V} \times \mathbf{V} \times ..... \times \mathbf{V}]$, produs cartezian.

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

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

unde:

$d = |D_p|$ cardinalul domeniului valorilor de intrare, D_p;

$r = |R_p|$ cardinalul domeniului valorilor de ieşire, R_p.

Pentru o stare iniţială s_0 şi o secvenţă de execuţie α, secvenţa corespunzătoare de stări:

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

Fie $D_p = (M_{x1}, M_{x2}, ..., M_{xd})$ domeniul valorilor de intrare;

$R_p = (M_{y1}, M_{y2}, ..., M_{yr})$ domeniul valorilor de ieşire;

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

2°. dacă $a_{k+1} = \underline{P}$, şi $a_l = \overline{P}$, $1 \le k$, atunci stările locaţiilor de memorie din domeniul de valori al lui P la momentul k+1 sunt:

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

şi

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

Altfel spus, dacă a_{k+1} este un eveniment de iniţiere 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 Rs, noile valori din Rs fiind determinate de f_p pe baza valorilor din domeniul de definiţie din momentul imediat precedent iniţierii lui P .

Secvenţa de stări $\sigma = s_0 s_1 ..... s_{2n}$ ce rezultă dintr-o secvenţă de execuţie, 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 verticala
   i. Avantaj: codificare minima (dimensiune)
   ii. Dezavantaj:
      1. eliminarea controlului paralel asupra resurselor
      2. inflexibilitatea dezvoltării sau completării sistemului în ceea ce priveşte introducerea de noi microoperații.

b. Organizare orizontala
   i. Avantaj: paralelism maxim + flexibilitate mare
   ii. Dimensiune prea mare

c. Codificare minimala
   i. Combina flexibilitatea si paralelismul potențial oferite de codificarea orizontală cu eficiența codificării verticale

d. Codificare minimala pe 2 niveluri
   i. Mai putin eficienta decat cea minimala

e. Codificare cu control rezidual
   i. asigură o economie de memorie de control atunci când unele primitive funcționale realizează aceeaşi 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 paralele

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 microprogramată a calculatorului didactic vom defini două tipuri de microinstrucțiuni şi anume:

- Microinstructiune operationala, care specifica controlul primitivelor functionale ale unitatii de executie
- microinstrucțiune de ramificație, care permite testarea stării primitivelor funcționale şi asigură ramificația în microprogram .

22. Algoritmi de impartire pe niveluri(2)

23. Legatura dintre algoritmi paraleli si arhitecturi paralele

ALGORITMI	ARHITECTURA
Granularitatea modulului de procesare(task, proces, thread, etc)	Complexitatea procesorului + comunicatia intre procesoare
Control concurent	Analiza modului de operare procesor(mono, pipeline, SIMD, MIMD, etc)
Mecanismul datelor - Structurare - Modalitate de acces	Structurarea memoriei
Geometria comunicatiei - statica/dinamica - simetrica/asimetrica	Retea de comutare - Crossbar - Delta $a^n x b^n$ - 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 vitezii 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)

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