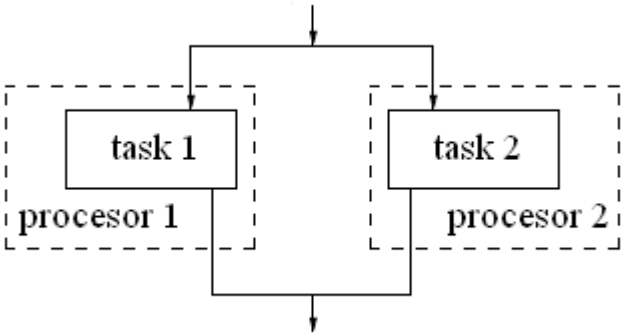
Curs 2 AAC

ARHITECTURI PARALELE DE CALCULATOARE
NOŢIUNI INTRODUCTIVE

Prelucrarea paralelă reprezintă utilizarea mai multor procesoare pentru execuţia simultană a mai multor părţi dintr-un program (task-uri).



Avantaje

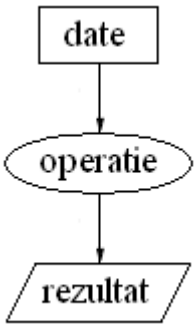
- Se elimină limitarea de viteză a procesoarelor secvenţiale (timp de execuţie mai scurt).
- Se elimină limita de miniaturizare a procesoarelor secvenţiale (nu putem construi un procesor complex la dimensiuni oricât de mici).
- Se elimină limitarea economică (imposibilitatea de a fabrica ieftin un procesor foarte rapid => e mai bine să se folosească două sau mai multe procesoare ieftine care să ruleze în paralel pe acelaşi calculator).

Evoluţia

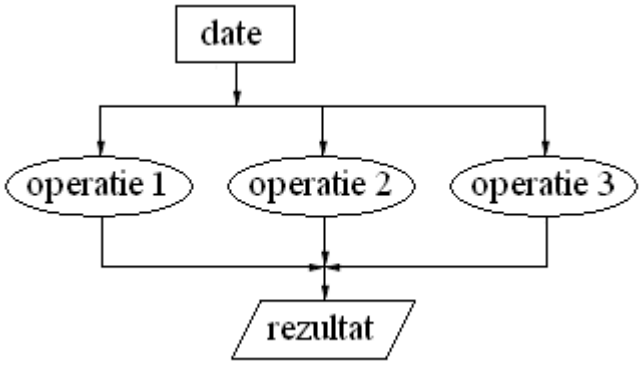
- Procesoare secvenţiale obişnuite, numite procesoare scalare: execută maxim o instrucţiune într-un ciclu procesor.
- Procesoare superscalare: pot executa mai multe instrucţiuni într-un ciclu procesor.
- Procesoare vectoriale: nu mai lucrează pe operanzi scalari ci pe structuri de date multiple: vectori uni / multi dimensionali.
- Arhitecturi multiprocesor: într-un singur cip există mai multe nuclee de procesare (dual core / quad core /octa core).

Clasificarea arhitecturilor de calculatoare

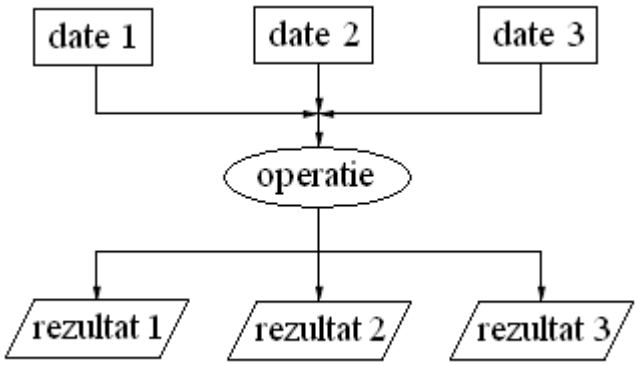
- **SISD** (*Single Instruction Single Data*): arhitectura secvenţială obişnuită.



- **MISD** (*Multiple Instruction Single Data*): un program e descompus în mai multe task-uri sau secvenţe de operaţii; operaţiile se execută în paralel asupra aceluiaşi set de date.

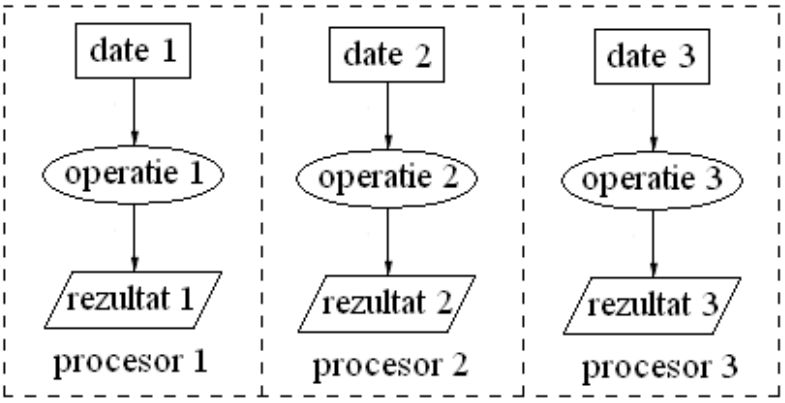


- **SIMD** (*Single Instruction Multiple Data*): un acelaşi program se execută pe mai multe seturi de date.



ex: prelucrari de masive (siruri, matrici); procesoare vectoriale

- **MIMD** (*Multiple Instruction Multiple Data*): reprezintă tipul general de arhitectură pentru calculatoarele paralele; se realizează operaţii diferite asupra unor seturi de date diferite.



Viteza de execuţie a programelor paralele

Acceleraţia: expresie a vitezei de execuţie: $Sp = \frac{Tes}{Tep}$ unde:

Tes – timpul de execuţie secvenţial;
Tep – timpul de execuţie paralel pe mai multe procesoare al aceluiaşi program.

Ideal: $Sp = \frac{Tes}{Tes/P} = P$ (numărul de procesoare)

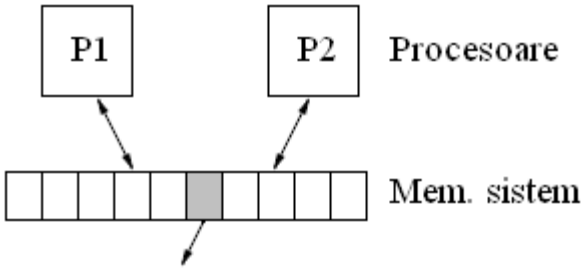
Real: $Tep \neq Tes/P$; $Tep = T\,\text{secv} + Tparalel$ (o porţiune din program se poate executa doar secvenţial în timpul Tsecv şi avem o porţiune care se poate executa în paralel)

$$\Rightarrow Sp = \frac{Tes}{T\,\text{secv} + Tparalel} \Rightarrow Sp < \frac{Tes}{T\,\text{secv}} = K$$ **- Legea lui Amdahl**

Oricât am creşte numărul de procesoare într-un sistem, acceleratia (viteza de execuţie în paralel) a unui program e întotdeauna limitată superior.

Accesul la memorie în procesarea paralelă

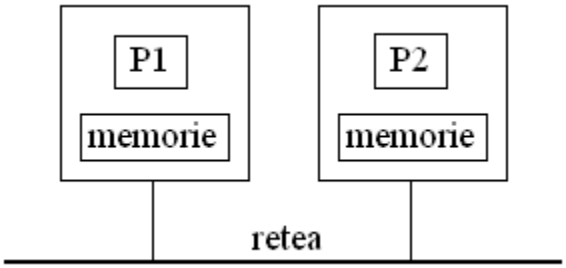
a) Acces partajat la memorie.
In acest caz, procesoarele împart acelaşi spaţiu de memorie.



La un anumit moment, o anumită locaţie din memoria partajată poate fi accesată doar de către un singur procesor.

Avantaj: realizarea unui schimb rapid de date între 2 task-uri care rulează pe 2 procesoare diferite.
Exemplu: arhitectura UMA – procesoarele accesează în mod identic orice locaţie din memoria partajată a sistemului; arhitectura Pentium multicore; arhitectura GPU.

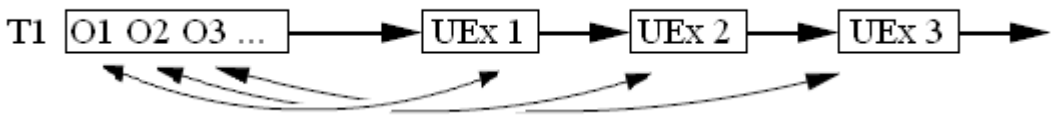
b) Acces distribuit la memorie.



Procesoarele pot avea acces atât la o memorie locală aflată pe calculatorul din care fac parte, cât şi la o memorie aflată la distanţă (alt calculator).
Exemplu: arhitectura NUMA – procesoarele au acces rapid la memoria locală şi acces lent la memoria aflată la distanţă; arhitectura COMA – în care memoriile locale ale procesoarelor sunt organizate ca nişte memorii cache.

Exemple de arhitecturi paralele

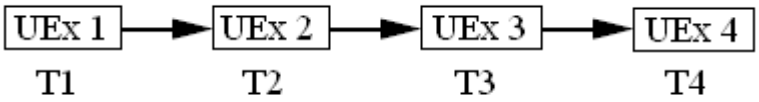
1) **Arhitectura Pipeline** – structură cu un singur procesor, dar care poate executa mai multe operaţii în paralel; astfel un anumit task se descompune într-o serie de operaţii care se execută independent de către unităţi de execuţie specializate.



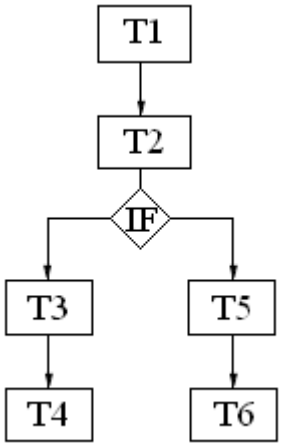
Există 2 tipuri de Pipeline: - de instrucţiuni; în care o instrucţiune se descompune într-o serie de microinstrucţiuni.
- aritmetic; în care o operaţie aritmetică se descompune într-o serie de microoperaţii.
În rezolvarea unui task pot exista anumite situaţii speciale, în care fluxul de operaţii din Pipeline se opreşte. Aceste situaţii speciale se numesc hazarduri (incidente) şi sunt de 3 tipuri:

a) Hazardul structural. El e generat de faptul că nu există destule unităţi de execuţie pentru operaţiile planificate.
b) Hazardul de date. El e generat din cauza dependenţelor de date. Un anumit task aşteaptă după datele unui alt task.
c) Hazardul de control. El e datorat ramificărilor care apar în program (instrucţiunile de salt).

De exemplu, dacă în structura Pipeline se încarcă la început patru operaţii:



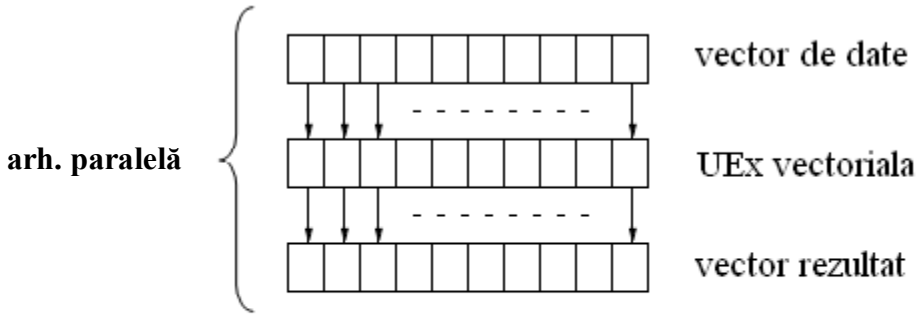
în cazul schimbării fluxului de instrucţiuni din program, instrucţiunile încărcate în avans în structura Pipeline (T3 şi T4) trebuie eliminate şi înlocuite cu instrucţiunile corecte (T5 şi T6).

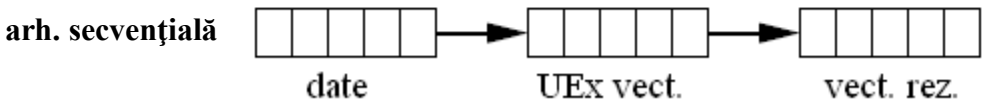


2) **Arhitecturi cu legături multiple** – acestea sunt structuri multiprocesor conectate în diverse configuraţii în care fiecare procesor realizează anumite operaţii simple într-un timp foarte scurt.

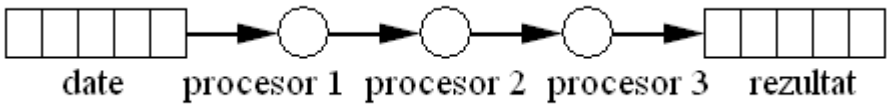
a) **Arhitecturi vectoriale**.
Exemple:

- **Procesoare vectoriale**



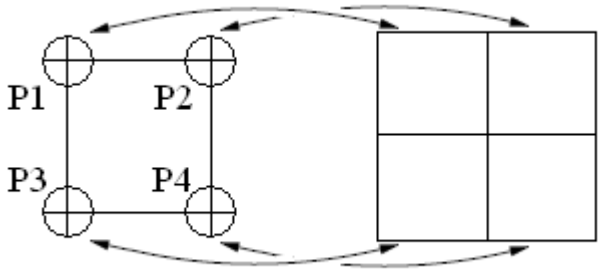


- **Procesoare sistolice**: la fiecare tact datele avansează o poziţie prin structura sistolică.



Există un şir de elemente de prelucrare simple, conectate în diferite topologii; elementele de prelucrare pot avea o memorie locală şi pot fi de tipul SIMD sau nu (fiecare element poate efectua aceeaşi sau o altă operaţie).

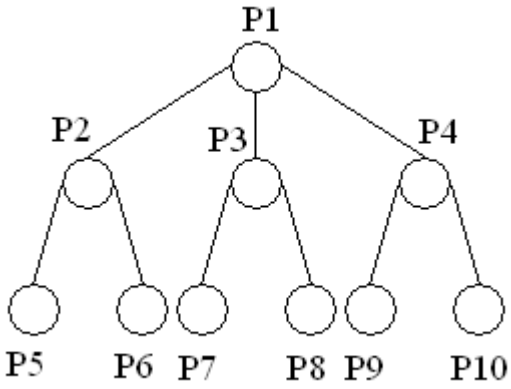
b) **Arhitecturi de tip reţea**.
Exemplu: - procesoare matriciale



Unităţile de prelucrare sunt autonome şi au o memorie locală ataşată.

De obicei există o unitate de control care distribuie sarcinile la structura matricială de procesare.

c) **Arhitectura de tip arbore**. Se foloseşte la calcule ştiinţifice / economice, programe de decizie.



d) **Arhitectura hipercub**. E o arhitectură mixtă între reţea şi arbore. Exemplu: fiecare nod al reţelei poate fi un arbore.

Granularitatea sistemelor multiprocesor

Granularitatea este rezoluţia (gradul de fineţe) cu care e rezolvată o anumită sarcină. În cazul de faţă, sarcina e execuţia paralelă a unui program $\Rightarrow$ granularitatea indică gradul de paralelizare pentru acel program.
Există 3 tipuri:

- Granularitate mică (grosieră) – realizarea paralelismului se face la nivel de proces.
- Granularitate medie – paralelismul se realizează la nivel de *thread* (fir de execuţie din interiorul unui proces alcătuit din mai multe instrucţiuni care utilizează aceleaşi resurse).
- Granularitate mare (fină) – paralelismul se realizează la nivel de instrucţiune.