Curs 2

Programare Paralela si Distribuita

- Arhitecturi paralele
- Clasificarea sistemelor paralele
- *Cache Consistency*
- Top 500 Benchmarking

Clasificarea sistemelor paralele
-criterii-

Resurse

- numărul de procesoare şi puterea procesorului individual;
- Tipul procesoarelor – omogene- heterogene
- Dimensiunea memoriei

Accesul la date, comunicatie si sincronizare

- complexitatea reţelei de conectare şi flexibilitatea sistemului
- distribuţia controlului sistemului, adică dacă multimea de procesoare este condusa de catre un procesor sau dacă fiecare procesor are propriul său controller;
- Modalitatea de comunicare (de transmitere a datelor);
- Primitive de cooperare (abstractizari)

Performanta si scalabilitate

- Ce performanta se poate obtine?
- Ce scalabilitate permite?

Clasificarea Flynn

Michael J. Flynn în 1966

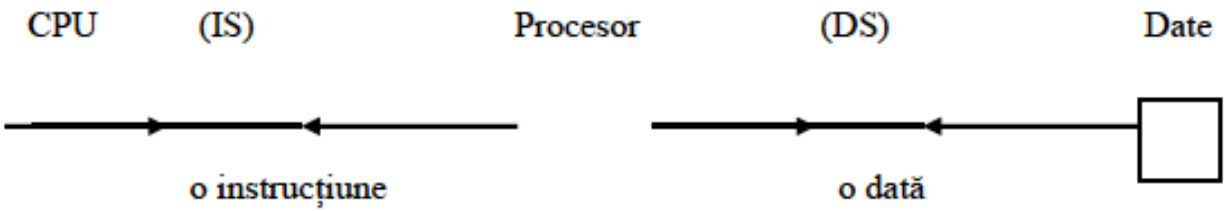
- SISD: sistem cu un singur flux de instrucţiuni şi un singur flux de date;
- SIMD: sistem cu un singur flux de instrucţiuni şi mai multe fluxuri de date;
- MISD: sistem cu mai multe fluxuri de instrucţiuni şi un singur flux de date;
- MIMD: cu mai multe fluxuri de instrucţiuni şi mai multe fluxuri de date.

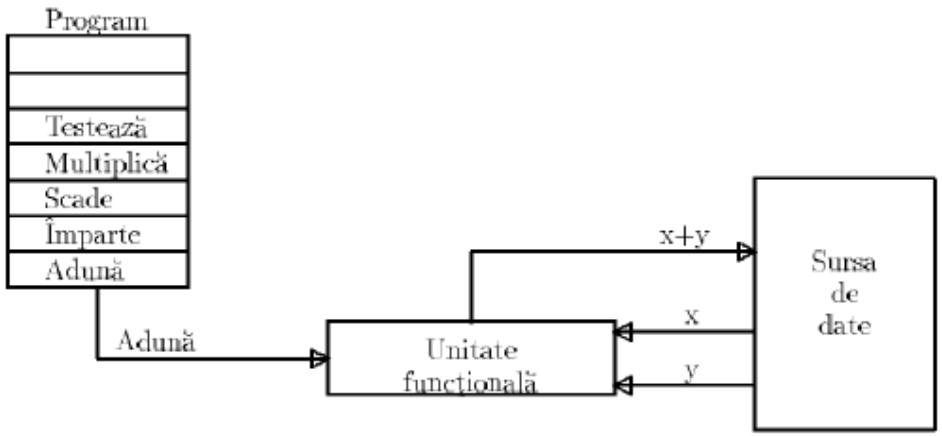
(imagini urm. preluate din ELENA NECHITA, CERASELA CRIŞAN, MIHAI TALMACIU, ALGORITMI PARALELI SI DISTRIBUIŢI)

SISD

Flux de instrucțiuni singular, flux de date singular (SISD)-

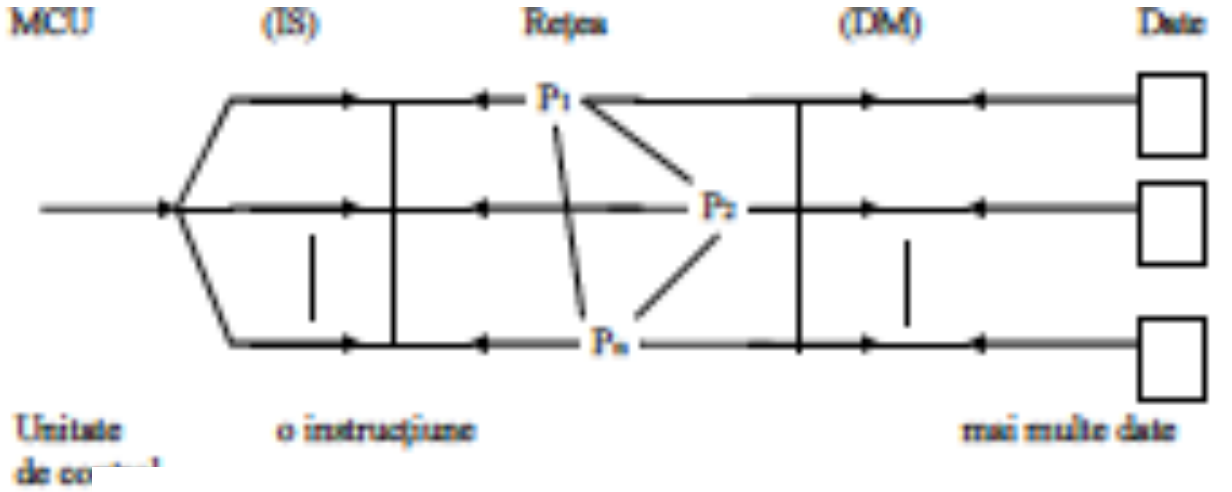
- microprocesoarele clasice cu arhitecturi von Neumann
- Functionare ciclica: preluare instr., stocare rez. in mem. , etc.

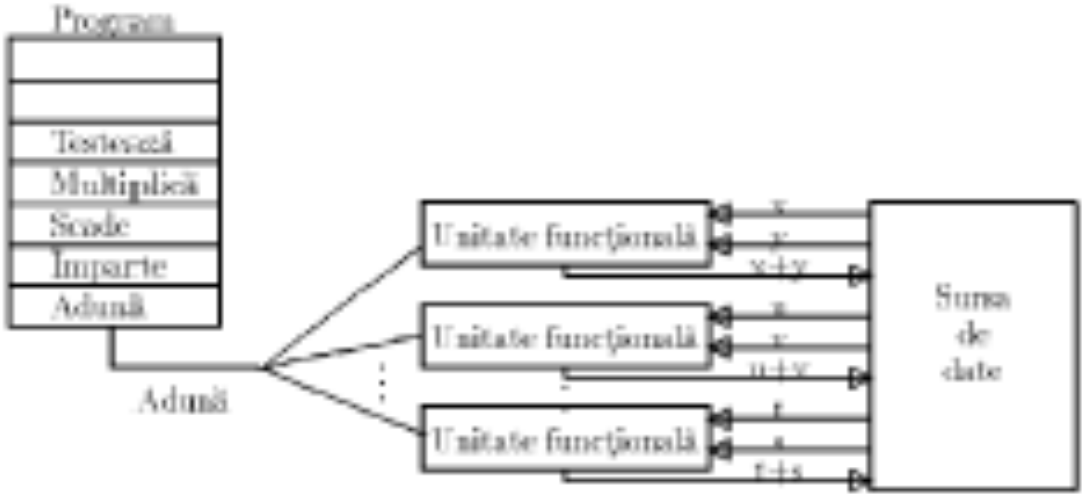


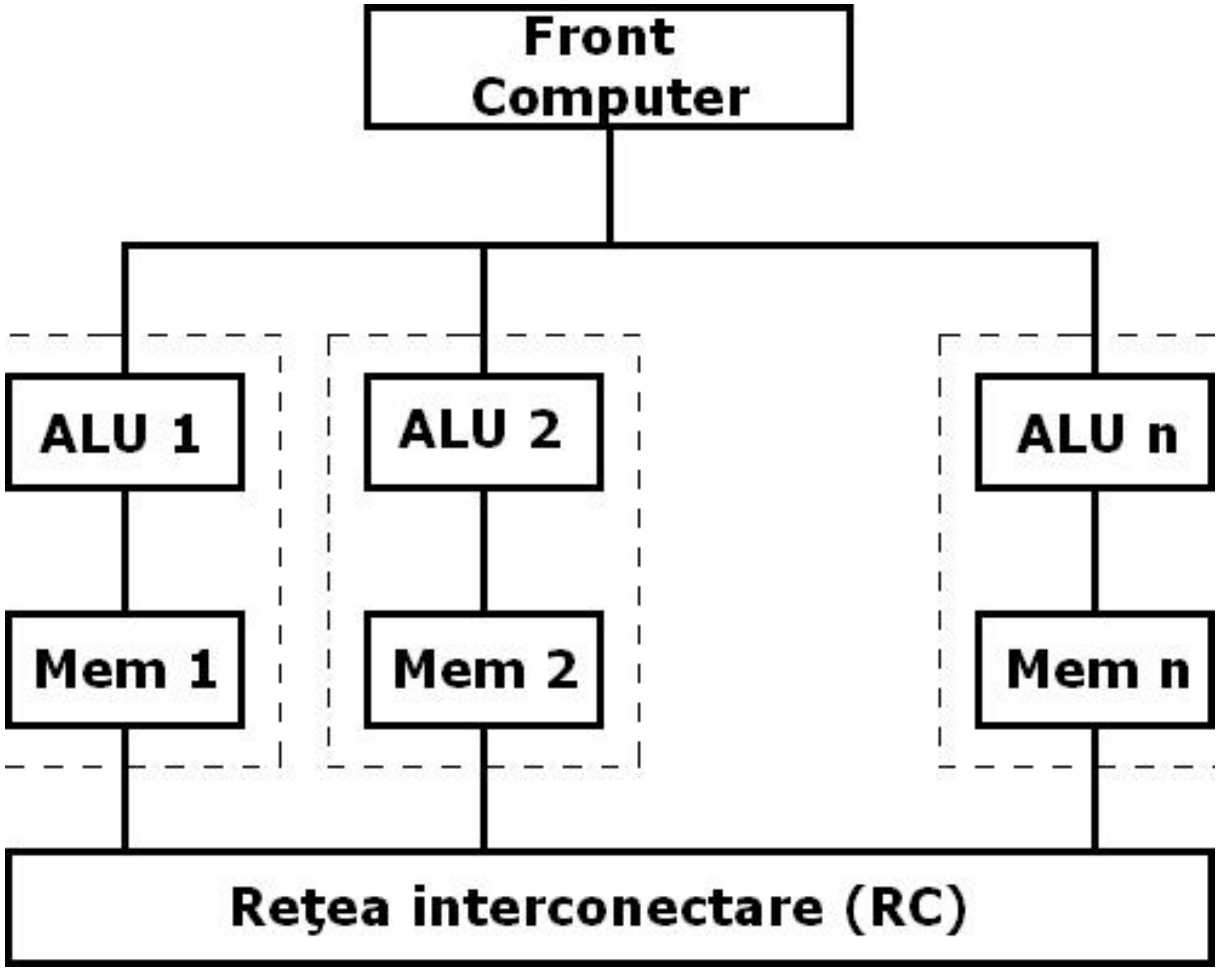


SIMD

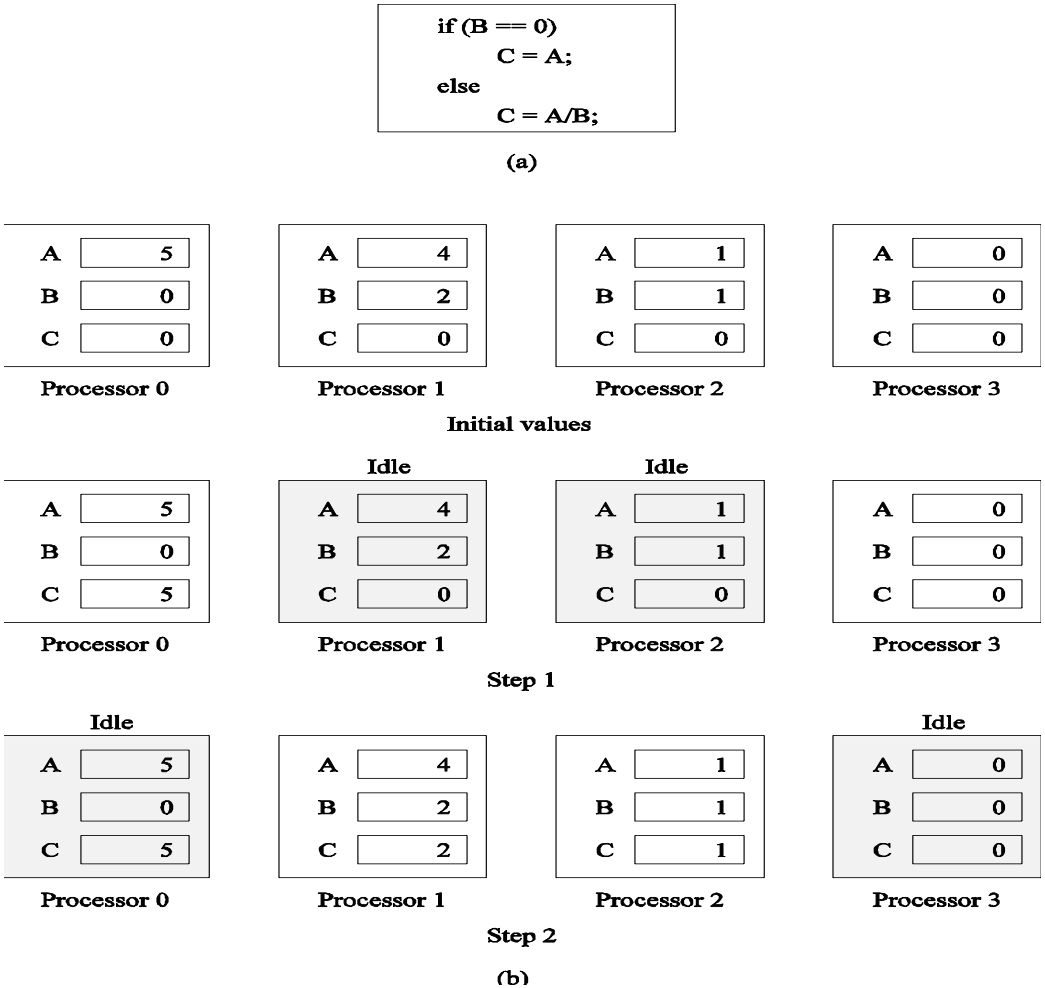
Flux de instrucțiuni singular, flux de date multiplu






Front
Computer
ALU 1
ALU 2
ALU n
Mem 1
Mem 2
Mem n
Reţea interconectare (RC)

Executie conditionala in SIMD Processors

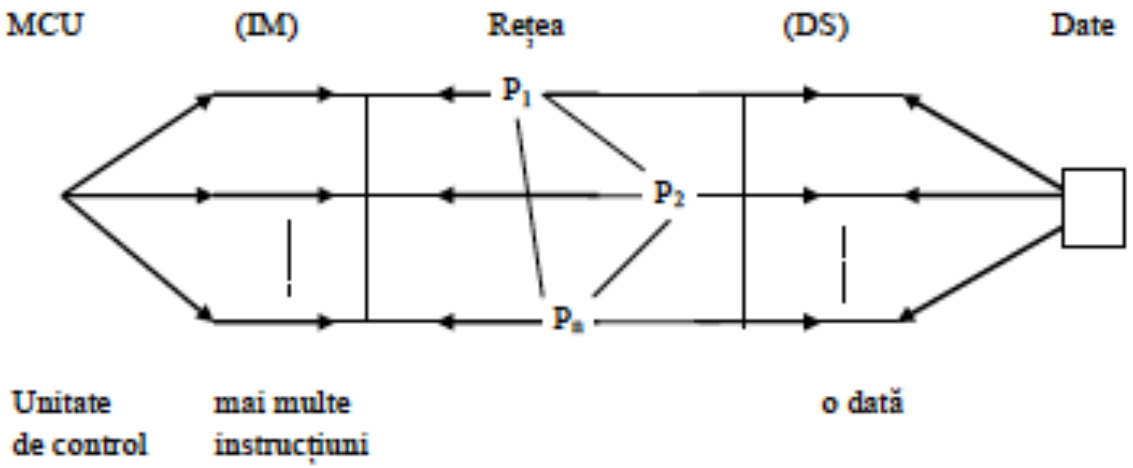


MISD

Flux de instrucțiuni multiplu, flux de date singular

? procesoare pipeline:

intr-un procesor pipeline o singura data este operata de diferite faze ale unei unităţi funcţionale, paralelismul fiind realizat prin simultana execuţie a diferitelor etape asupra unui şir de date secvenţiale



Arhitectura sistolica

Orchestrate data flow for high throughput with less memory access

Different from pipelining

Nonlinear array structure, multidirection data flow, each PE may have (small) local instruction and data memory

Different from SIMD

Each PE may do something different

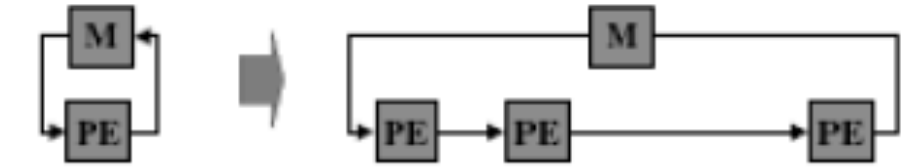
Initial motivation

VLSI enables inexpensive special-purpose chips

Represent algorithms directly by chips connected in regular pattern

Very-large-scale integration

Systolic Architectures



Replace a processing element(PE) with an array of PE's without increasing I/O bandwidth

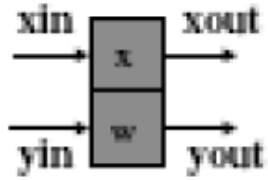
Exemplu: matrix-vector multiplication

$$y_i = \sum_{j=1}^{n} a_{ij} x_j, i = 1, \ldots, n$$

Recursive algorithm

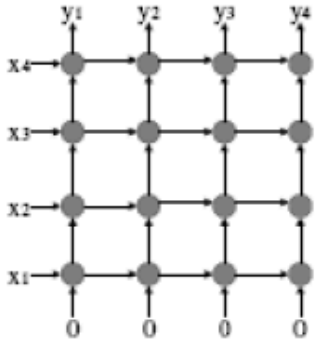
```
for i = 1 to n
  y(i,0) = 0
  for j = 0 to n
    y(i,0) = y(i,0) + a(i,j) * x(j,0)
```

Use the following PE



xout = x
x = xin
yout = yin + w * xin

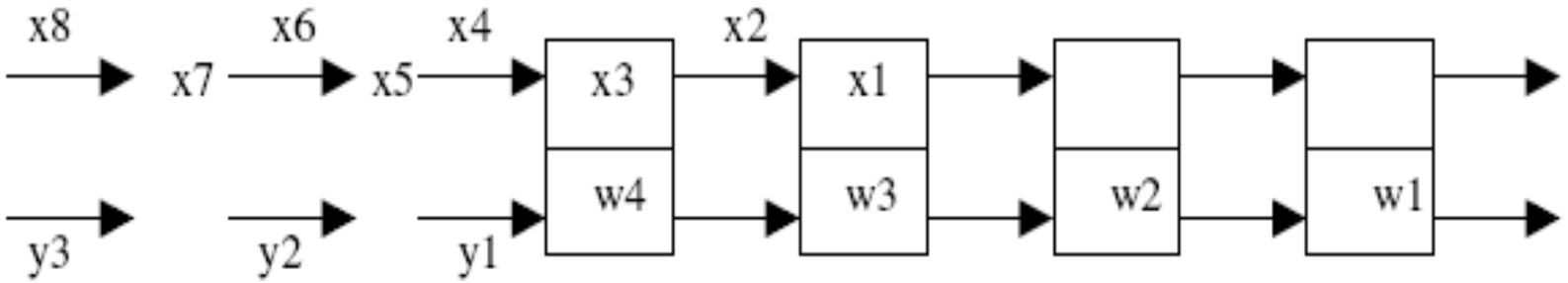
Systolic Array Representation of Matrix Multiplication

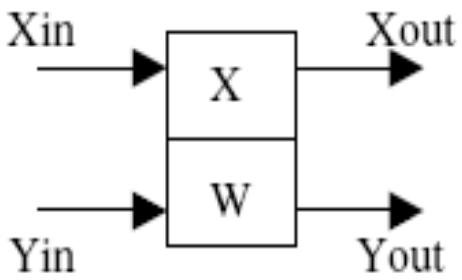


Exemplu – retea liniara

Exemplu: se consideră un sistem simplu pentru calcularea convoluţiilor, utilizând o reţea liniară de elemente de prelucrare:

$$y(i) = w1*x(i)+ w2*x(i+1)+w3*x(i+2)+w4*x(i+3)$$



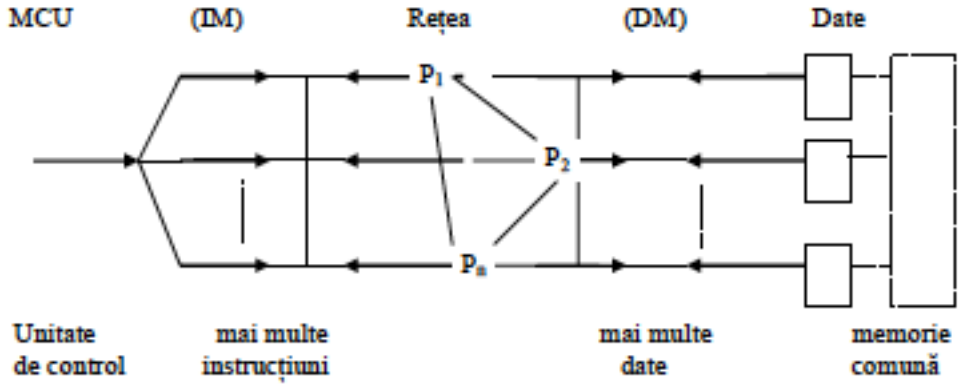


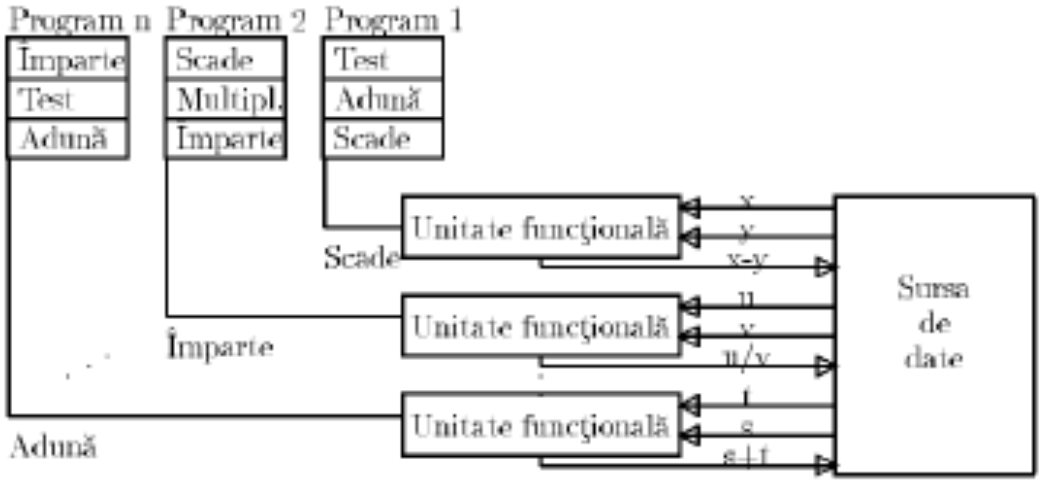
Xout=X
X=Xin
Yout=Yin+W*Xin

Reţea liniară pentru calcularea convoluţiilor.

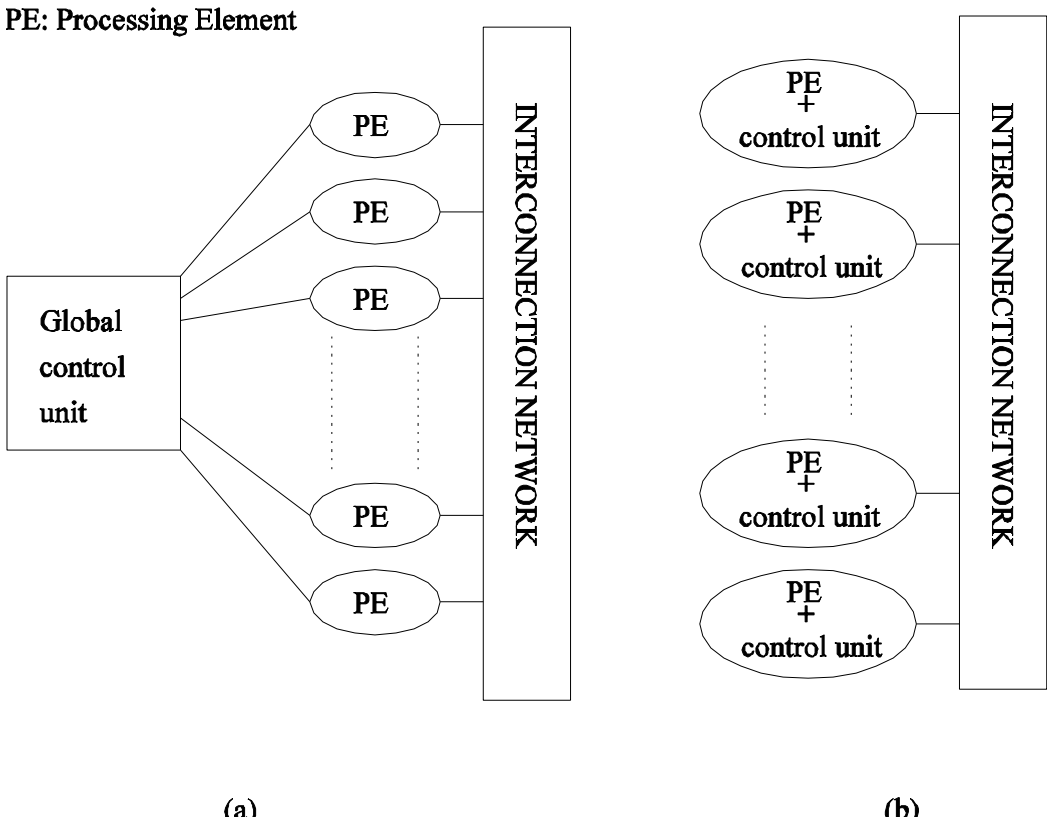
MIMD

Flux de instrucțiuni multiplu, flux de date multiplu

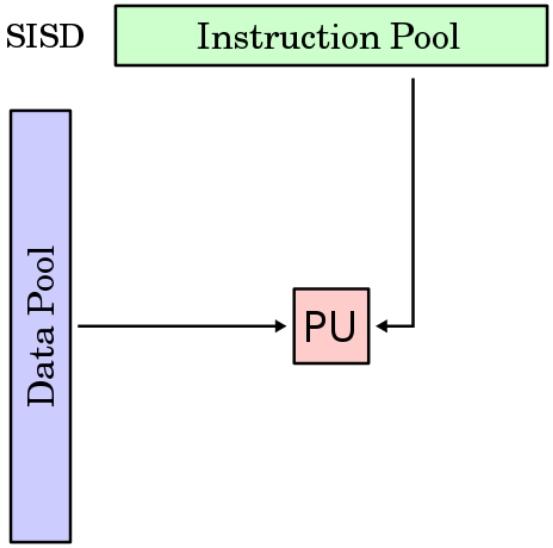


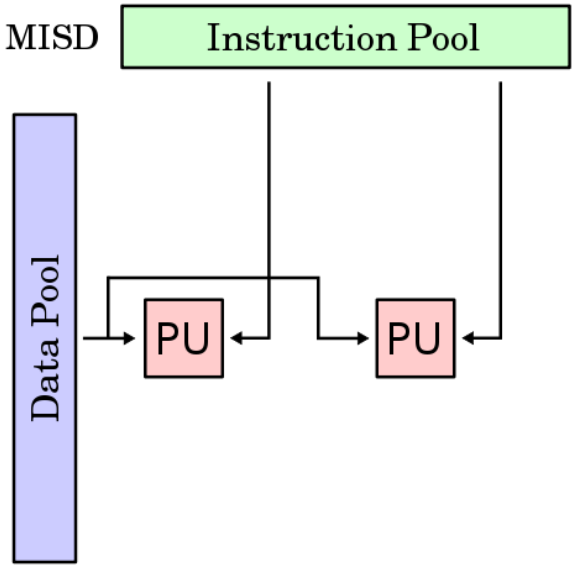


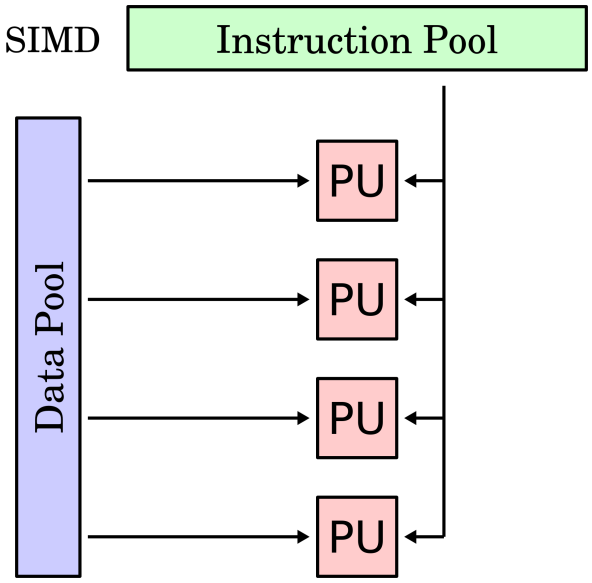
SIMD versus MIMD

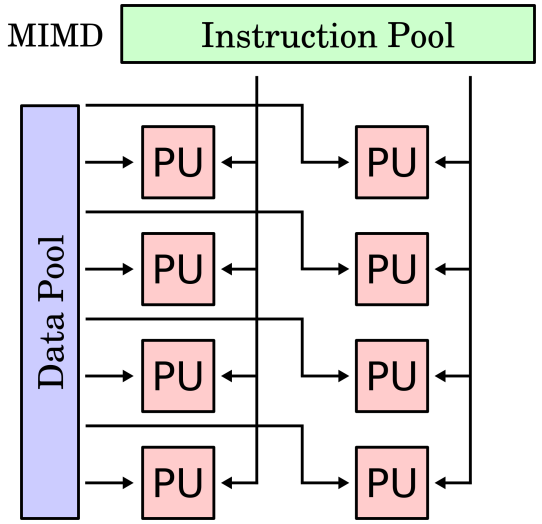


Scheme Comparative





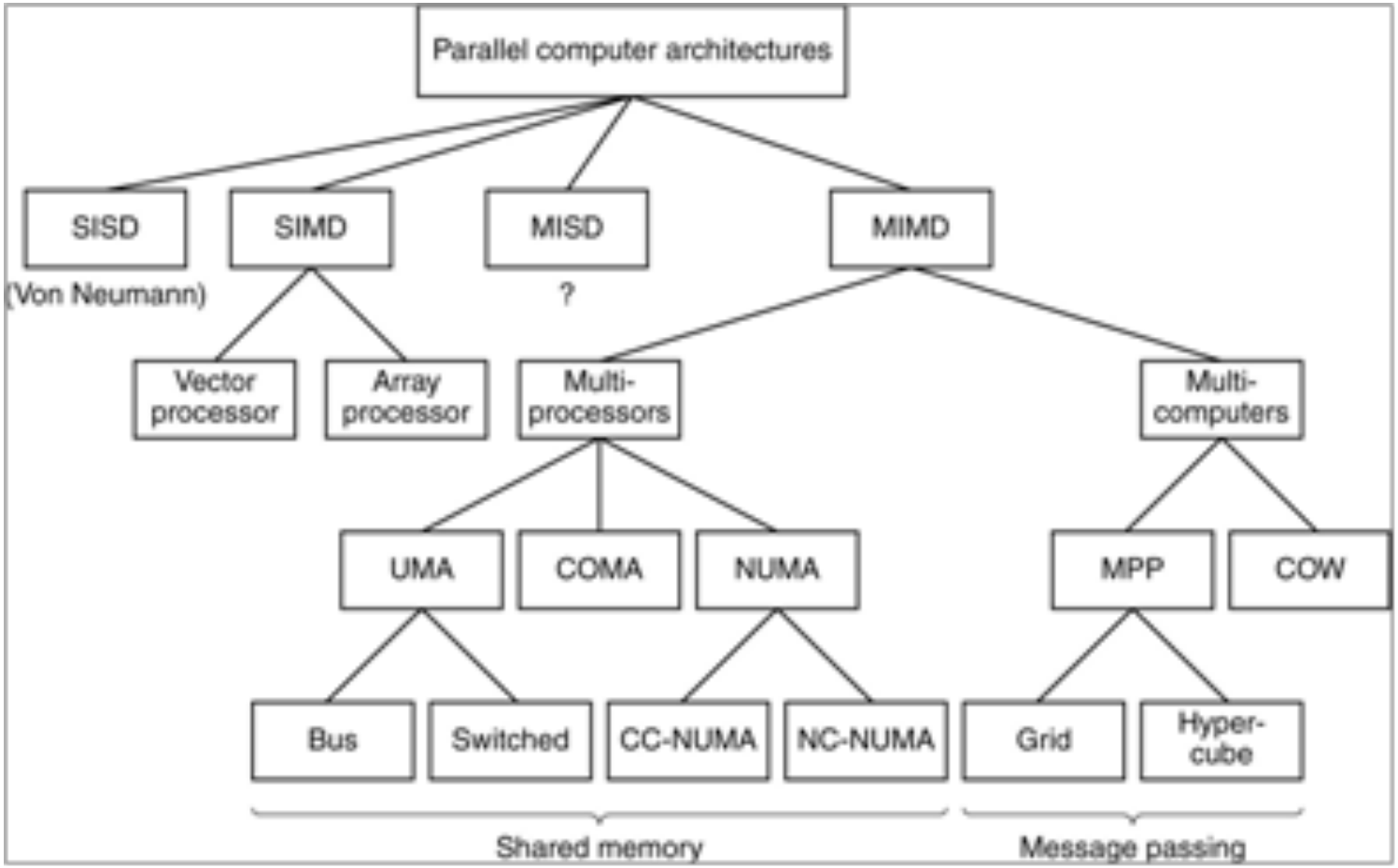




Paralelizare la nivel hardware – istoric

- Etapa 1 (1950s): executie secv a instructiunilor
- Etapa 2 (1960s): sequential instruction issue
  - Executie Pipeline,
  - *Instruction Level Parallelism* (ILP)
- Etapa 3 (1970s): procesoare vectoriale
  - Unitati aritmetice care fol. Pipeline
  - Registrii, sisteme de memorie paralele *multi-bank*
- Etapa 4 (1980s): SIMD si SMPs
- Etapa 5 (1990s): MPPs si clustere
  - *Communicating sequential processors*
- Etapa 6 (>2000): many-cores, acceleratori,

Vedere generala



Modelul Data Flow

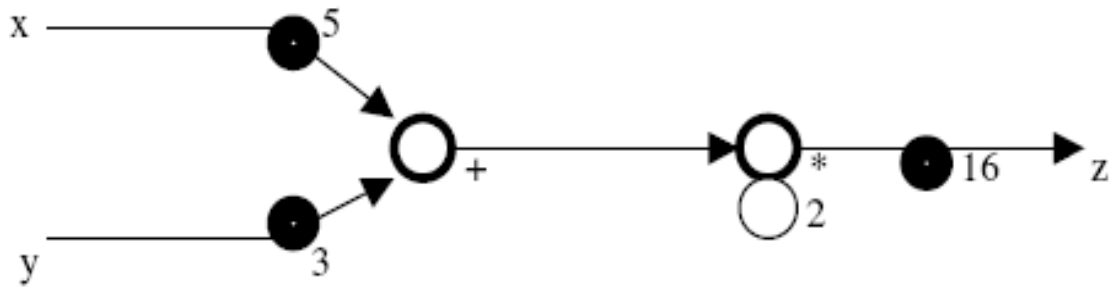
- Modelul de procesare data flow specifica efectuarea unei operatii imediat ce operanzii devin disponibili. Aceasta elimina necesitatea existentei contorului de program. Rezultatul produs de o instructiune este utilizat ca o data pura sau token si este furnizat direct urmatoarei instructiuni.

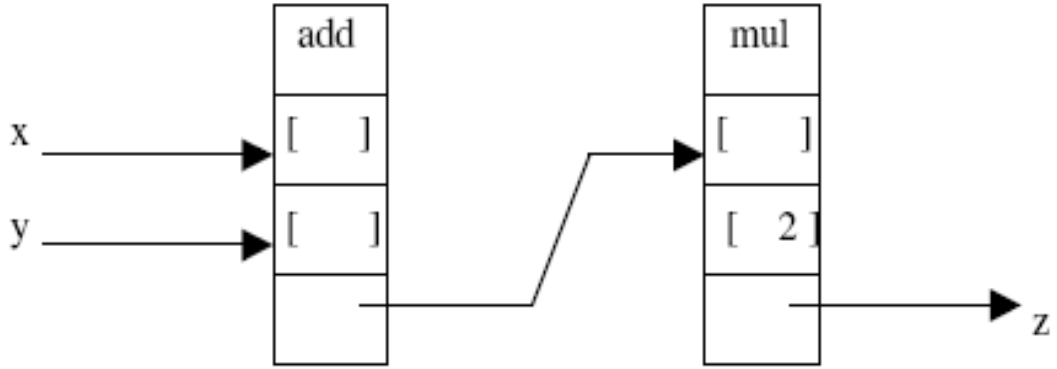
- Masinile data flow nu necesita adrese pentru memorarea variabilelor ceea ce contrasteaza complet cu masinile Von Neumann.

- Operatia se efectueaza numai cind token-ul este prezent pe ambele intrari ale operatorului

Arhitectura DataFlow (Flux de date)

Exemplu: se consideră calcularea expresiei: $z = (x + y) * 2$





A program is represented as a directed acyclic graph
a node = an instruction
an edge = data dependency

• Firing rule: A node can be scheduled for execution if and only if its input data become valid for consumption

. *Graful si şabloanele în flux de date.*

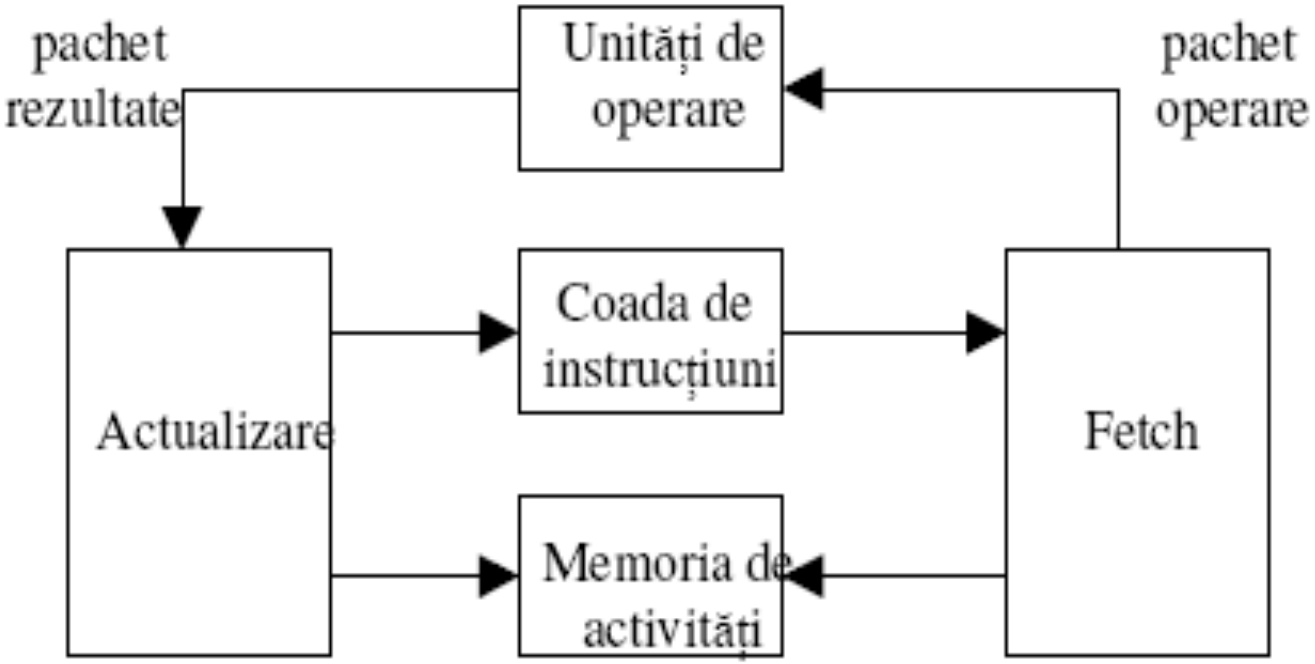
Data-Flow *computer*

Execution of instructions is driven by data availability
Basic components
Data are directly held inside instructions
Data availability check unit
Token matching unit
Operations can be dispatched to processors
Tokens carry tags of next instruction to processor
Tag compared in matching store
A match fires execution
Machine does the hard parallelization work
Chain reaction of asynchronous instruction executions

Advantages

Very high potential for parallelism
High throughput
Free from side-effect

Disadvantages -> Hard to build correctly
Time lost waiting for unneeded arguments
High control overhead
Difficult in manipulating data structures



Mecanismul de execuţie al unui program în flux de date.

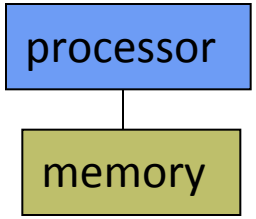
SUMARIZARE

Vedere actuala asupra tipurilor de arhitecturilor paralele

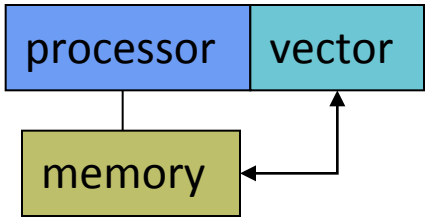
Parallel Architecture Types

from course pres. Introduction to Parallel Computing
CIS 410/510, Univ. of Oregon

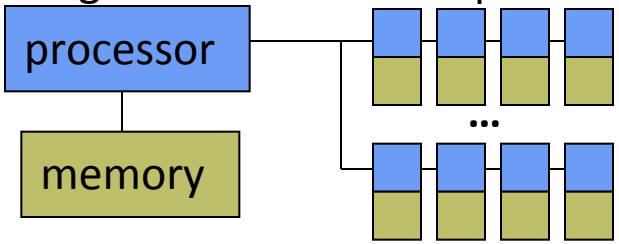
- Uniprocessor
  - Scalar processor



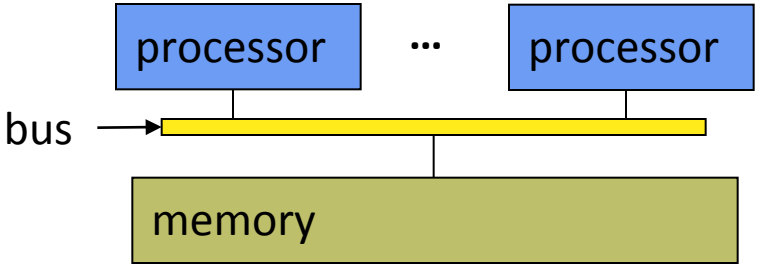
  - Vector processor



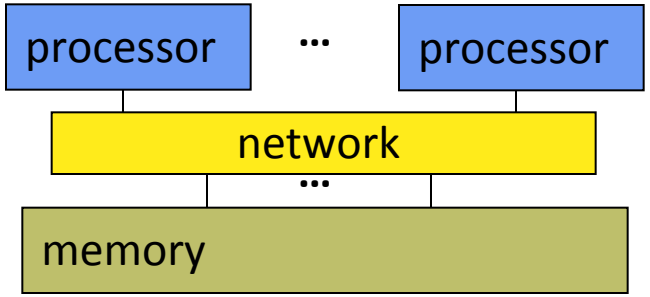
  - Single Instruction Multiple Data



- Shared Memory Multiprocessor (SMP)
  - Shared memory address space
  - Bus-based memory system

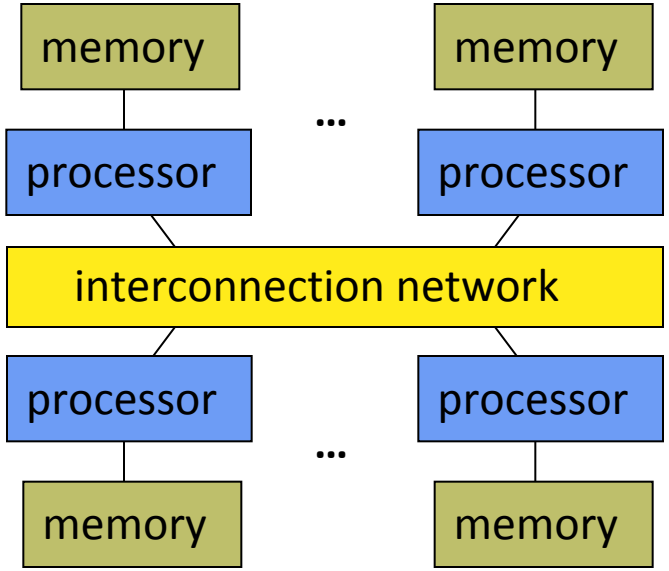


  - Interconnection network



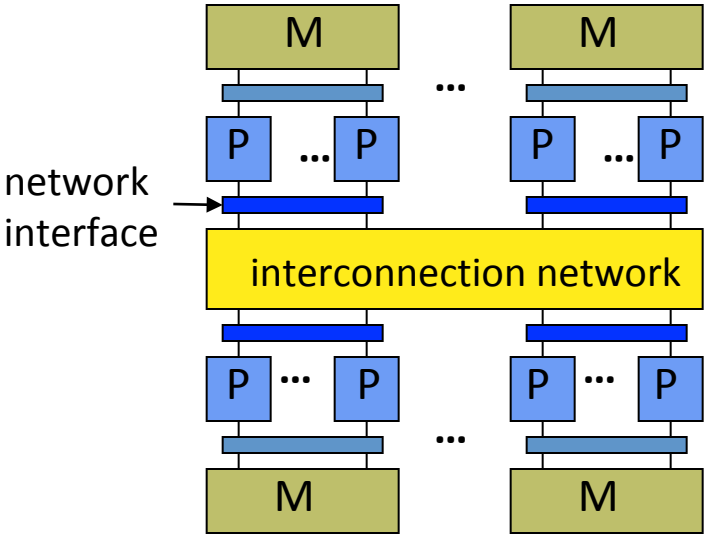
Parallel Architecture Types (2)

- Distributed Memory Multiprocessor
  - Message passing between nodes



  - Massively Parallel Processor (MPP)
    - Many, many processors

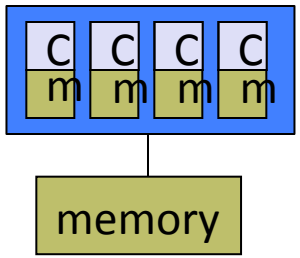
- Cluster of SMPs
  - Shared memory addressing within SMP node
  - Message passing between SMP nodes



  - Can also be regarded as MPP if processor number is large

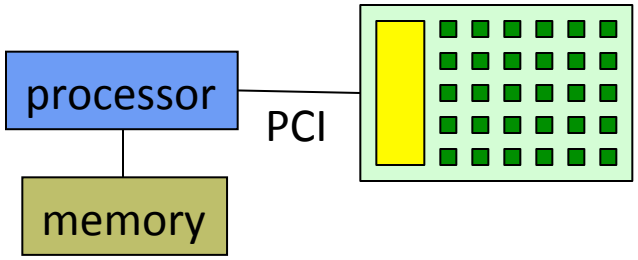
Parallel Architecture Types (3)

- Multicore
  - Multicore processor

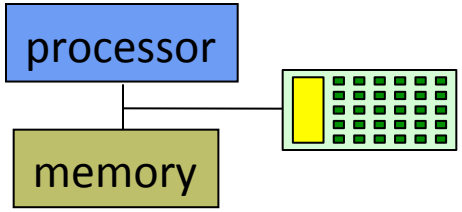


cores can be hardware multithreaded (hyperthread)

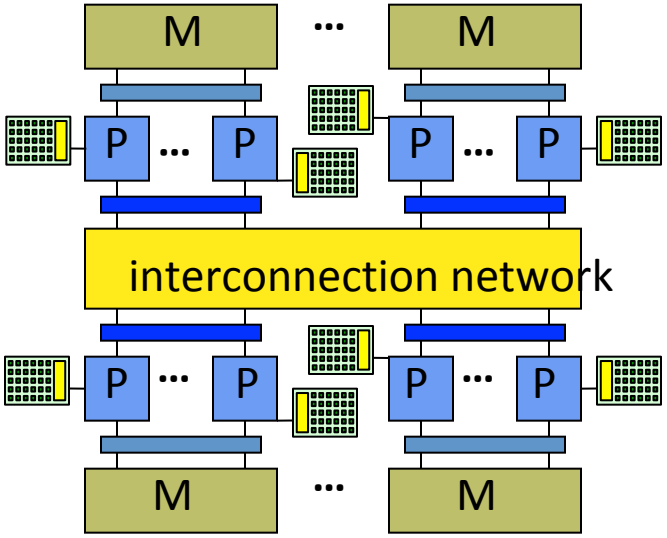
  - GPU accelerator



  - "Fused" processor accelerator



- Multicore SMP+GPU Cluster
  - Shared memory addressing within SMP node
  - Message passing between SMP nodes
  - GPU accelerators attached



Tipuri de Arhitecturi Paralele

- Instruction-Level Parallelism
  - Paralelism la nivel de executie a unei instructiuni
- Procesoare vectoriale
  - Operatii pe date multiple stocate in registrii de tip vector
- Shared-memory Multiprocessor (SMP)
  - Mai multe procesoare care partajeaza memorie
  - Symmetric Multiprocessor (SMP) (acelasi tip de procesoare)
- Multicomputer
  - Mai multe computere conectate via network
  - Cluster - Distributed-memory
- Massively Parallel Processor (MPP)

MIMD

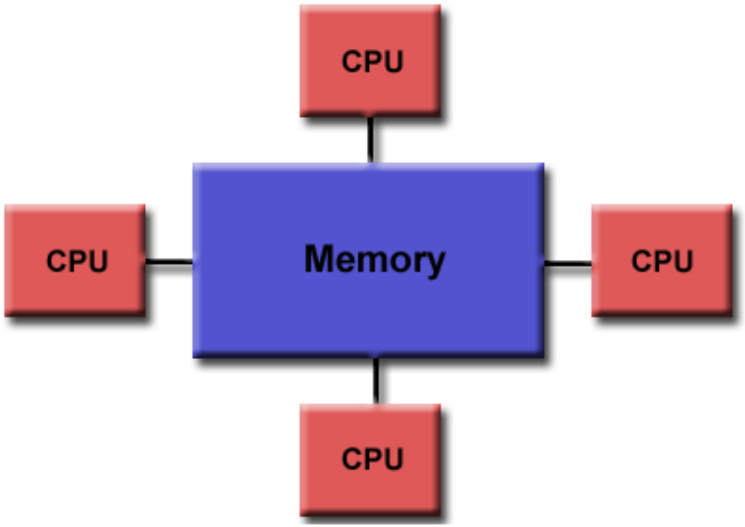
- - Clasificare in functie de tipul de memorie

Memorie partajata/ Shared Memory

- Toate procesoarele pot accesa intreaga memorie -> un singur spatiu de memorie (*global address space.)*
- 2 clase mari: ***UMA*** and ***NUMA***.

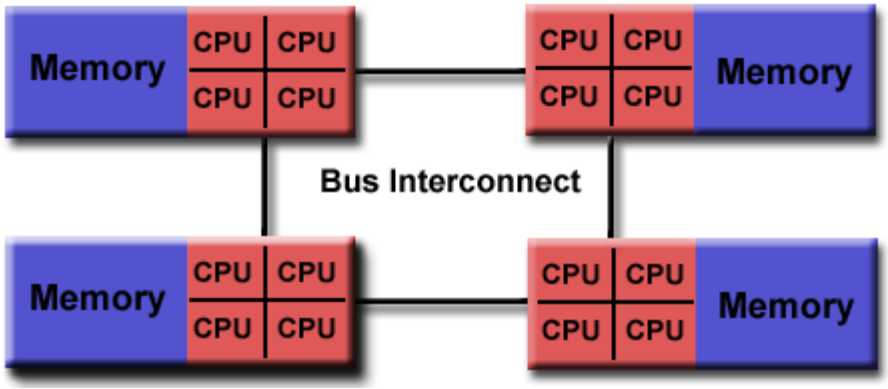
Shared Memory (UMA)

- **Uniform Memory Access (UMA):**
- Acelasi timp de acces la memorie
- Numit si CC-UMA - Cache Coherent UMA. (daca un procesor modifica o locatie de memorie toate celelalte “stiu” despre aceasta modificare. *Cache coherency* se obtine la nivel hardware.



Non-Uniform Memory Access (NUMA):

- Se obtine deseori prin unirea a 2 sau mai multe arh. SMP
- Nu e acelasi timp de acces la orice locatie de memorie
- Poate fi si varianta CC-NUMA - Cache Coherent NUMA



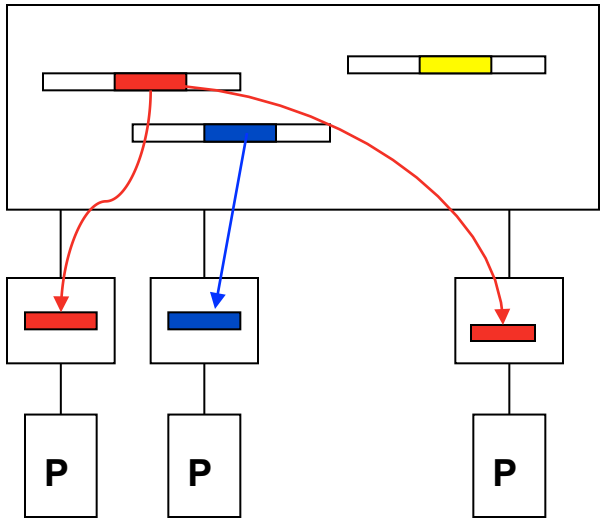
Shared Memory

Avantaje:
- *Global address space*
- *Partajare date rapida si uniforma*

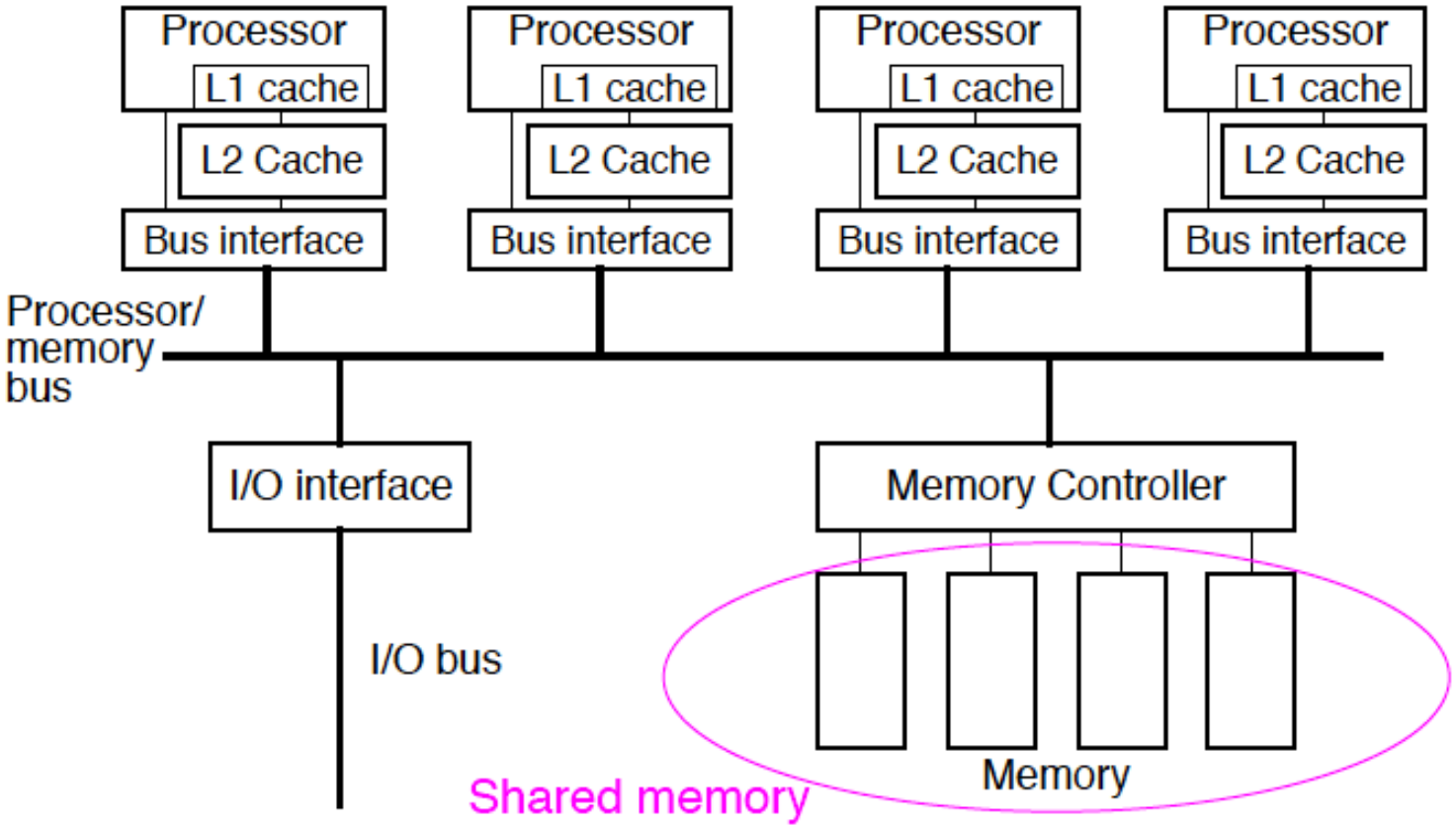
Dezavantaje:
- Lipsa scalabilitatii.
- Sincronizare in sarcina programatorului
- Costuri mari

Caching in sistemele cu memorie partajata

- Se reduce latenta medie
- Se reduce largimea de banda -bandwidth- medie
- Datele se transfera logic
  - store reg → mem
  - load reg ← mem
- Procesoarele pot partaja date eficient

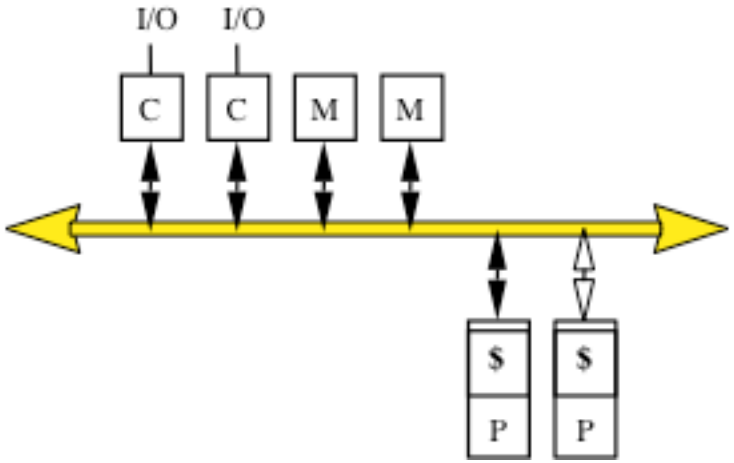


Niveluri de caching

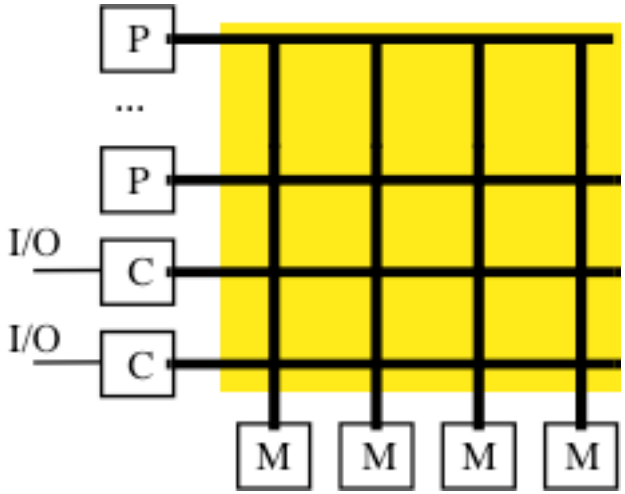


Bus-based SMP

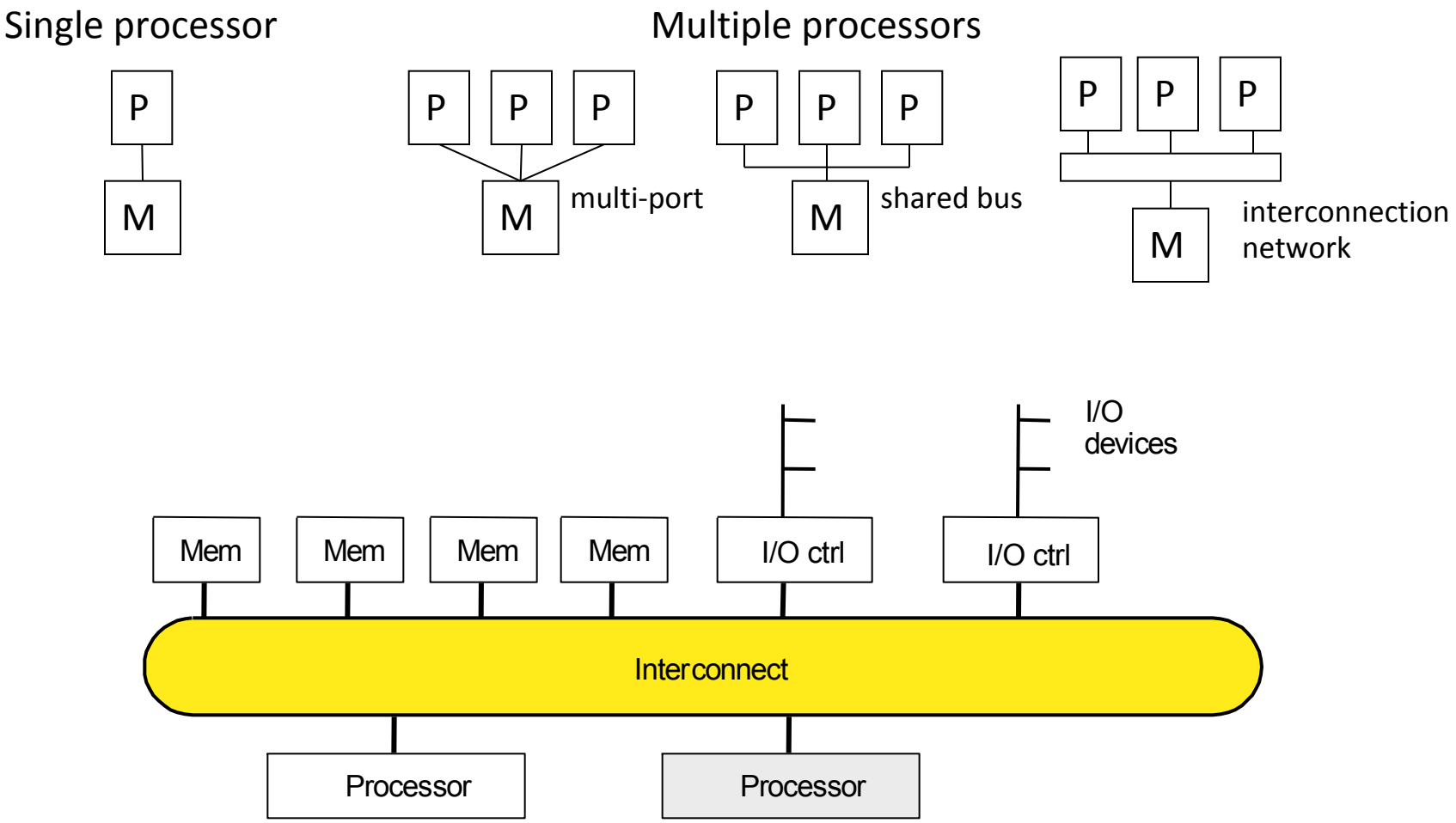
- *Uniform Memory Access* (*UMA*)
- Pot avea module multiple de memorie



Crossbar SMP



Shared Memory Multiprocessors (SMP) - overview



Parametrii de performanta corespunzatori accesului la memorie

- Latenta = timpul in care o data ajunge sa fie disponibila la procesor dupa ce s-a initiat cererea.

- Largimea de banda (*Bandwidth*) = rata de transfer a datelor din memorie catre procesor

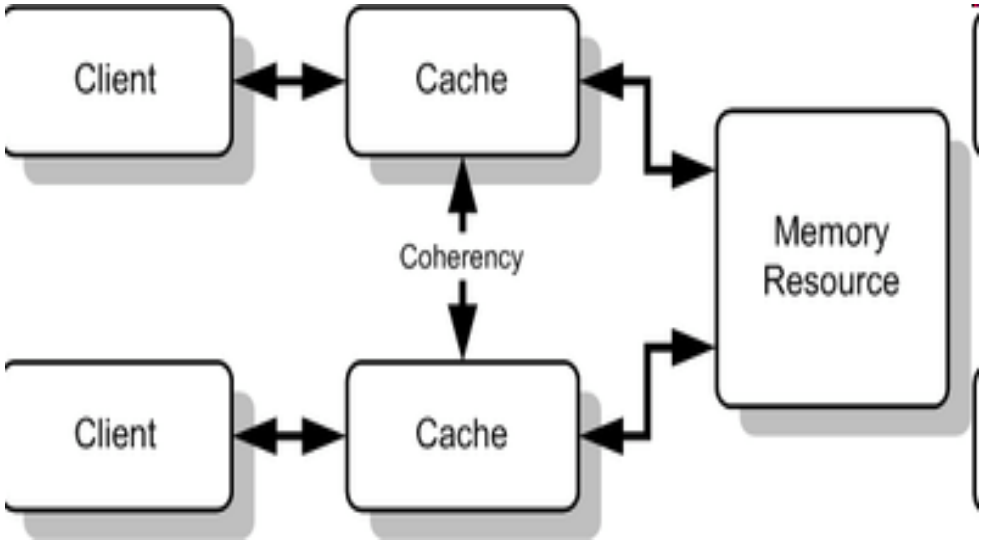
  - store reg → mem
  - load reg ← mem

Bandwidth vs. Throughput

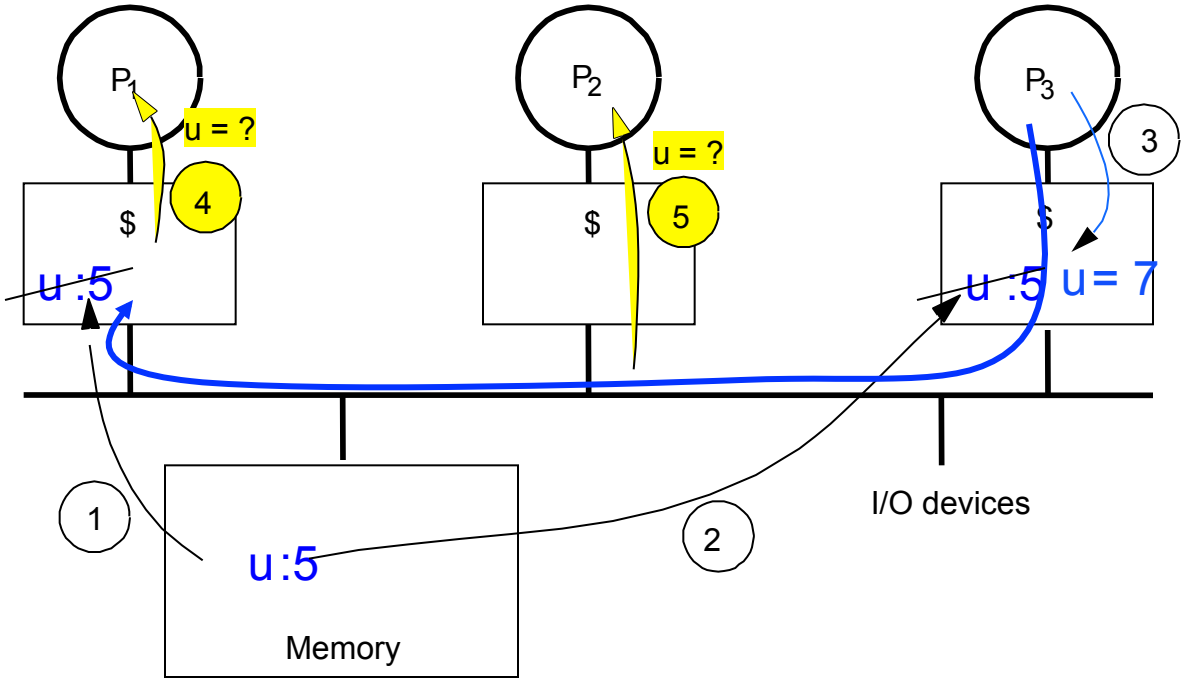
- Bandwidth - theoretical maximum units of work per unit of time
  Throughput - actual units of work per unit of time

- As opposed to the time per unit of work (speed/latency).

Cache coherence



Examplu: *Write-back Invalidate*



Cache Coherency <-> SMP

- Memoriile cache sunt foarte importante in SMP pentru asigurarea performantei
  - Reduce timpul mediu de acces la date
  - Reduce cerinta pentru largime de banda- *bandwidth*- plasate pe interconexiuni partajate
- Probleme coresp. *processor caches*
  - Copiile unei variabile pot fi prezente in cache-uri multiple;
  - o scriere de catre un procesor poate sa nu fie vizibila altor procesoare
    - acestea vor avea valori vechi in propriile cache-uri

  $\Rightarrow$ *Cache coherence* problem
- Solutii:
  - organizare ierarhica a memoriei;
  - Detectare si actiuni de actualizare.

Termeni

- Memory Operations - operatii cu memoria (load, store, read-modify-write, …)
- Perspectiva procesor
  - *Write*: citirile ulterioare returneaza valoarea noua
  - *Read*: scrierile ulterioare nu pot afecta valoarea
- *Coherent memory system*
  - Asigura o ordine seriala a operatiilor cu memoria a.i.
    - operatiile executate de un proces apar in ordinea executiei lor
    - valoarea returnata de fiecare citire este aceea returnata de scrierea anterioara

  $\Rightarrow$ *write propagation + write serialization*

Cerinte pentru un sistem *Cache Coherent System*

- Sa furnizeze un set de stari, o diagrama de tranzitie de stari si actiuni specifice
- Sa gestioneze protocolul de consistenta
  - (o) Determina cand sa invoce protocolul
  - (a) gaseste informatii despre starea altor cache-uri pentru a determina actiunea corecta
    - Cand trebuie sa comunice cu alte copii din cache-uri
  - (b) Localizarea altor copii
  - (c) Comunicare cu alte copii (invalidare/update)
- (0) se face la fel in toate sistemele
  - Starea se mentine in cache
  - Protocolul se invoca atunci cand un acces de tip “access fault” apare
- Diferite abordari in functie de f (a) (b) (c)

Motivatii pentru asigurarea consistentei memoriei

- Coerenta implica faptul ca scrierile la o locatie devin vizibile tuturor procesoarelor in aceeasi ordine .
- cum se stabileste ordinea dintre o citire si o scriere?
    - Sincronizare (event based)
  - Implementarea unui protocol hardware pentru *cache coherency.*
  - Protocolul se poate baza pe un model de consistenta a memoriei.

P_1	P_2
/* Assume initial value of A and flag is 0 */	
`A = 1;`	`while (flag == 0);` /* spin idly */
`flag = 1;`	`print A;`

Asigurarea consistentei memoriei

- Specificare de constrangeri legate de ordinea in care operatiile cu memoria pot sa se execute.
- Implicatii exista atat pentru programator cat si pentru proiectantul de sistem .
  - programatorul le foloseste pentru a asigura corectitudinea ;
  - proiectantul de sistem le poate folosi pentru a contrange gradul de reordonare de instructiuni al compilatorului sau al hardware-ului.
- Contract intre programator si sistem.

(Consistenta secventiala) Sequential Consistency

- Ordine totala prin intreteserea accesurilor de la diferite procesoare
  - *program order*
  - Operatiile cu memoria ale tuturor procesoarelor par sa inceapa, sa se execute si sa se termine 'atomic' ca si cum ar fi doar o singura memorie (no cache).

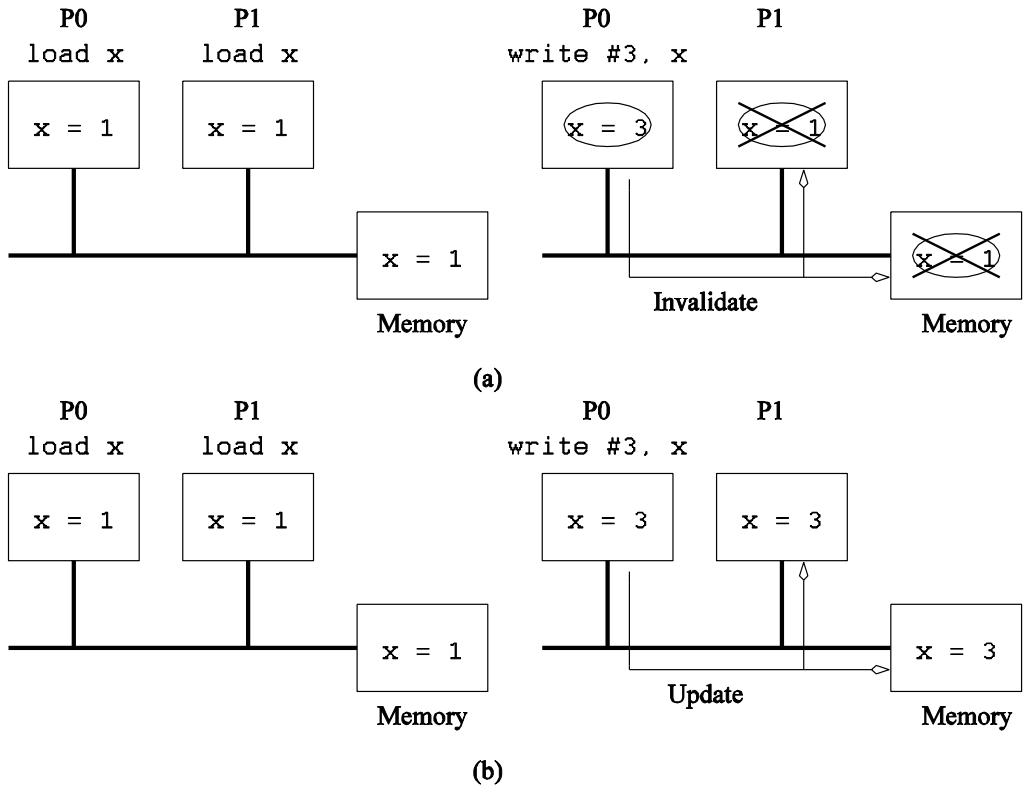
*"A multiprocessor is **sequentially consistent** if the result of any execution is the same as if the operations of all the processors were executed in some sequential order, and the operations of each individual processor appear in this sequence in the order specified by its program."*
[Lamport, 1979]

Consistenta secventiala -Conditii

- Exista o ordine totala specifica programului=>
- Conditii suficiente:
  - Fiecare proces initiaza operatiile cu memoria in ordinea specificata a programului
  - Dupa initierea unei operatii *write*, procesul care a initiat-o asteapta terminarea inainte sa initiaza alta operatie cu memoria (*atomic writes*)
  - Dupa o initiere unei operatii *read* procesul initiator asteapta terminarea operatiilor write corespunzatoare (global) si a operatiei read inainte sa initieze alta operatie.

Exemplu: *Cache Coherence*

Atunci cand o variabila este modificata, toate copiile ei trebuie sa fie actualizate sau invalidate.



(a) Invalidate protocol; (b) Update protocol for shared variables.

Update <->Invalidate Protocols

- Daca un procesor doar citeste o valoare odata si nu o mai foloseste, un protocol de tip *update* poate genera un overhead semnificativ.
- Daca 2 procesoare fac teste si update la o variabila este indicat protocolul “update”.
- Ambele sufera de problema: “false sharing overheads” (2 cuvinte de memorie care nu sunt partajate, dar sunt in aceeasi linie de cache sunt actualizate).
- Majoritatea masinilor folosesc “invalidate protocols”.

Invalidate Protocol

- Fiecarei copii a unei date i se asociaza o stare.
- Set de stari posibile: *shared, invalid, dirty.*
- *Shared* -> mai multe copii valide; dupa un write la o copie celelalte trec in stare invalid.
- *Dirty* -> doar o singura copie exista deci nu e nevoie de invalidari ulterioare.
- *Invalid*-> o operatie read va face o cerere de data din memorie.

Invalidate Protocol

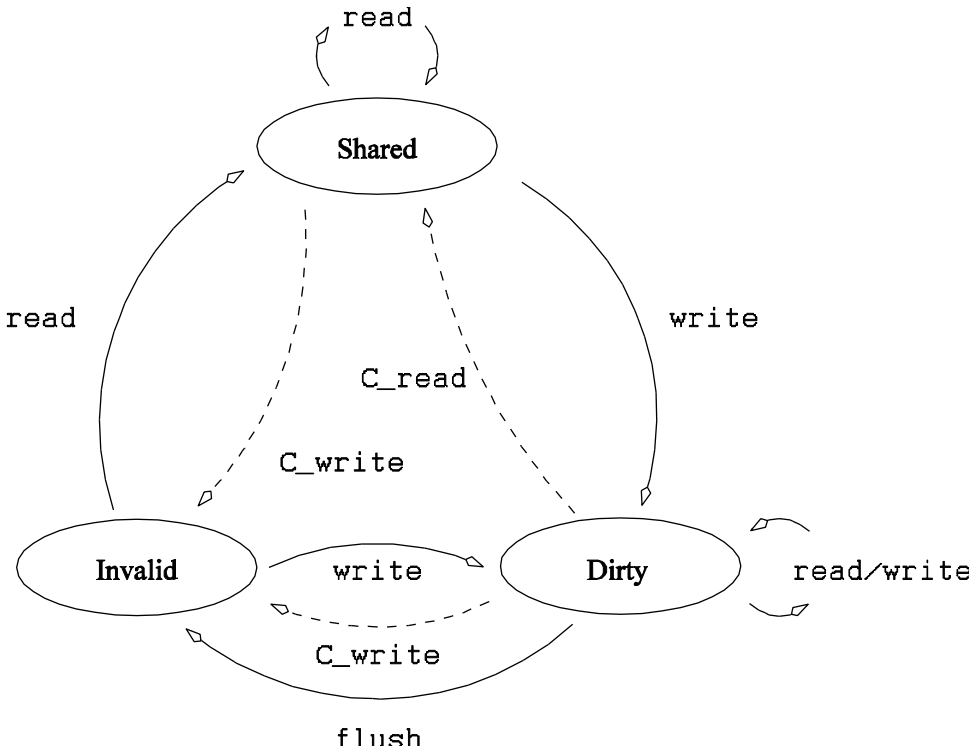


Diagrama de stare.

Invalidate Protocol

Time ↓

Instruction at Processor 0	Instruction at Processor 1	Variables and their states at Processor 0	Variables and their states at Processor 1	Variables and their states in Global mem.
				x = 5, D y = 12, D
read x		x = 5, S		x = 5, S
	read y		y = 12, S	y = 12, S
x = x + 1		x = 6, D		x = 5, I
	y = y + 1		y = 13, D	y = 12, I
read y		y = 13, S	y = 13, S	y = 13, S
	read x	x = 6, S	x = 6, S	x = 6, S
x = x + y		x = 19, D	x = 6, I	x = 6, I
	y = x + y	y = 13, I	y = 19, D	y = 13, I
x = x + 1		x = 20, D		x = 6, I
	y = y + 1		y = 20, D	y = 13, I

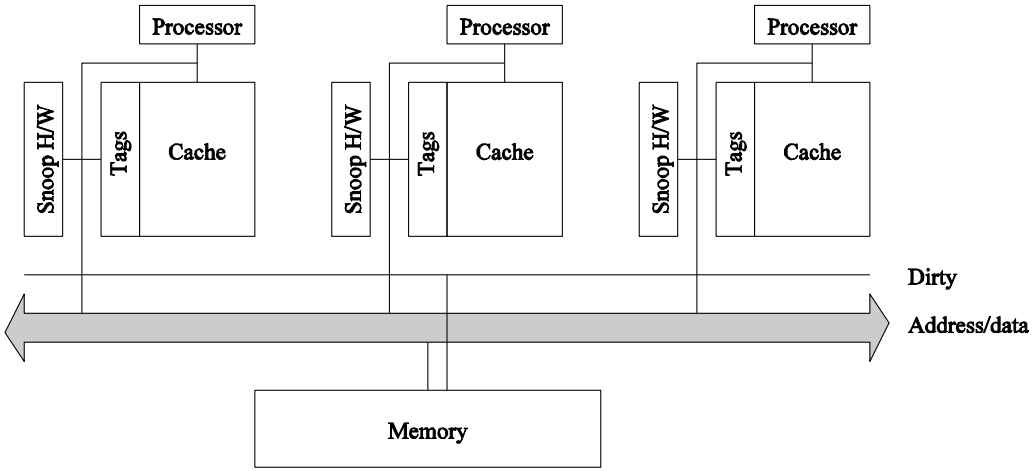
Exemplu de executie a unui program paralel.

Arhitectura: *Bus-based Cache-Coherent (CC)*

- Tranzactii pe magistrala unica (Bus Transactions)
  - Protocol (Bus protocol): arbitrar, comanda/addr, data
  - Fiecare componenta observa fiecare tranzactie
- Diagrama de tranzitie pentru starea blocului asociat cache-ului
    - *invalid, valid, dirty*
  - *Snoopy protocol*
- Alegeri de baza
  - *Write-through vs Write-back*
  - *Invalidate vs. Update*

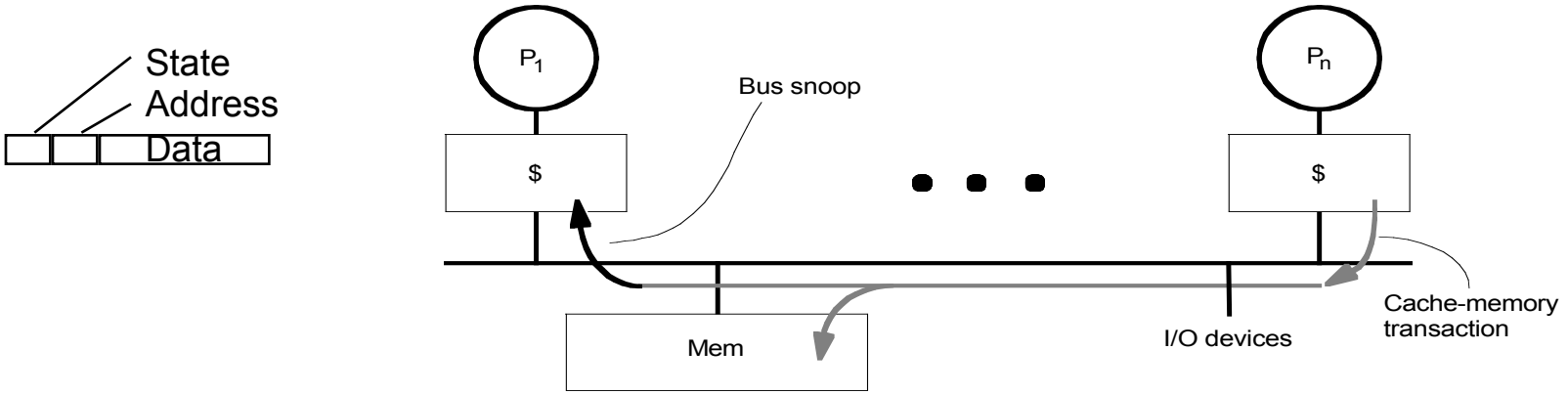
Snoopy Cache Systems

- Pe retea se transmit cererile pt asigurarea corentei-> bandwidth => scadere performanta



A simple snoopy bus based cache coherence system.

Protocoale *Snoopy Cache-Coherency*



- Bus/Magistrala este mediul de transmisie global
- Memoriile cache “stiu” ceea ce contin
- Controllerul de cache verifica “snoops” toate tranzactiile de pe magistrala partajata
  - Tranzactiile relevante sunt cele pentru blocul continut de cache
  - Actiuni care asigura coerenta
    - invalidare, update, ori furnizare valoare
  - depinde de starea blocului si de protocol

Directory Based Systems

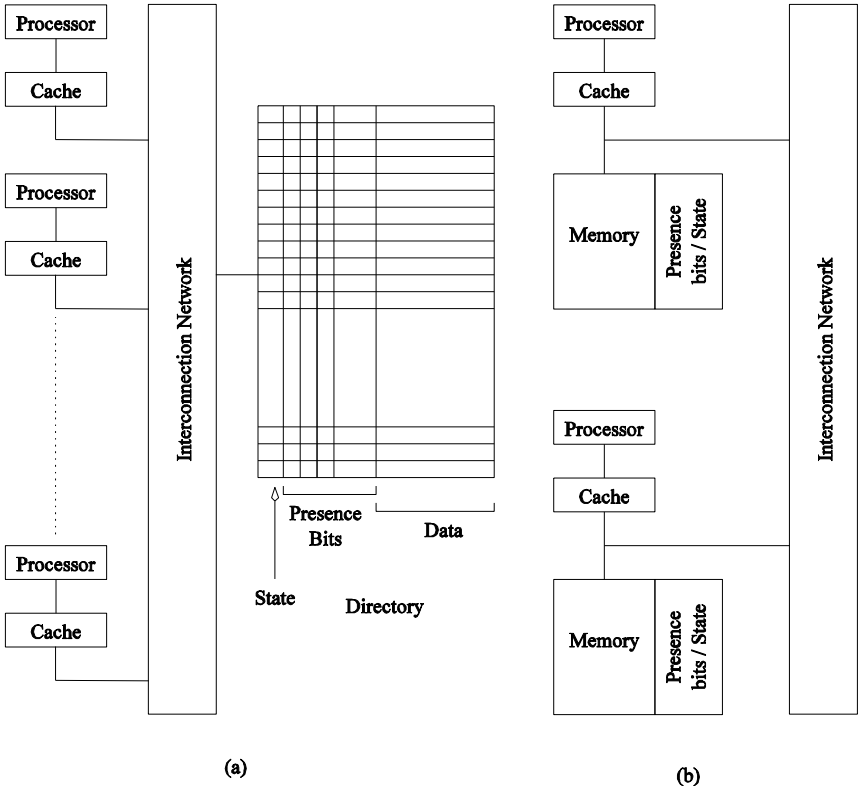
- In *snoopy caches* – operatiile de asigurare a coerentei se trimit catre toate procesoarele.

- Pentru eficienta se pot trimite doar celor care trebuie sa fie notificate – se foloseste un director care pastreaza un vector de prezenta pentru fiecare data (cache line) impreuna cu starea globala.

- Necesitatea de a folosi un mediu de transmitere broadcast este inlocuita de director.

- Spatiu suplimentar necesar pentru director.
- Directorul este un punct de concurenta (*contention)* => distributed directory schemes

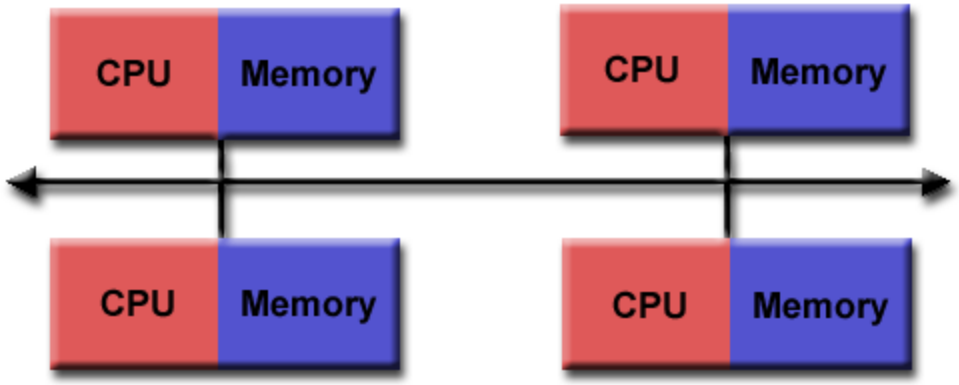
Directory Based Systems



(a) a centralized directory; (b) a distributed directory.

Arhitecturi cu Memorie Distribuita/*Distributed Memory*

- Retea de interconectivitate /***communication network***
- Procesoare cu memorie locala ***local memory***. Memory



Arhitecturi cu memorie distribuita

Avantaje:

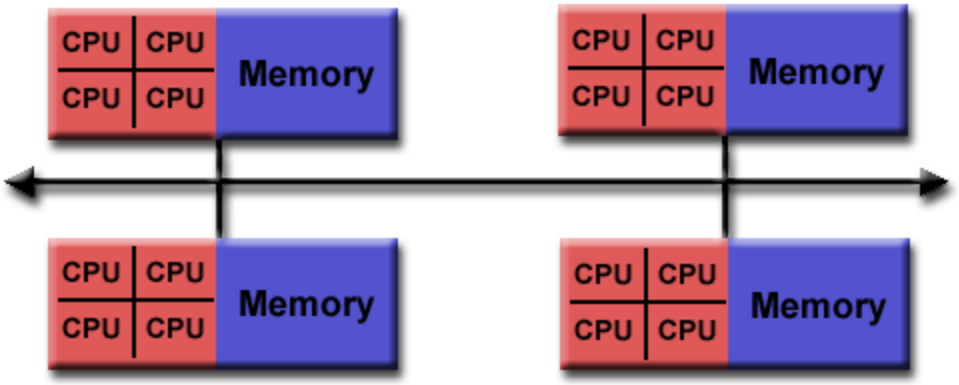
- Memorie scalabila – odata cu cresterea nr de procesoare
- Cost redus – retele

Dezavantaje:

- Responsibilitatea programatorului sa rezolve comunicatiile.
- Dificil de a mapa structuri de date mari pe mem. distribuita.
- Acces Ne-uniform la memorie

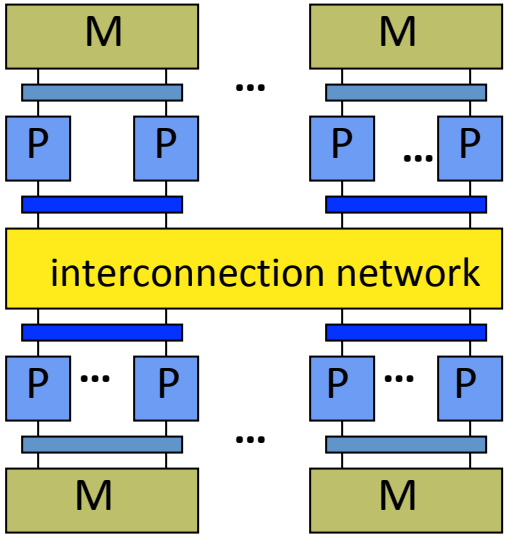
Hybrid Distributed-Shared Memory

- Retea de SMP



SMP Cluster

- Clustering
  - Noduri integrate
- Motivare
  - Partajare resurse
  - Se reduc costurile de retea
  - Se reduc cerintele pt largimea de banda (*bandwidth*)
  - Se reduce latenta globala
  - Creste performanta per node
  - Scalabil



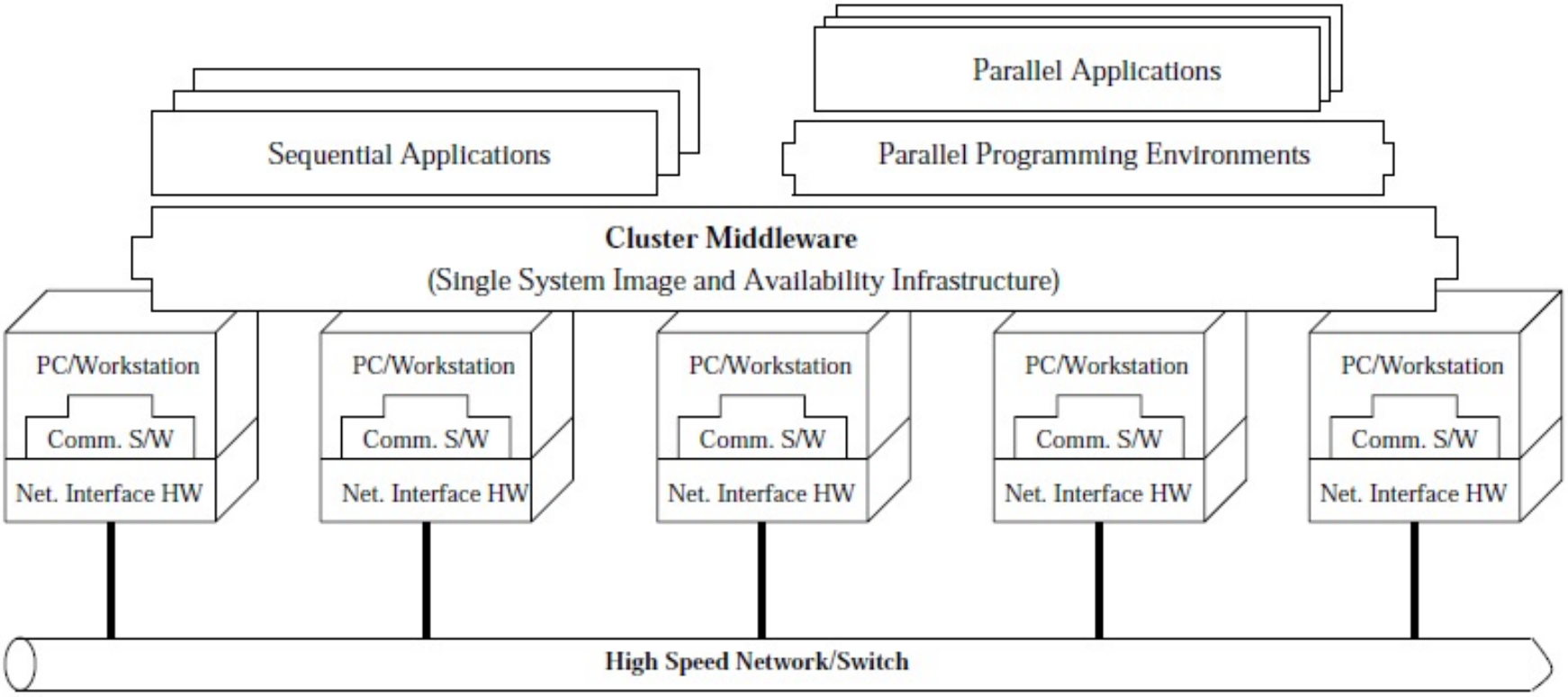
MPP(Massively Parallel Processor)

- Fiecare nod este un sistem independent care are local:
  - Memorie fizica
  - Spatiu de adresare
  - Disc local si conexiuni la retea
  - Sistem de operare

- *MPP (massively parallel processing) is the coordinated processing of a program by multiple processors that work on different parts of the program, with each processor using its own operating system and memory. Typically, MPP processors communicate using some messaging interface. In some implementations, up to 200 or more processors can work on the same application. An "interconnect" arrangement of data paths allows messages to be sent between processors. Typically, the setup for MPP is more complicated, requiring thought about how to partition a common database among and how to assign work among the processors. An MPP system is also known as a "loosely coupled" or "shared nothing" system.*
- *An MPP system is considered better than a symmetrically parallel system (SMP) for applications that allow a number of databases to be searched in parallel. These include decision support system and data warehouse applications.*

COW

- Cluster of Workstations



Performanta

- Problema: daca un procesor este evaluat la nivel k MFLOPS si sunt p procesoare, performanta totala este de ordin k*p MFLOPS?
- Mai concret: daca un calcul necesita 100 sec. pe un procesor se va putea face in 10 sec. pe 10 procesoare?
- Cauze care pot afecta performanta
  - Fiecare proc. –unitate independenta
  - Interactiunea lor poate fi complexa

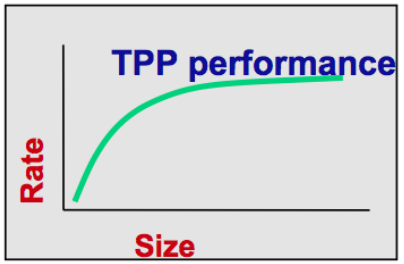
- *Need to understand performance space*

Scalabilitate

- Cat de mult se poate mari sistemul? (unitati de procesare, unitati de memorie)
- Cate procesoare se pot adauga fara a se diminua caracteristicile generale ale acestuia (viteza de comunicare, viteza de accesare memorie, etc.)
- Masuri de eficienta (performance metrics)

Top 500 Benchmarking
https://www.top500.org/project/linpack/

- Cele mai puternice 500 calculatoare din lume
- High-performance computing (HPC)
  - Rmax : *maximal performance Linpack benchmark*
    - Sistem dens liniar de ecuatii (Ax = b)
- Informatii date
  - Rpeak : *theoretical peak performance*
  - Nmax : dimensiunea problemei necesara pt a se atinge Rmax
  - N1/2 : dimensiunea problemei necesara pt a se atinge 1/2 of Rmax
  - Producator si tipul calculatorului
  - Detalii legate de instalare (location, an,…)
- Actualizare de 2 ori pe an



Rank	Site	System	Cores	Rmax (TFlop/s)	Rpeak (TFlop/s)	Power (kW)
1	National Supercomputing Center in Wuxi					
China	**Sunway TaihuLight** - Sunway MPP, Sunway SW26010 260C 1.45GHz, Sunway					
NRCPC	10,649,600	93,014.6	125,435.9	15,371		
2	National Super Computer Center in Guangzhou					
China	**Tianhe-2 (MilkyWay-2)** - TH-IVB-FEP Cluster, Intel Xeon E5-2692 12C 2.200GHz, TH Express-2, Intel Xeon Phi 31S1P					
NUDT	3,120,000	33,862.7	54,902.4	17,808		
3	DOE/SC/Oak Ridge National Laboratory					
United States	**Titan** - Cray XK7 , Opteron 6274 16C 2.200GHz, Cray Gemini interconnect, NVIDIA K20x					
Cray Inc.	560,640	17,590.0	27,112.5	8,209		

Tianhe-2 (MilkyWay-2) - TH-IVB-FEP Cluster,

- Compute Nodes have 3.432 Tflop/s per node
  - 16,000 nodes
  - 32000 Intel Xeon CPU
  - 48000 Intel Xeon Phi
- Operations Nodes
  - 4096 FT CPUs
- Proprietary interconnect
  - TH2 express
- 1PB memory
  - Host memory only
- Global shared parallel storage: 12.4 PB
- Cabinets: 125+13+24 =162