



Arhitectura Sistemelor de Calcul



Computer Science
& Engineering
Department

Universitatea Politehnica Bucuresti
Facultatea de Automatica si Calculatoare

cs.ncit.pub.ro

curs.cs.pub.ro



- De ce avem nevoie de paralelism?
- Structuri de calcul cu prelucrare paralela
- Clasificarea sistemelor de calcul / Flynn: SISD, SIMD, MISD, MIMD
- Exemple de utilizare a structurilor:
 - SISD
 - SIMD
 - MIMD
- Exemplu cu/fara dependenta de date pe sisteme de calcul MIMD



Words of Wisdom

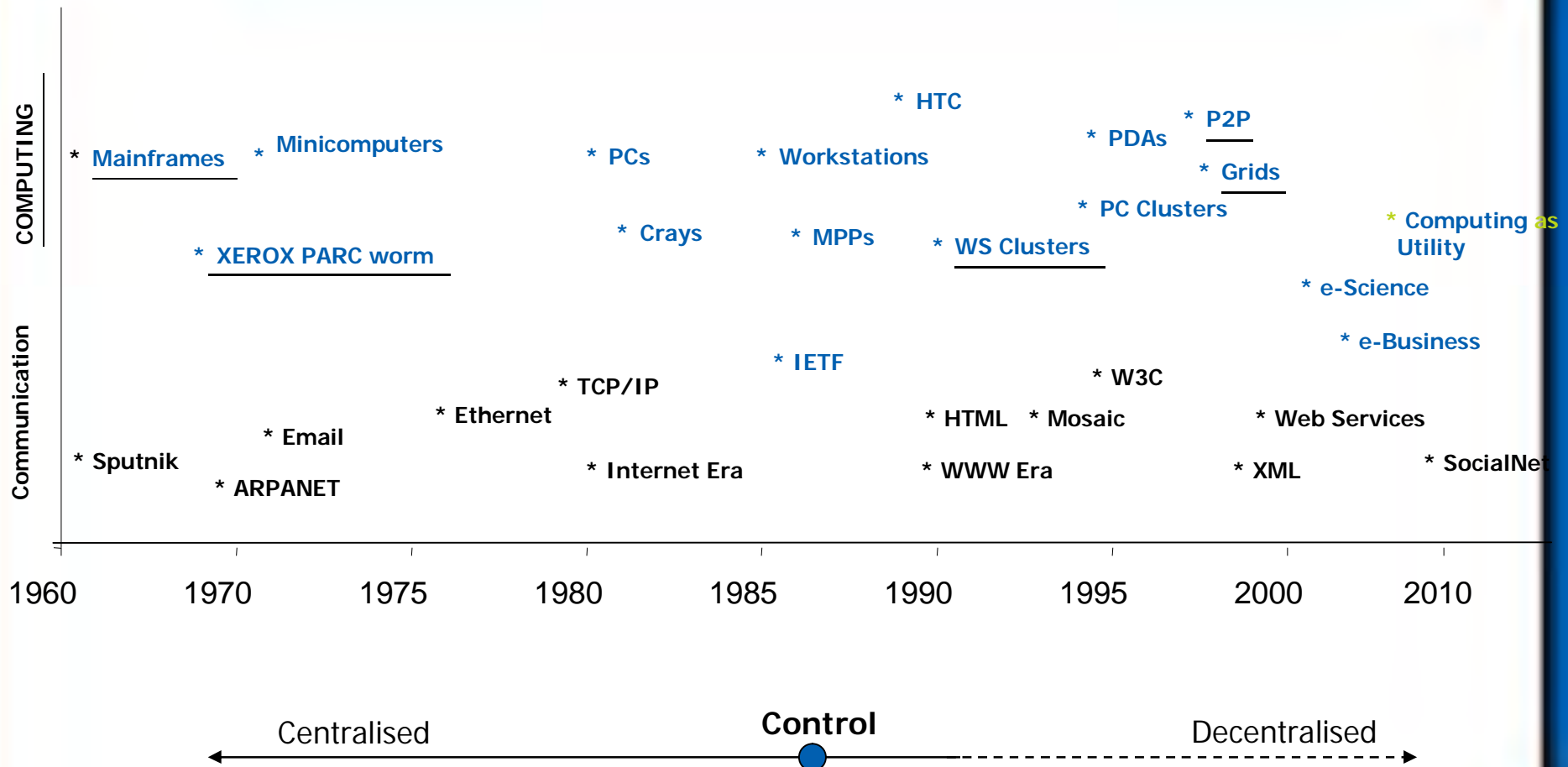
3

- “I think there is a world market for maybe five computers.”
 - Thomas Watson, chairman of IBM, 1943.
- “There is no reason for any individual to have a computer in their home”
 - Ken Olson, President and founder of Digital Equipment Corporation, 1977.
- “640KB [of main memory] ought to be enough for anybody.”
 - Bill Gates, Chairman of Microsoft, 1981.



Computing and Communication Technologies Evolution: 1960-2010!

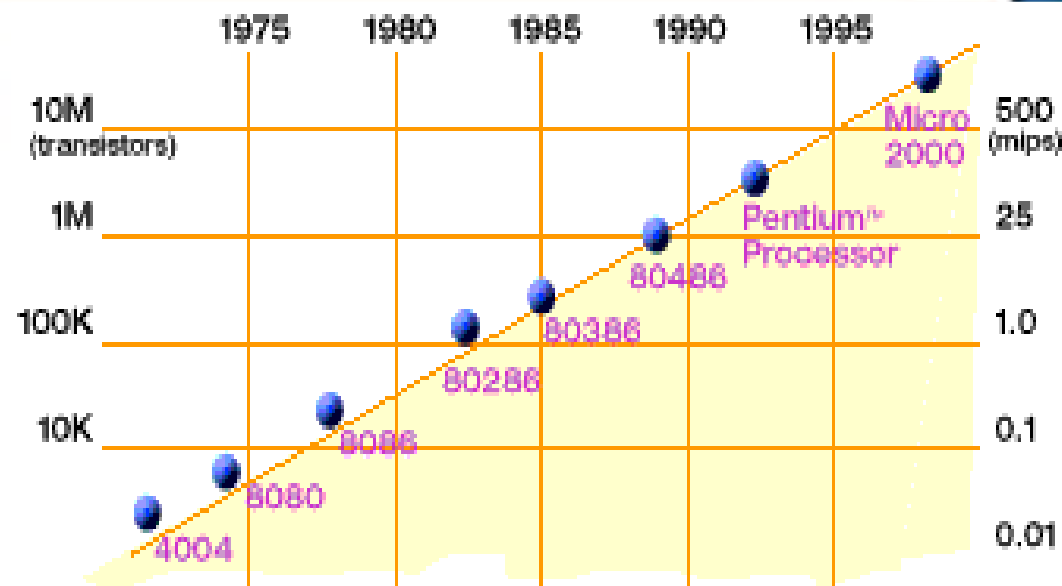
4





Evolutia Microprocesoarelor

5



Gordon Moore (cofondator Intel) a prezis in 1965 ca densitatea tranzistoarelor in cipurile cu semiconductori se va dubla intr-un interval aproximativ de 18 luni

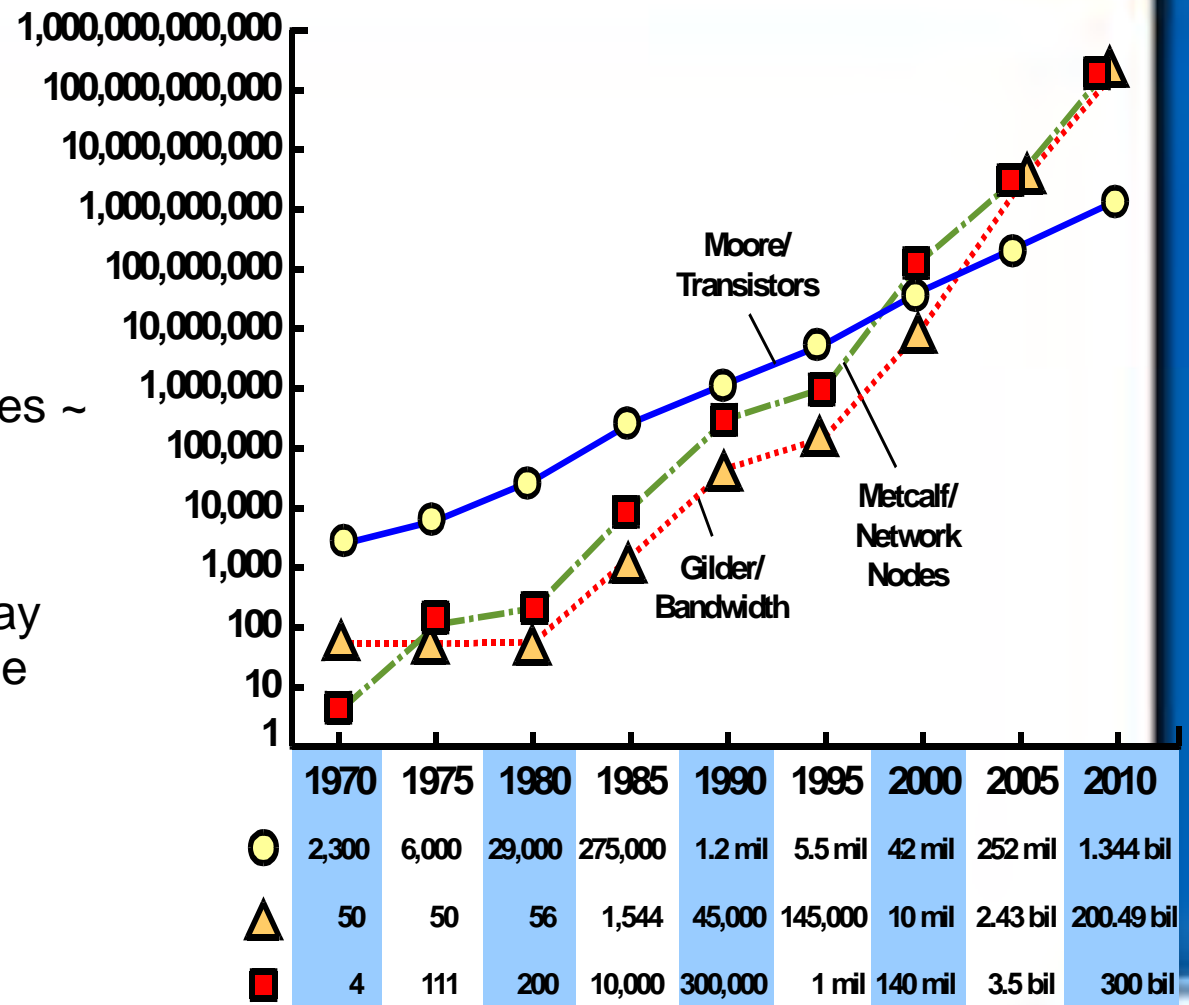
**Legea lui Moore
se confirma!**



Performance, Capability, Value of ICT as defined by the three Laws of Computing

6

- Moore's Law.
 - Transistors on a single chip doubles ~ every 18–24 months.
- Gilder's Law.
 - Aggregate bandwidth triples ~ every year.
- Metcalfe's Law.
 - The value of a network may grow exponentially with the number of participants.



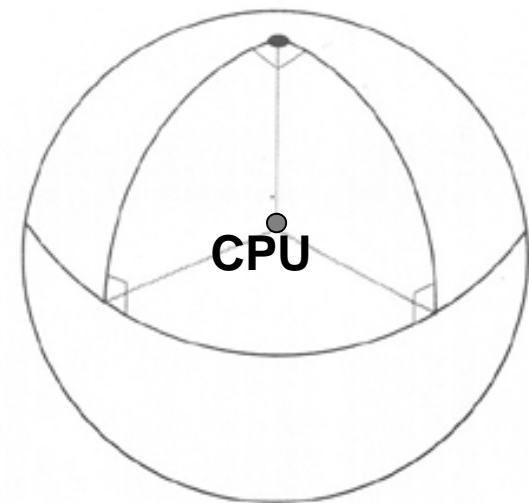
Source: Cambridge Energy Resource Associates



Un procesor de 10TFlops?

7

- Putem construi un CPU neconcurent care
 - Oferă 10,000 de miliarde de operații în virgulă mobilă pe secundă (10 TFlops)?
 - Operează pe 10,000 miliarde de bytes (10 TByte)?
- Este reprezentativ pentru necesitățile cercetătorilor din ziua de azi
- Ceasul procesorului trebuie să fie de 10,000 GHz
- Presupunem că datele circulă cu viteza luminii
- Presupunem că procesorul este o sferă “ideală”





Un procesor de 10TFlops?

8

- Masina genereaza o instructiune pe ciclu, si de aceeas ceasul trebuie sa fie de aproximativ $10,000\text{GHz} \sim 10^{13} \text{ Hz}$
- Datele au de parcurs o distanta intre memorie si CPU
- Fiecare instructiune necesita cel putin 8 bytes de memorie
- Presupunem ca datele circula cu viteza luminii $c = 3e^8 \text{ m/s}$
- Astfel, distanta intre memorie si CPU trebuie sa fie $r < c / 10^{13} \sim 3e^{-6} \text{ m}$
- Apoi, trebuie sa avem 10^{13} bytes de memorie in $4/3\pi r^3 = 3.7e^{-17} \text{ m}^3$
- Si de aceea, fiecare cuvant de memorie trebuie sa ocupe maxim $3.7e^{-30} \text{ m}^3$
 - Ceea ce inseamna **3.7 Angstrom³**
- Aceasta dimensiune corespunde unei molecule foarte mici, formata din cativa atomi...
- Densitatea curenta a memoriei este de $10\text{GB}/\text{cm}^3$, sau cu un factor de ordinul **10^{20} mai mica decat ceea ce ar fi necesar!**
- **Concluzie:** Acest procesor nu va fi disponibil pana cand quantum computing nu va deveni mainstream 😊



Paralelismul este necesar!

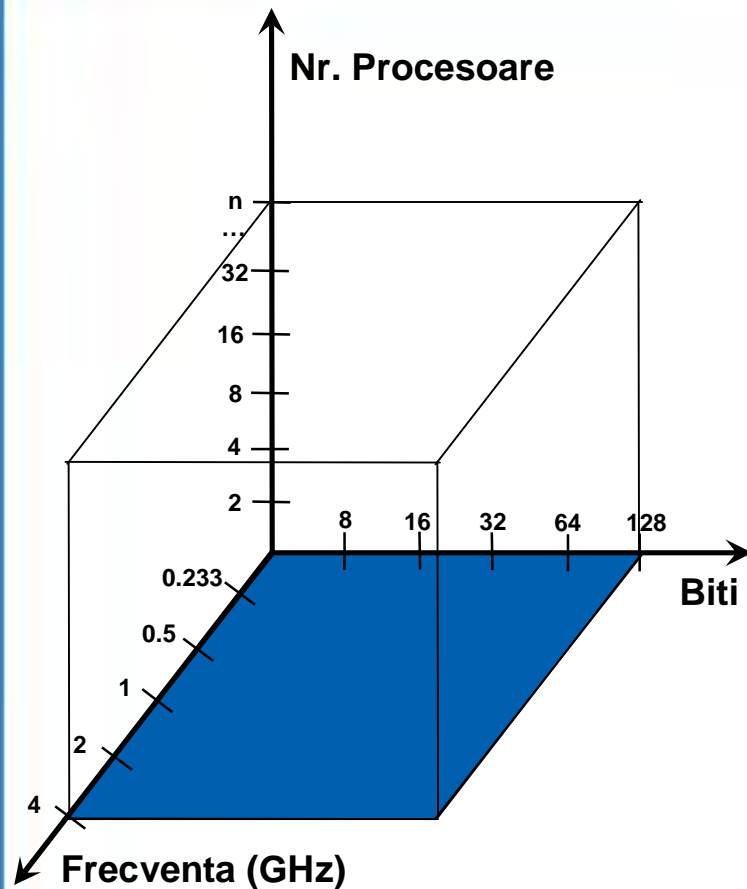
9

- Paralelism la nivel de Bit (Bit-level parallelism)
 - Operatii in virgula mobila
- Paralelism la nivel Instructiune (ILP)
 - Mai multe instructiuni pe ciclu
- Paralelism la nivelul memoriei
 - Suprapunerea intre operatii cu memoria si de calcul
- Paralelism al sistemului de operare
 - Mai multe thread-uri, procese pe CPU-uri multiple, in cadrul aceleasi masini
- Paralelism distribuit
 - Mai multe masini conectate impreuna



Structuri de Calcul cu Prelucrare Paralela

11



- Aria → puterea de calcul
- Marimea puterii de calcul: volumul cubului - “n” e limitat doar de cost → [arhitecturi paralele de calcul](#)
- Orice problema are un grad de paralelism intrinsec ce depinde de natura ei
- Trebuie tinut seama de:
 - Algoritmi paraleli specifici
 - Limbaje adecvate de programare
 - Limitările SO
 - Arhitectura intrinseca a SC utilizat

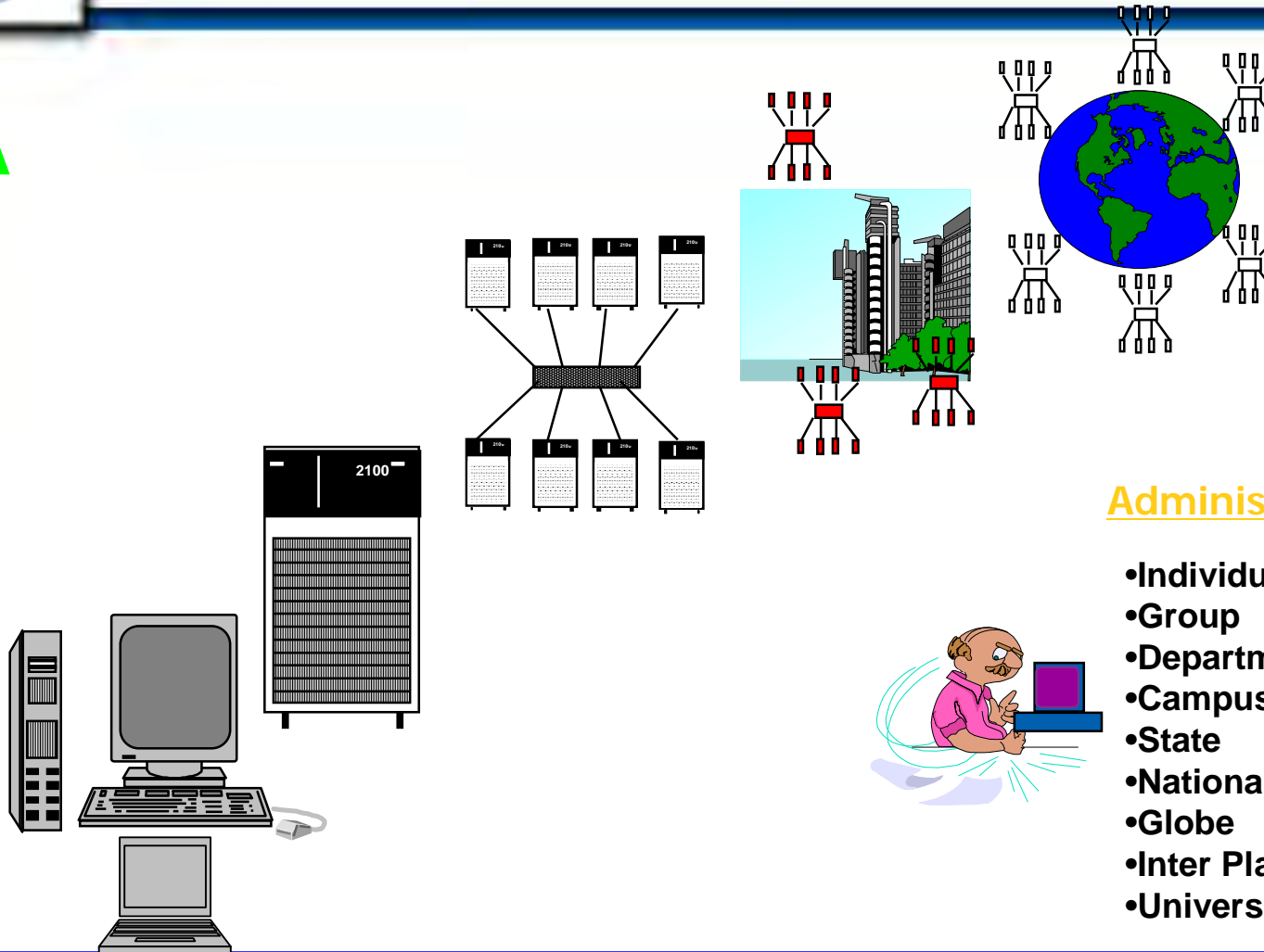
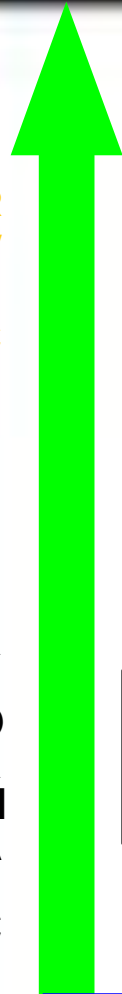
In general gradul de paralelism este de 10%



Computing is Scaling: Towards Inter-Planetary Level



SERVICES
+
PERFORMANCE



Administrative Barriers

- Individual
- Group
- Department
- Campus
- State
- National
- Globe
- Inter Planet
- Universe

Personal Device

SMPs or
SuperComputers

Local
Cluster

Enterprise
Cluster/Grid

Global
Grid

Inter Planet
Grid



Nivelele Prelucrării Paralele – 1

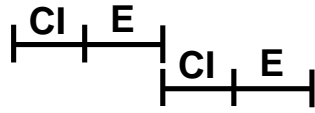
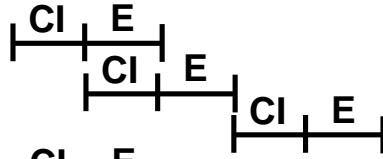
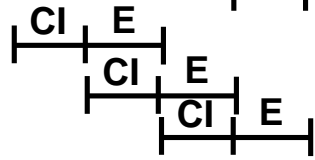
14

- 1: Paralelismul la nivel de Bloc/Job:
 - Intre Job-uri diverse
 - Sunt necesare: un mecanism de salvare a contextului; un ceas pentru divizarea timpului; canale I/O pt transfer
 - Intre fazele unui Job
 - Citirea sursei programului
 - Compilare
 - Link-are
 - Executarea codului obiect
 - Salvarea rezultatelor
 - Anumite faze ale unui job pot fi suprapuse
- 2: Paralelismul la nivel de subansamble Hardware:
 - Intre elemente de prelucrare ale vectorilor
 - Intre componentele logice ale dispozitivelor aritmetico/logice (carry look ahead)



Nivelele Prelucrării Paralele – 2

15

- 3: Paralelismul la nivel de Program:
 - Intre diverse sectiuni (independente) ale unui program
 - Intre bucelele (loop-urile) unui program
 - Ambele presupun analiza dependentelor de date!
- 4: Paralelismul la nivel de Instructiune:
 - Intre diverse faze ale ciclului instructiune: CI = Citire & Interpretare
E = Executie
 - Implementare seriala:
 - Implementare serie-paralela:
 - Implementare paralela:
 - Este necesar un mecanism de predictie al salturilor



- Structuri de calcul cu prelucrare paralela
- Clasificarea sistemelor de calcul / Flynn: SISD, SIMD, MISD, MIMD
- Exemple de utilizare a structurilor:
 - SISD
 - SIMD
 - MIMD
- Exemplu cu/fara dependenta de date pe sisteme de calcul MIMD



Clasificarea Sistemelor de Calcul

17

- **Sisteme Matriceale (Processor Array):**
 - Unitatea de baza a informatiei este **vectorul**
 - Dispun de instructiuni similare SC Von Neumann – operatiile asupra vectorilor sunt efectuate in aceeasi instructiune
- **Sisteme Multiprocesor (Multiprocessor Systems):**
 - Formate din N unitati de prelucrare interconectate printr-o retea de comutare (Strans/Slab cuplata)
 - Sistemele lucreaza independent la realizarea aceluiasi Job
- **Sisteme Pipeline:**
 - Dispun de mai multe unitati de prelucrare asezate in banda de asamblare (RISC):
 - Fiecare UC executa o prelucrare specifica si transfera rezultatul subansamblului adiacent



Taxonomia lui Flynn

18

- Impartirea sistemelor de calcul in functie de:
 - Fluxul de Instructiuni – secventa de instructiuni executate de procesor
 - Fluxul de Date – secventa de operanzi manipulata de procesor
- Bazate pe acest criteriu se desprind:
- I – SISD = Single Instruction Single Data Stream (structura Von Neumann)
- II – SIMD = Single Instruction Multiple Data Stream
- III – MISD = Multiple Instruction Single Data Stream
- IV – MIMD = Multiple Instruction Multiple Data Stream

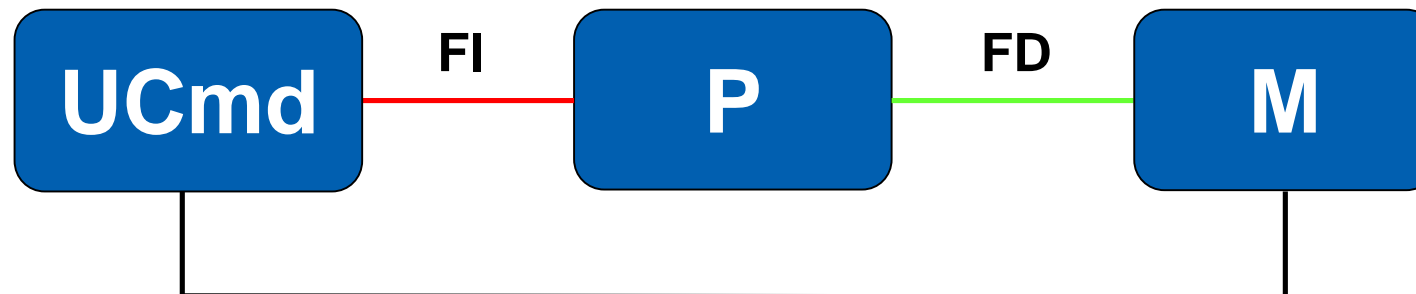


I – SISD

19

- **UCmd** = Unitate de comanda
- **P** = Unitate de prelucrare aritmetica
- **M** = Memorie
- **FI** = Flux Instructiuni
- **FD** = Flux Date

SISD = 1 FI & 1 FD



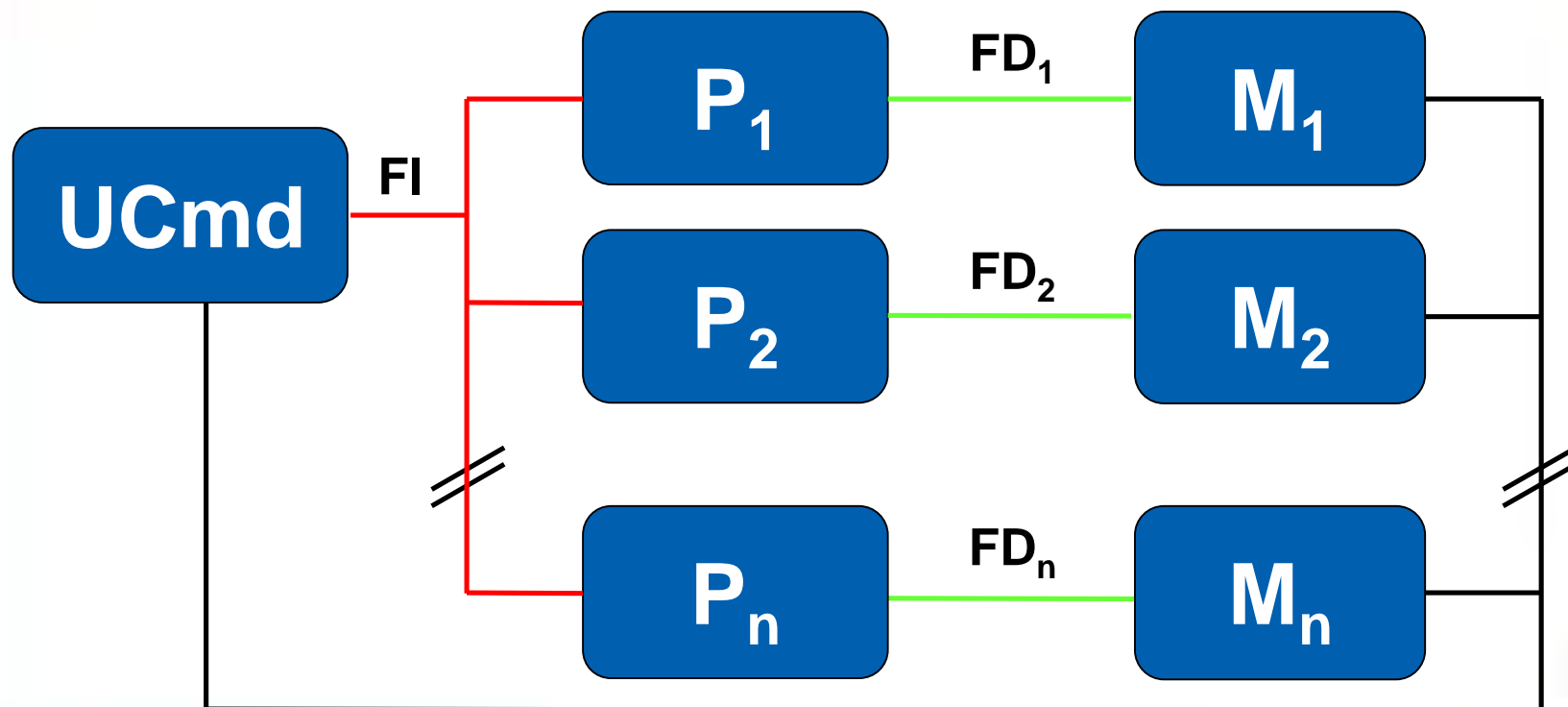


II – SIMD

20

- SIMD sunt masini vectoriale:
 - O singura UCmd
 - Mai multe procesoare si module de memorie (orice procesor vede orice memorie)
- Toate procesoarele fac aceeasi op impusa de UCmd prin FI

SIMD = 1 FI & n FD



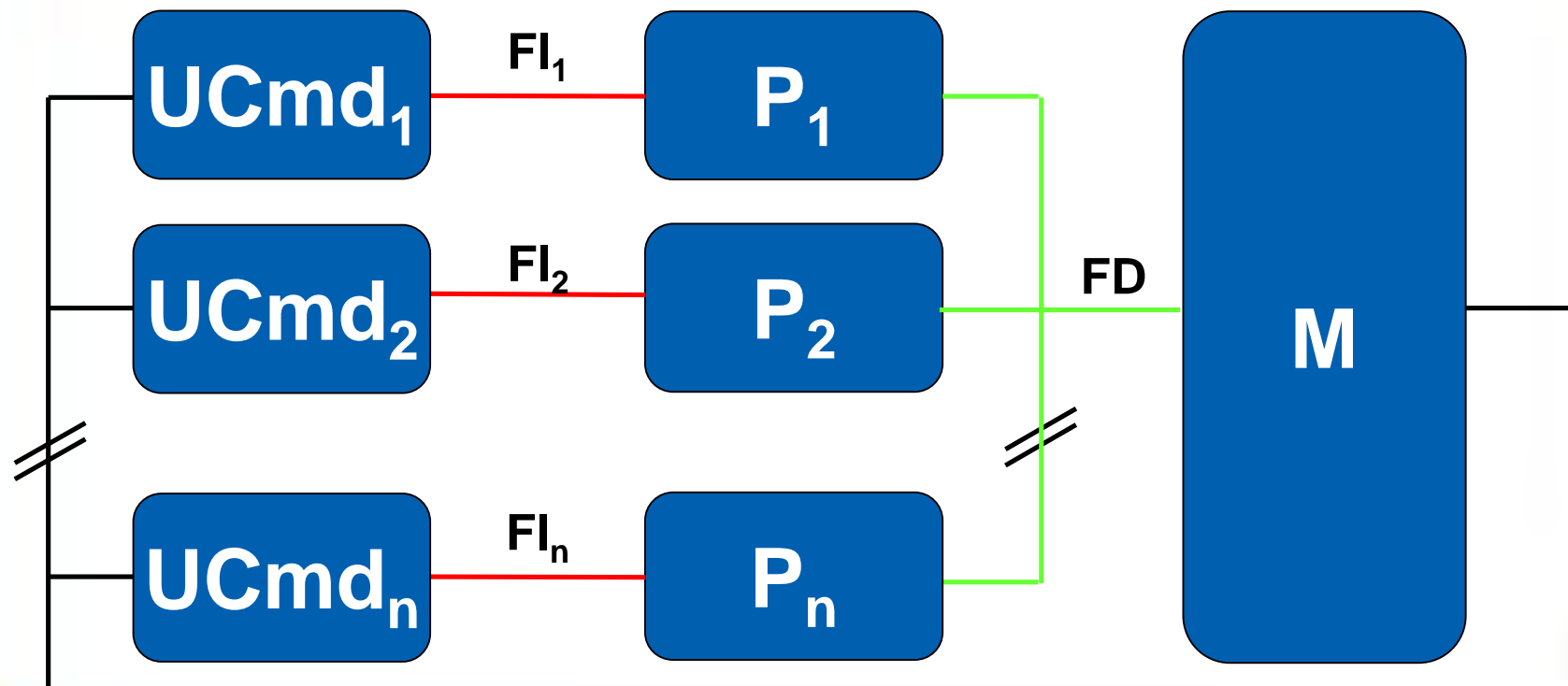


III – MISD

21

- MISD au:
 - Mai multe UCmd si procesoare
 - Un singur modul de memorie
- Domeniu de aplicatie restrans si special: aplicarea altor algoritmi pe aceleasi date (Apps: meteo/evidenta populatiei)

$$\text{MISD} = n \text{ FI} \& 1 \text{ FD}$$

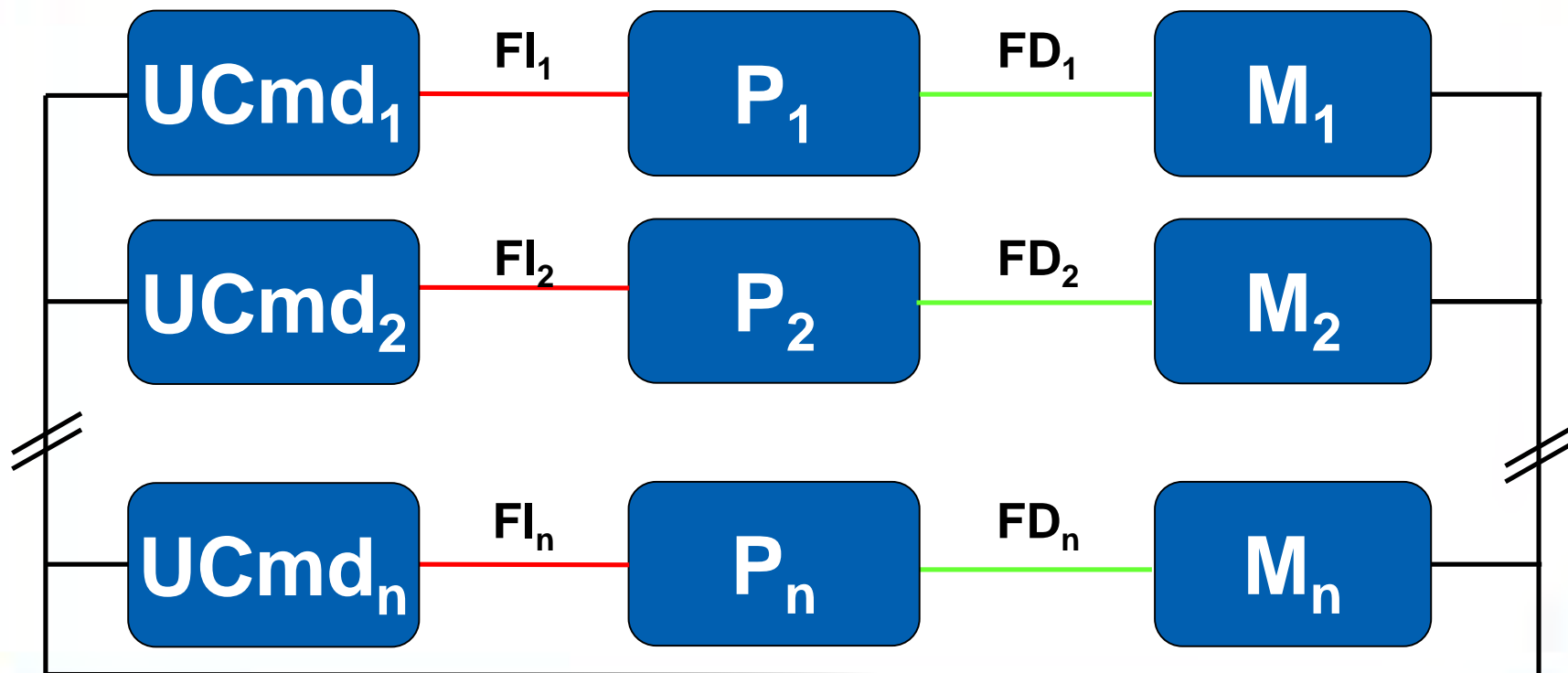




IV – MIMD

22

- MIMD pot comunica: (P-P sau P-M) **MIMD = n FI & n FD**
- Toate procesoarele participa la acelasi program
- Mod programare:
 - Shared memory (strans cuplate) – memorie partajata (e.g. OpenMP)
 - Distributed memory (slab cuplate) – transfer de mesaje (e.g. MPI)





- Structuri de calcul cu prelucrare paralela
- Clasificarea sistemelor de calcul / Flynn: SISD, SIMD, MISD, MIMD
- Exemple de utilizare a structurilor:
 - SISD
 - SIMD
 - MIMD
- Exemplu cu/fara dependenta de date pe sisteme de calcul MIMD



Exemplu de Utilizare – SISD

24

- Problema: $A[n, n]$, $B[n, n]$, $C = A \times B$
- Pe structuri de calcul SISD – 3 for-uri:

```
for i = 0 to n - 1
  for k = 0 to n - 1
    cik = 0
    for j = 0 to n - 1
      cik = cik + aij * bjk
    end j loop
  end k loop
end i loop
```

- Complexitatea acestui algoritm este... $O(n^3)$! → nesatisfacatoare! (mai ales dacă n e mare...)



Exemplu de Utilizare – SIMD

25

- Aceeasi problema: $A[n, n]$, $B[n, n]$, $C = A \times B$
- Avem n procesoare & toate executa aceeași instrucțiune odată → în fiecare calcul se calculează câte o linie și nu doar un element
- Considerând $(0 \leq k \leq n - 1)$ → se operează pentru toți indicii k simultan, adică se calculează pe linii

```
for i = 0 to n - 1
```

```
     $c_{ik} = 0$  ( $0 \leq k \leq n - 1$ )
```

```
    for j = 0 to n - 1
```

```
         $c_{ik} = c_{ik} + a_{ij} * b_{jk}$  ( $0 \leq k \leq n - 1$ )
```

```
    end j loop
```

```
end i loop
```

- Complexitatea acestui algoritm este... $O(n^2)$! → considerabil mai bine ca în cazul SISD



Comentarii & Observatii – SIMD

26

- Fiecare element al matricei produs C , este o suma ce se efectueaza secvential
- Cele n sume se calculeaza apoi in paralel !?
- a_{ij} NU depinde de $k \rightarrow$ accesul la aceasta memorie se face aproximativ secvential \rightarrow NU e chiar “sume se calculeaza in paralel”!
- Solutia: structurile SIMD trebuie sa dispuna de o instructiune de Broadcast & o retea de comutare (RC) ce sa asigure acest Broadcast
- P_j citeste a_{ij} prin RC (constanta a_{ij} e difuzata catre toate procesoarele)
- **Concluzie SIMD:** probleme mari la comunicatiile inter-procesoare & accesul si organizarea datelor!



Exemplu de Utilizare – MIMD

27

- Aceeasi problema: $A[n, n]$, $B[n, n]$, $C = A \times B$
- O UCmd/P trebuie sa preia functia de master: organizare si control + partajarea calculelor pe procesoare individuale
- Conway a propus o metoda cu 2 primitive: FORK & JOIN
 - FORK: desface un FI in n FI executabile simultan pe proc indep
 - JOIN: reuneste n FI intr-unul singur cand cele n FI s-au terminat

```
for k = 0 to n - 2 (nu si pt el insusi)
  FORK Adr
end k loop
```

```
Adr:
```

```
  for i = 0 to n - 1
     $c_{ik} = 0$  ( $0 \leq k \leq n - 1$ ) - pe coloane k fix
    for j = 0 to n - 1
       $c_{ik} = c_{ik} + a_{ij} * b_{jk}$  ( $0 \leq k \leq n - 1$ )
    end j loop
  end i loop
```

```
JOIN
```

Complexitatea este... $O(n^2)$!
→ la fel ca SIMD



Comentarii & Observatii – MIMD

28

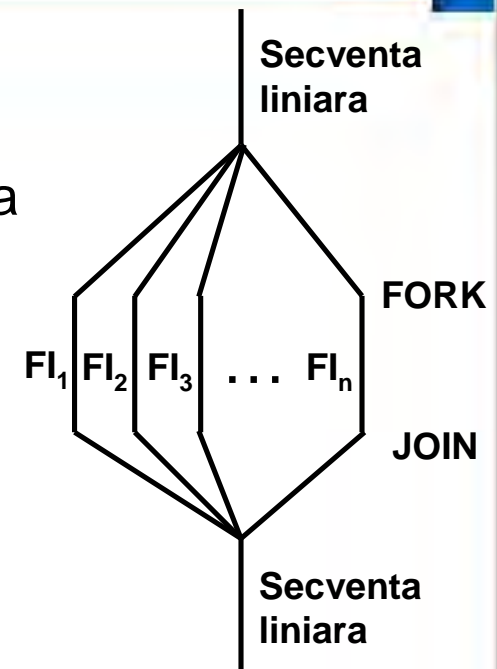
- In mod deliberat programul pt SIMD a fost scris a.i. actiunile P-urilor sa simuleze actiunile din structura MIMD
- Fiecare P_j calculeaza un C_{ik} in paralel
- Diferente SIMD/MIMD:
 - In SIMD procesoarele se sincronizau instructiune (It) cu It
 - In MIMD nu exista (neaparat) sincronizari intre FI ale P-urilor din structura
 - La SIMD se calculeaza elementele lui C pe linie si coloana (FI unic)
 - La MIMD orice procesoare pot calcula orice elemente din C (fiecare P are un FI propriu!)
- Castigul e acelasi; se pot folosi mai multe perechi FORK/JOIN
- Concluzie MIMD: mai usor de programat & utilizat decat SIMD – dar, mai scump! (Totul se plateste...)



Implementarea FORK & JOIN

29

- P_1 (master) pregateste taskurile intr-o coada de asteptare impreuna cu contextele asociate lor
- Celelalte procesoare $P_2 \dots P_n$ inspecteaza coada pana gasesc un task ce asteapta & il executa
- Daca numarul de procesoare = numarul de procese
→ se executa simultan toate
- Daca numarul de procesoare \ll numarul de procese
→ dupa executia unui proces, un procesor preia din coada un nou proces pt exec
- **Atentie:** aici procesoarele sunt considerate omogene si procesele independente intre ele (caz f rar in realitate!)
- Instructiunea JOIN n:
 - Pentru fiecare FI exista un contor unic ce e initializat cu 0 la inceput
 - Pentru fiecare FI contorul este incrementat si comparat cu n
 - Trebuie ca toate procesele sa se fi terminat (pana la n) si abia apoi se continua cu urmatoarea secventa liniara pe P_1 (master)





- Structuri de calcul cu prelucrare paralela
- Clasificarea sistemelor de calcul / Flynn: SISD, SIMD, MISD, MIMD
- Exemple de utilizare a structurilor:
 - SISD
 - SIMD
 - MIMD
- Exemplu cu/fara dependenta de date pe sisteme de calcul MIMD



Exemplu fara dependenta de date

31

- Patru procesoare P_1, P_2, P_3, P_4 si 10 procese $Proc_i$ ($i = 1 \dots 10$)
- Modelul de executie este:

```
for i = 0 to 9
```

```
  FORK Adr
```

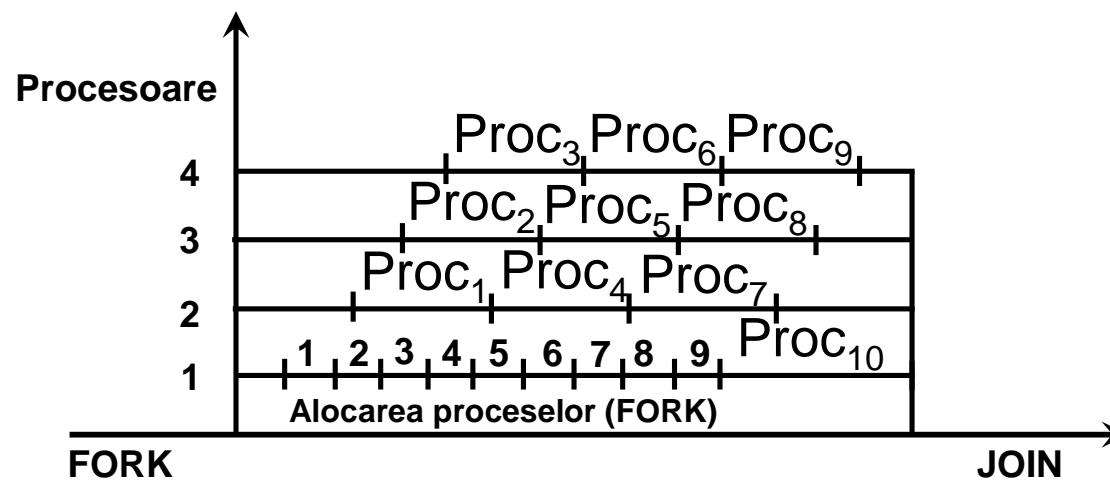
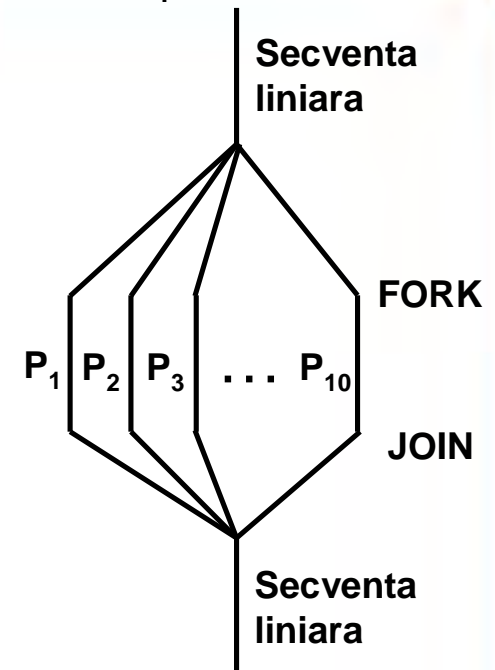
```
end i loop
```

```
Adr:
```

```
  Proci
```

```
  JOIN
```

Coada de asteptare a proceselor:

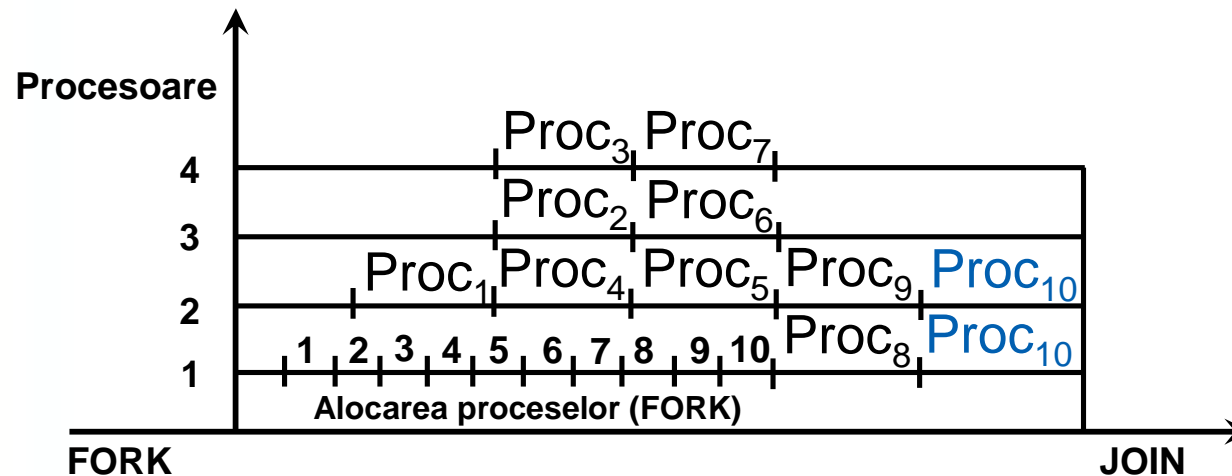
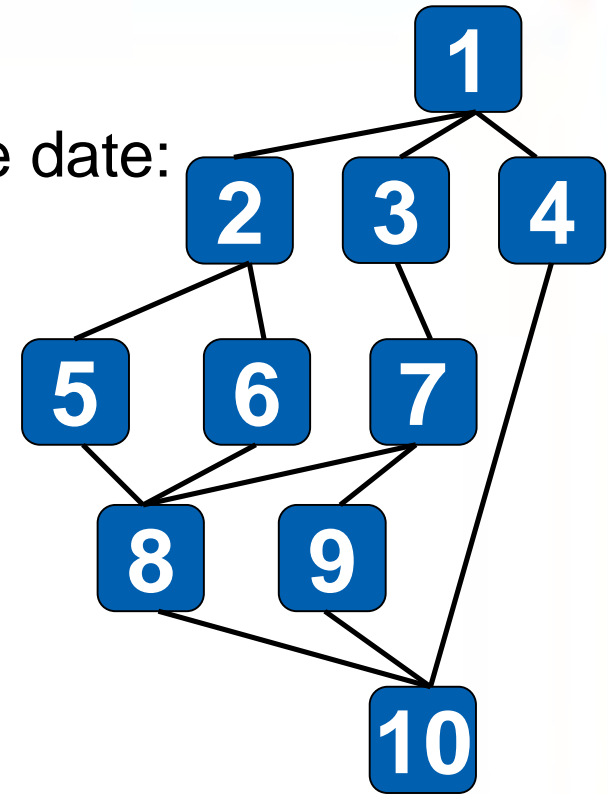




Exemplu cu dependenta de date

32

- Patru procesoare P_1, P_2, P_3, P_4 si 10 procese $Proc_i$ ($i = 1 \dots 10$)
- Aceeasi coada de asteptare
- Presupunem urmatoarea dependenta de date:
- Lucrurile stau altfel: exista procesoare ce asteapta datorita dependentei de date





Concluzii Dependenta de Date

33

- In exemplele prezentate nu apar restrictii de precedenta
- Planificarea FIFO nu e necesar cea mai potrivita → alocare statica
- Alocarea dinamica e mai buna → mai ales cand \exists procesoare specializate
- Programe de uz general → 10% paralelism
- Programe dedicate → 90% paralelism
- Dependenta de date → sincronizari in timp
- Eficienta MIMD depinde de:
 - Eficienta algoritmilor de planificare
 - Consumul de timp impus de precedenta de date
- Problema repartizarii pe procesoare → NP completa!



What Next?

34

- Q & A?
- Next time:
 - The Cell/B.E.:
 - Motivation
 - Architecture
 - Programming
 - Applications
 - Performance
 - Summary & Conclusions
 - Cell Resources & Documentation