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,
  - dacă multimea de procesoare este coordonata 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

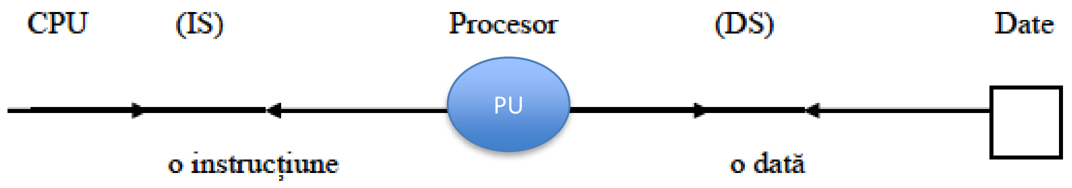
Michael J. Flynn în 1966

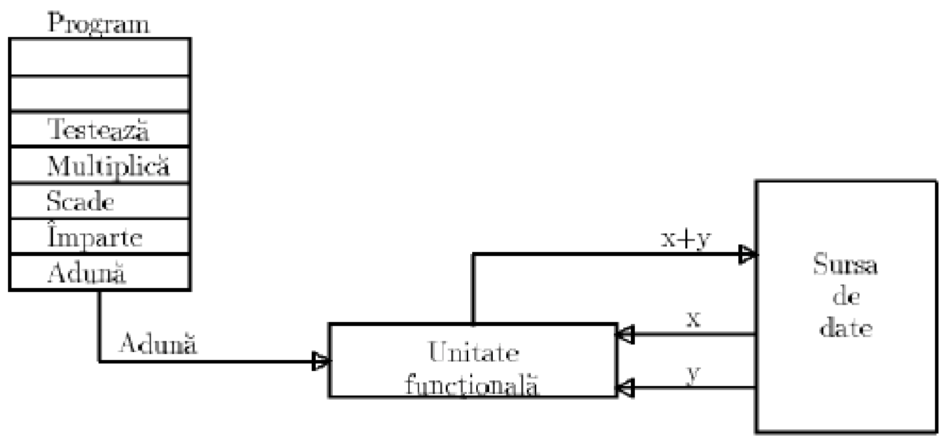
- SISD: sistem cu un singur flux de instrucţiuni şi un singur flux de date;
- SIMD: sistem cu un singur flux de instrucţiuni şi mai multe fluxuri de date;
- MISD: sistem cu mai multe fluxuri de instrucţiuni şi un singur flux de date;
- MIMD: cu mai multe fluxuri de instrucţiuni şi mai multe fluxuri de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

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

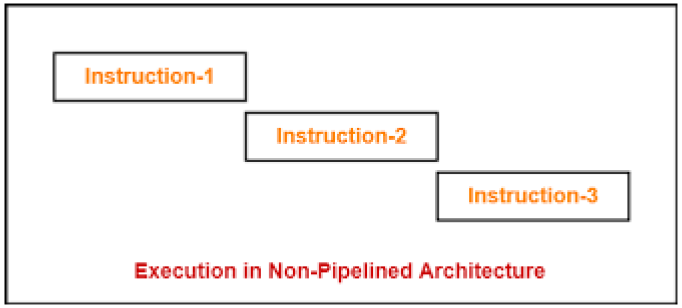
SISD(Single instruction stream, single data stream)

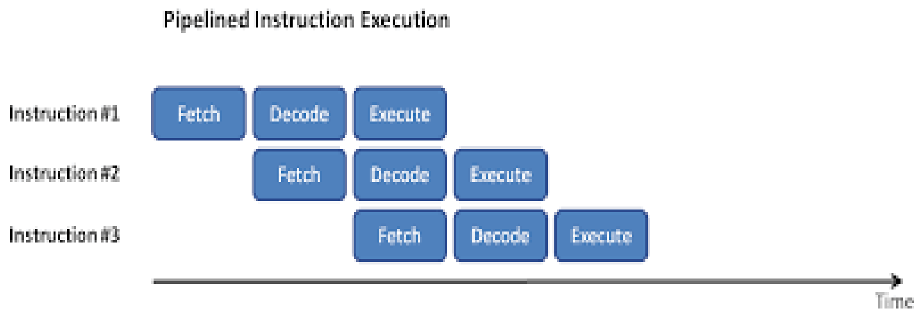
Flux de instrucțiuni singular, flux de date singular (SISD)-
- microprocesoarele clasice cu arhitecturi von Neumann
- Functionare ciclica: preluare instr., stocare rez. in mem. , etc.

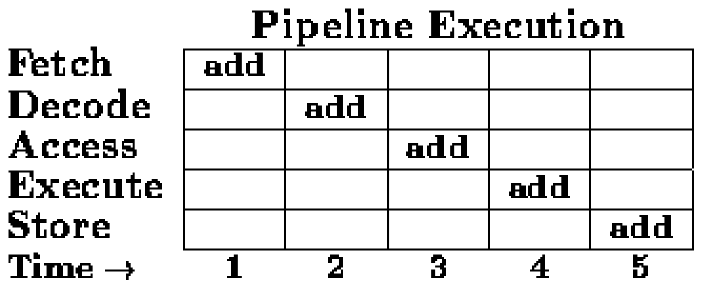




Non-pipelined vs. pipeline procesors



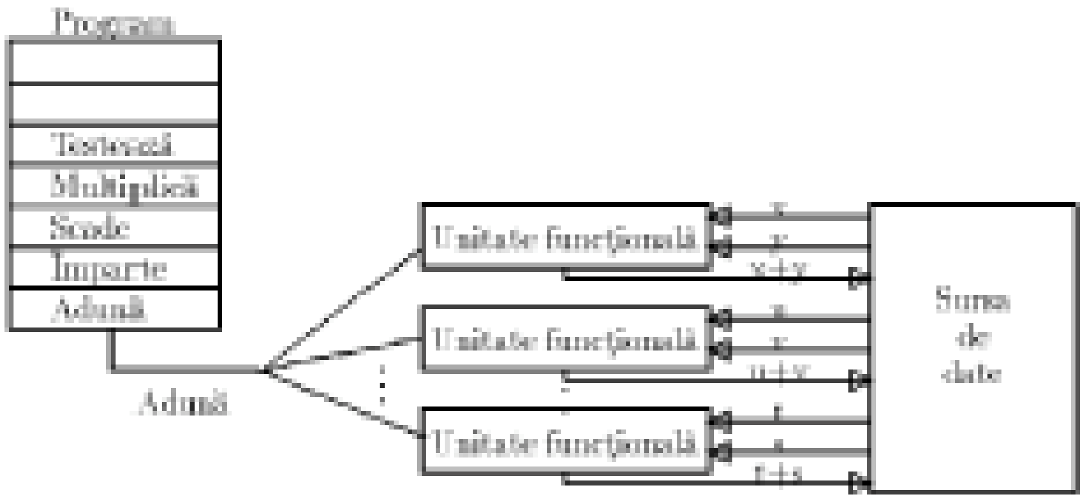


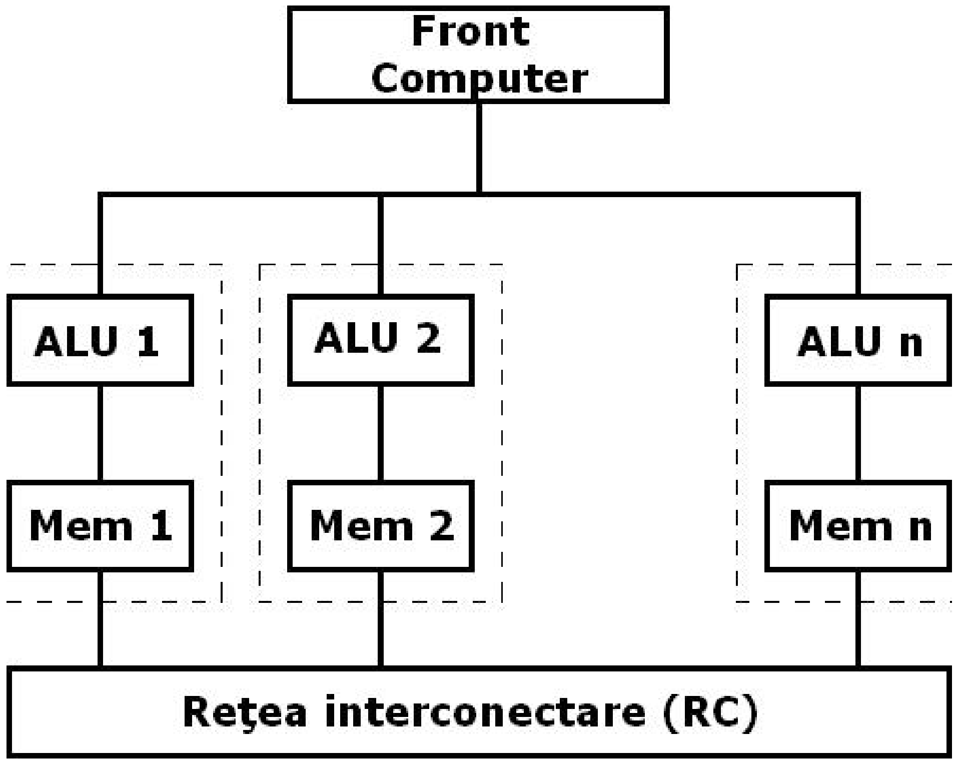

Pipeline Execution

Fetch	add				
Decode		add			
Access			add		
Execute				add	
Store					add
Time →	1	2	3	4	5

SIMD (Single instruction stream, multiple data stream)

Flux de instrucțiuni singular, flux de date multiplu

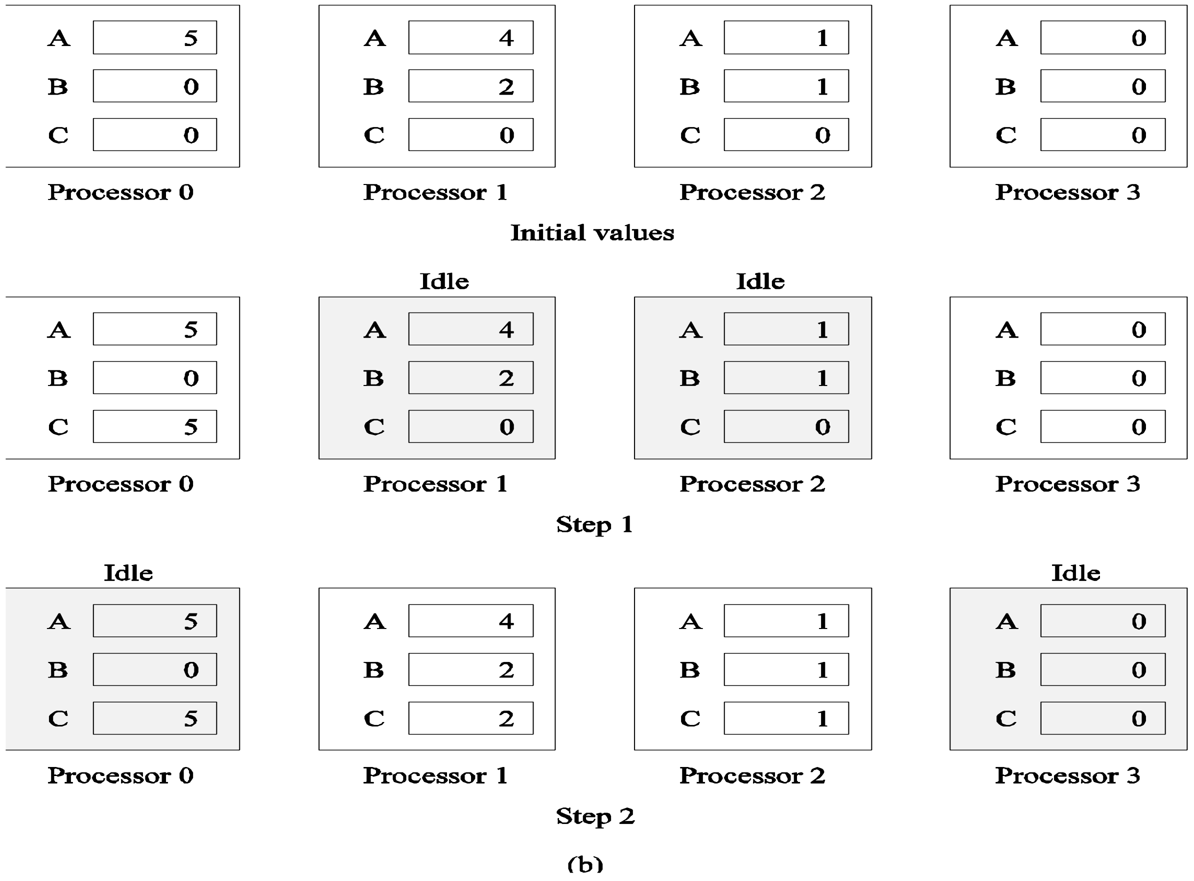



Front Computer
ALU 1
ALU 2
ALU n
Mem 1
Mem 2
Mem n
Reţea interconectare (RC)

Executie conditionala in SIMD Processors

```
if (B == 0)
        C = A;
else
        C = A/B;
```

(a)

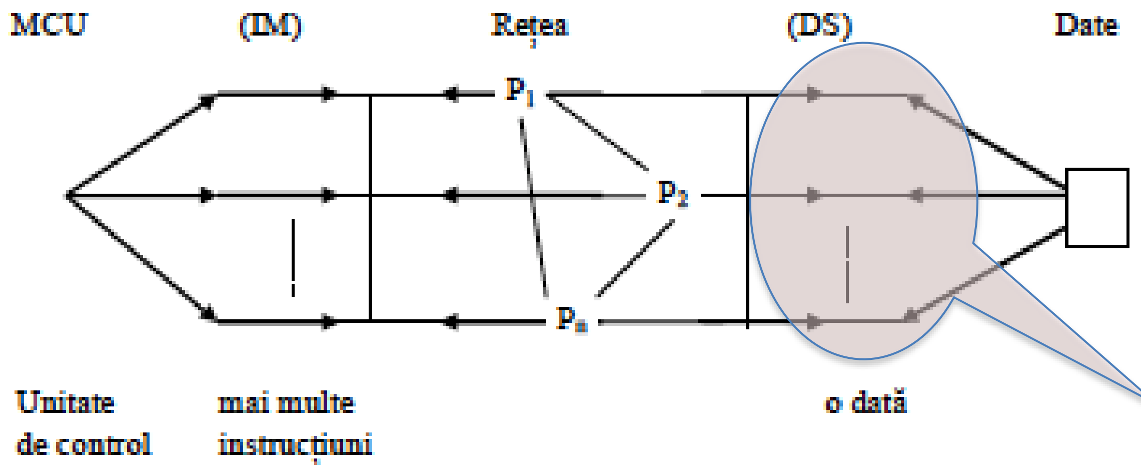


(b)

MISD (multiple instruction stream, single data stream)

Flux de instrucțiuni multiplu, flux de date singular

- multime vida !!!



nu se poate considera ca este un singur stream de date => sunt mai multe care eventual contin aceeasi valoare (copie) => nu se incadreaza

~~~ procesoare pipeline:

intr-un **procesor pipeline** exista un singur flux (stream) de date dar aceasta trece prin transformari succesive (mai multe instructiuni) iar paralelismul este realizat prin execuţia simultana a diferitelor etape de calcul asupra unor date diferite (secventa de date care intra succesiv pe streamul de date)
~~~

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

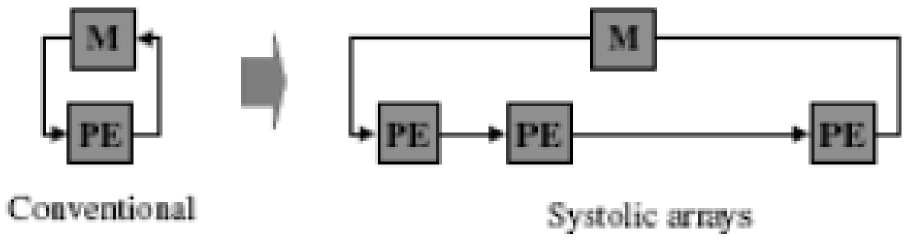
Initial motivation

VLSI enables inexpensive special-purpose chips

Represent algorithms directly by chips connected in regular pattern

Very-large-scale integration

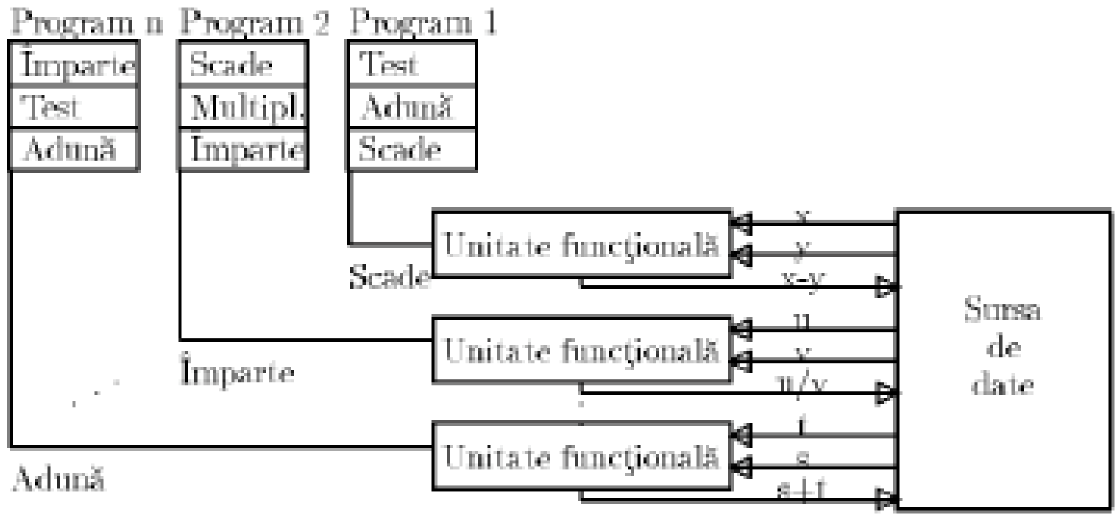
Systolic Architectures



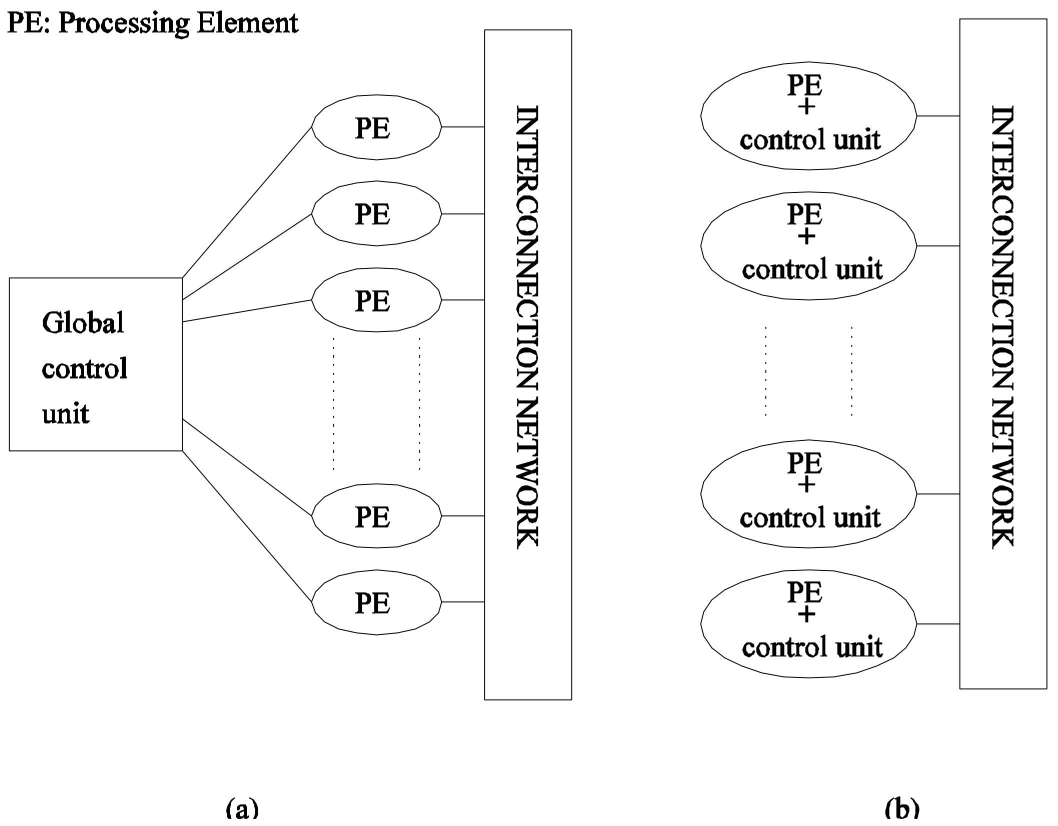
Replace a processing element(PE) with an array of PE's without increasing I/O bandwidth

MIMD (multiple instruction stream, multiple data stream)

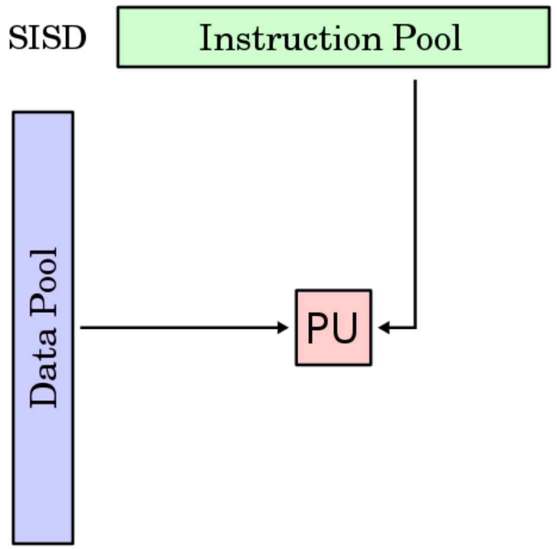
Flux de instrucțiuni multiplu, flux de date multiplu

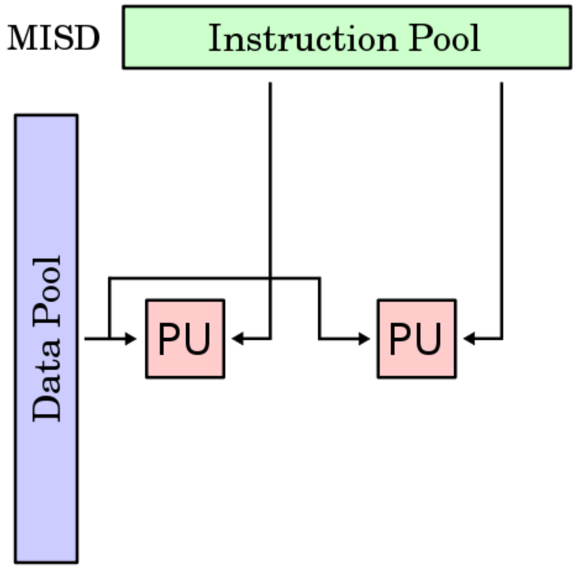


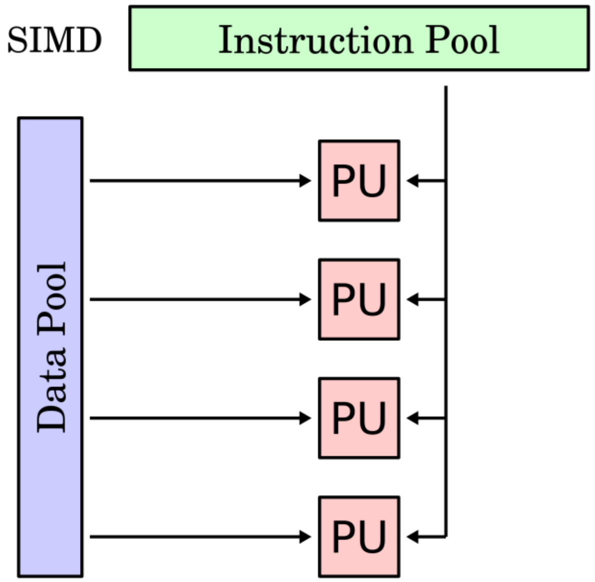
SIMD versus MIMD

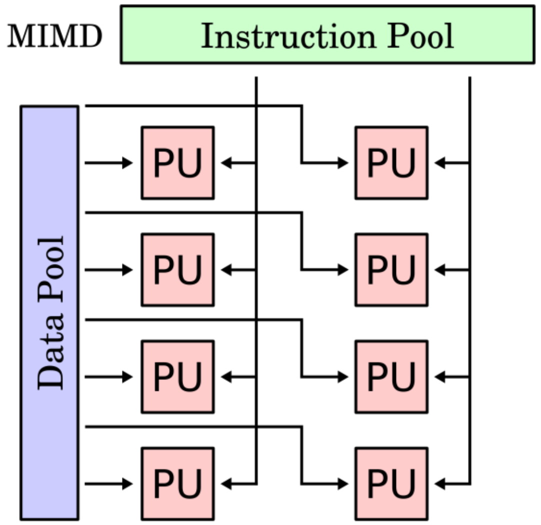


Sumar -scheme Comparative – clasificare Flynn





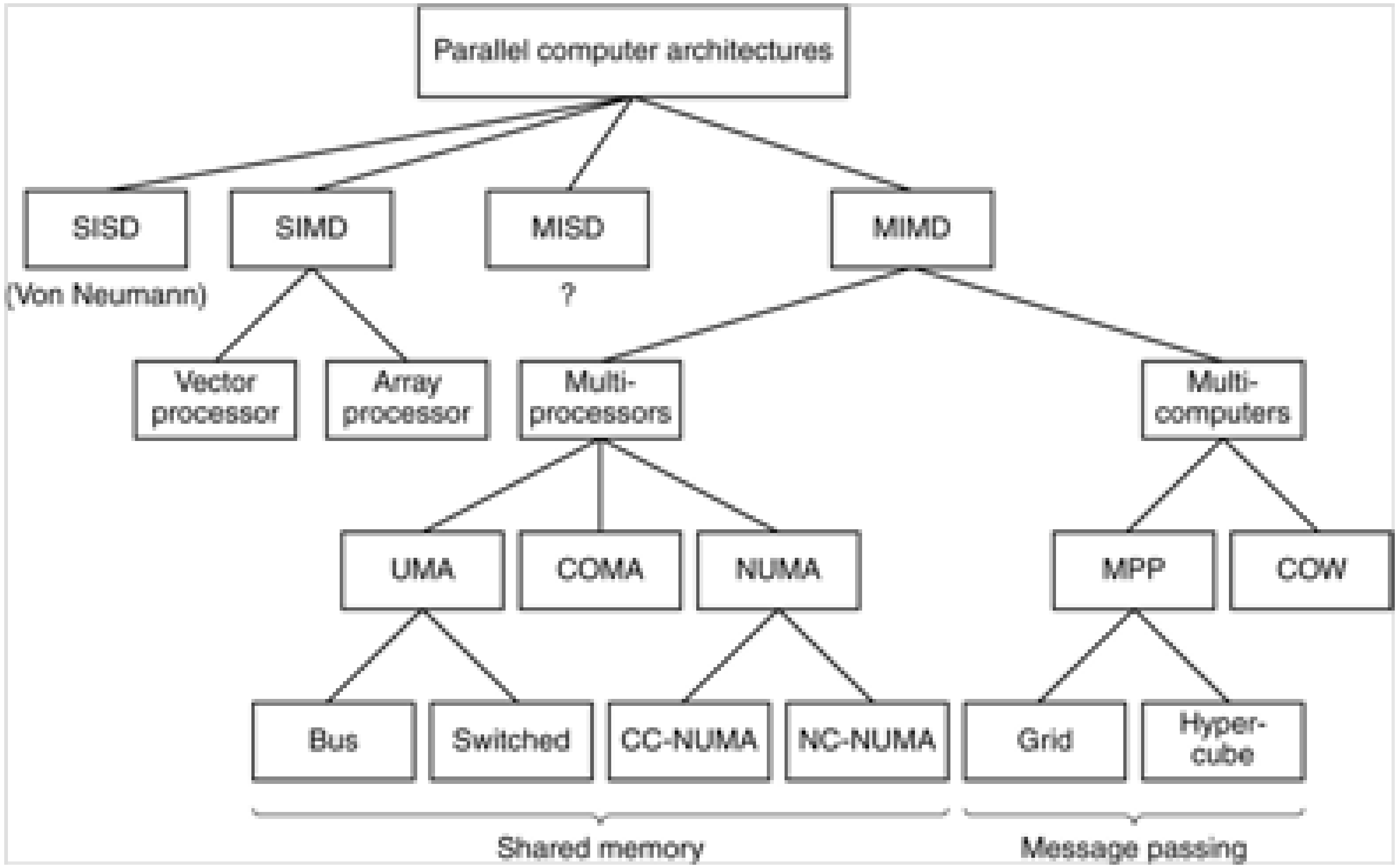




Paralelizare la nivel hardware – istoric

- Etapa 1 (1950s): executie secventiala 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, multi-cores, acceleratori, heterogenous clusters

Vedere generala



MIMD

- Clasificare in functie de tipul de memorie
  - partajata
  - distribuita
  - hibrida

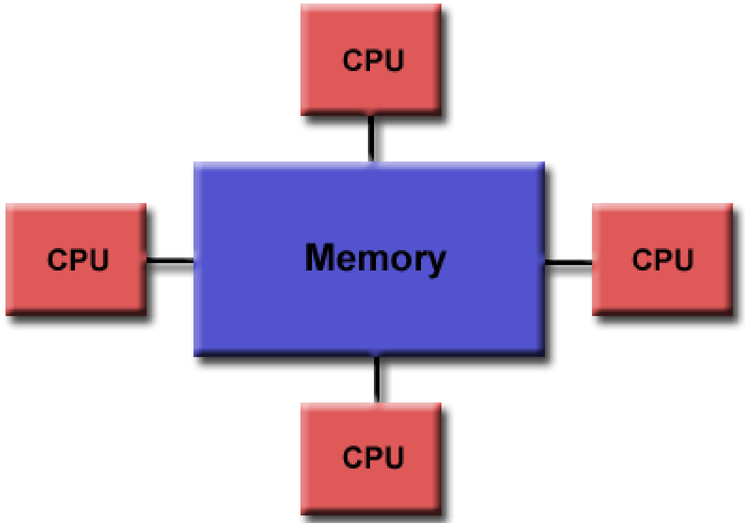
Memorie partajata/ Shared Memory

- Toate procesoarele pot accesa intreaga memorie -> un singur spatiu de memorie (*global address space.)*

- Shared memory=> 2 clase mari: ***UMA*** and ***NUMA***.

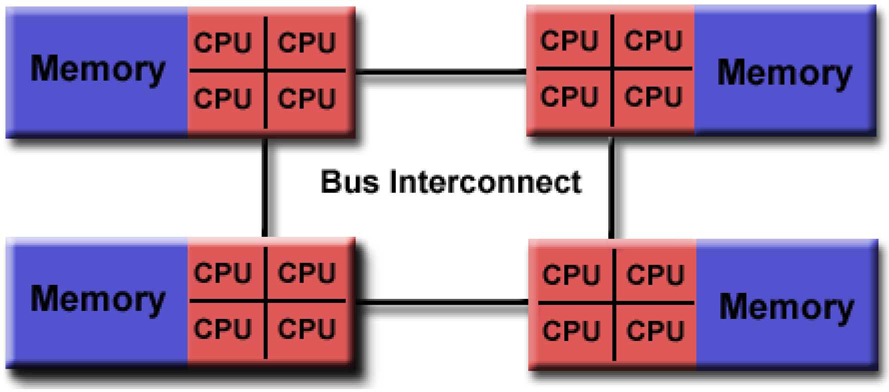
Shared Memory (UMA)

- **Uniform Memory Access (UMA):**
- Acelasi timp de acces la memorie
- **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.



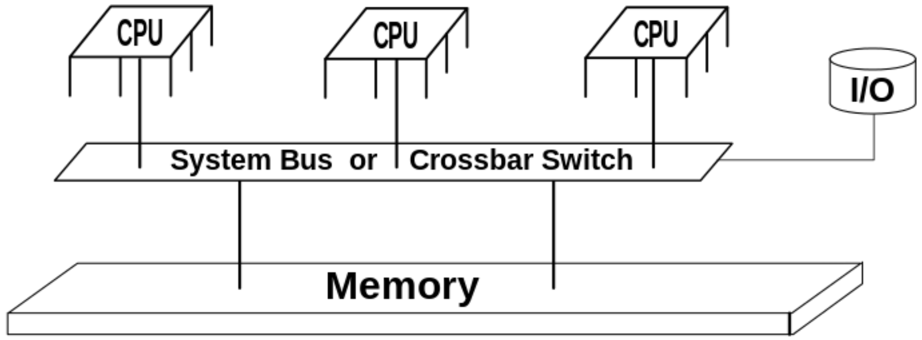
Non-Uniform Memory Access (NUMA):

- Se obtine deseori prin unirea a 2 sau mai multe arhitecturi UMA
- Nu e acelasi timp de acces la orice locatie de memorie
- ***Poate*** fi si varianta CC-NUMA - Cache Coherent NUMA
  - ex. HP's Superdome, SUN15K, IBMp690

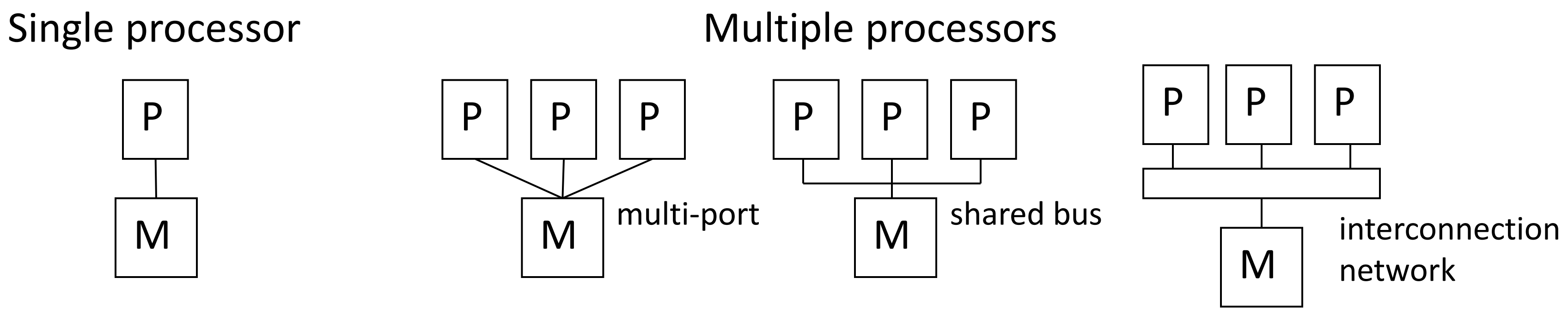


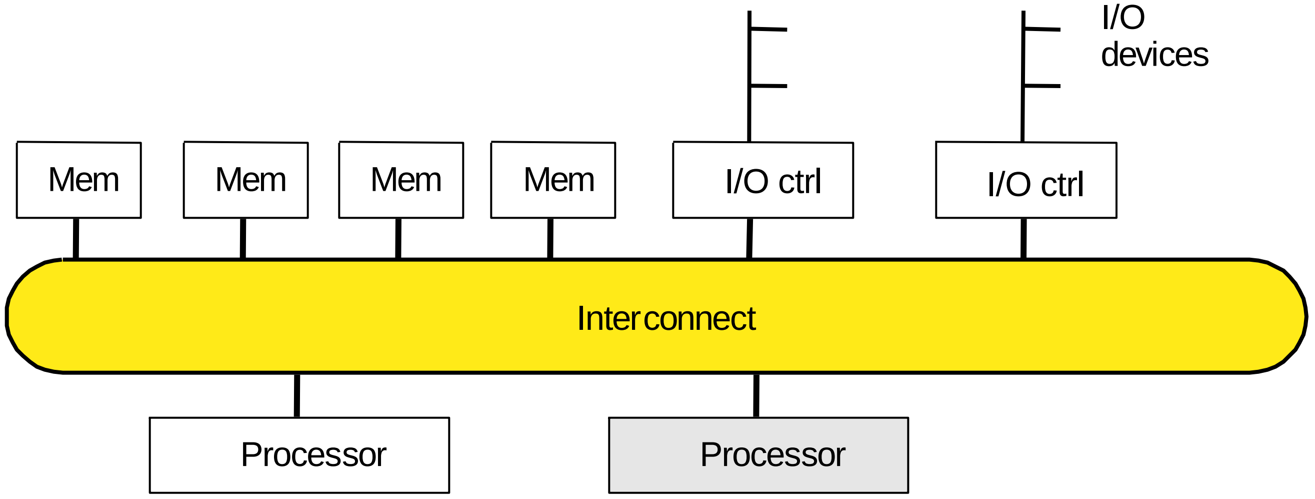
SMP Symmetric multiprocessor computer

- acces similar la toate procesoarele dar si la I/O devices, USB ports ,hard disks,...
- o singura memorie comuna
- un sistem de operare
- controlul procesoarelor – egal (similar)
- distributia threadurilor – echilibrata+echidistanta
- exemplu simplu: 2 procesoare Intel Xeon-E5 processors ->aceeasi motherboard
- Ex. - servere



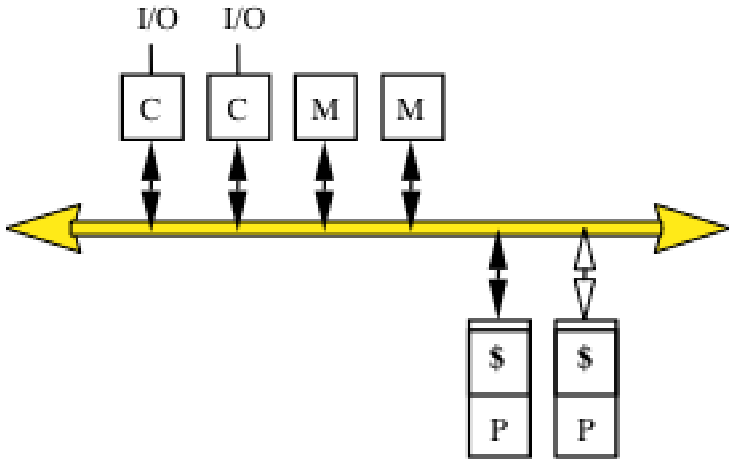
Shared Memory Multiprocessors (SMP) - overview



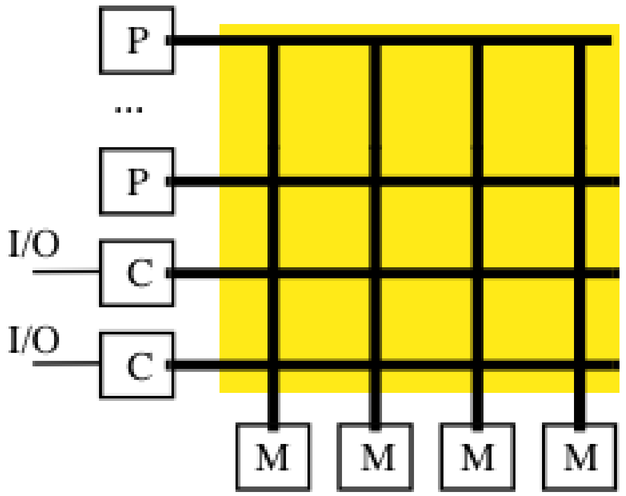


Bus-based SMP(Symmetric Multi-Processor)

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



Crossbar SMP



Parametrii de performanta corespunzatori accesului la memorie

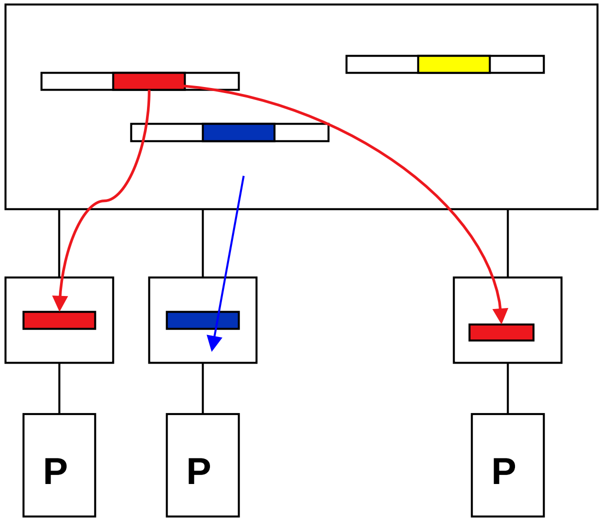
- Latenta = timpul in care o data ajunge sa fie disponibila la procesor dupa ce s-a initiat cererea.

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

  – store reg $\rightarrow$ mem
  – load reg $\leftarrow$ mem

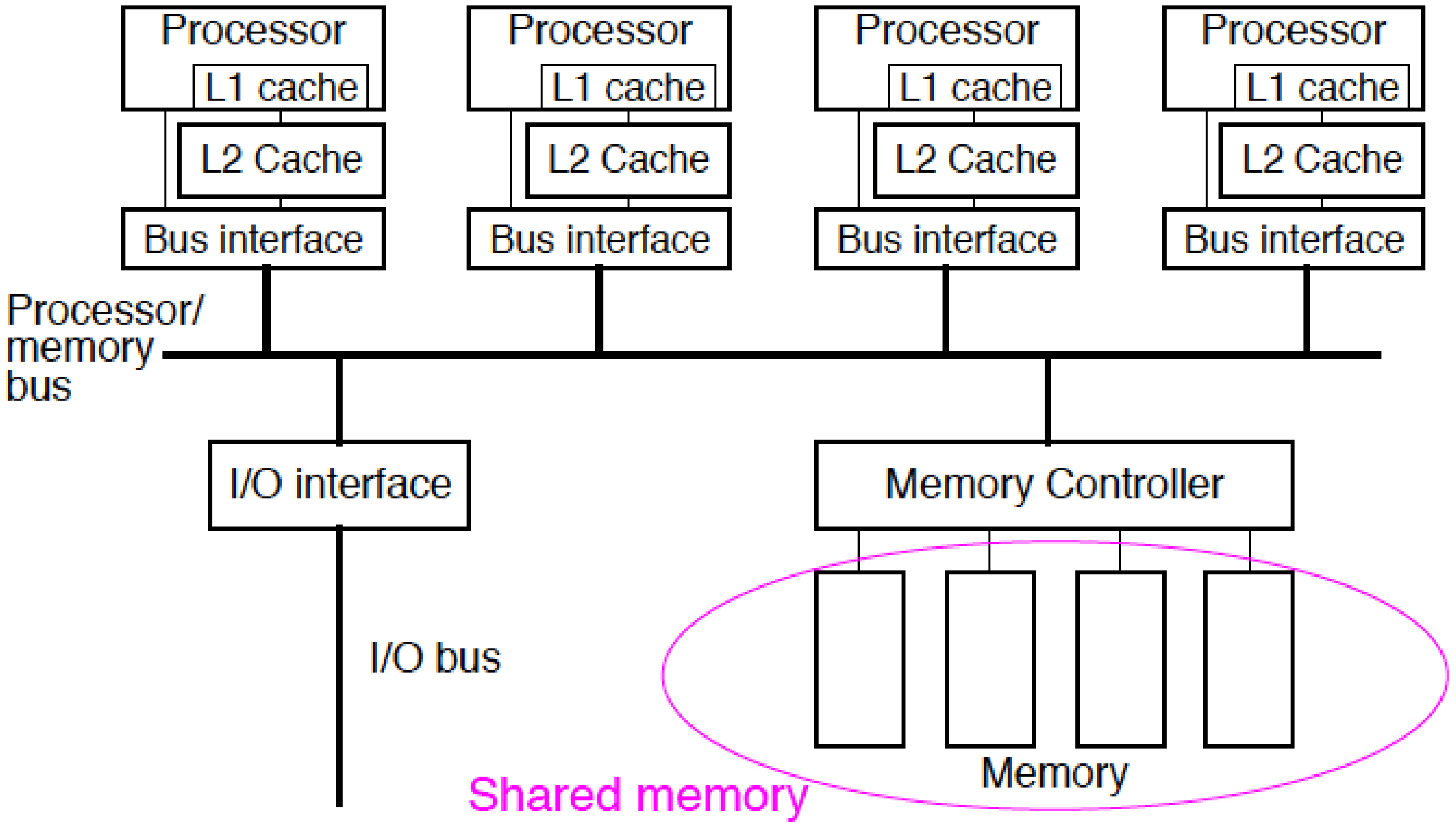
Caching in sistemele cu memorie partajata

- Folosirea memoriilor cache intr-un system de tip SMP introduce probleme legate de ***cache coherency:***
  - Cum se garanteaza faptul ca atunci cand o data este modificata, aceasta modificare este reflectata in celelalte memorii cache si in main memory?

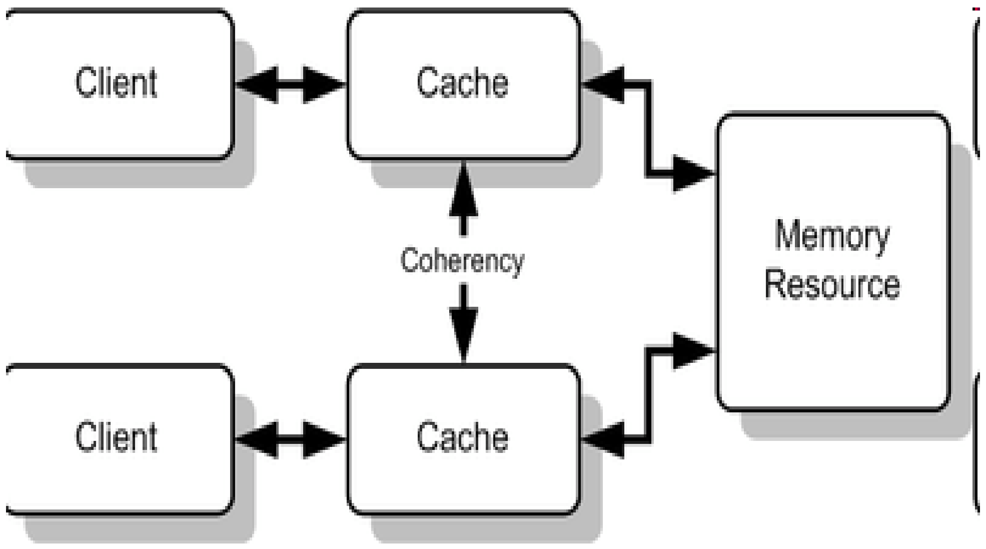


- *coherency* diminueaza scalabilitatea
  - shared memory systems=> maximum 60 CPUs (2016).

Niveluri de caching



Cache coherence



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.

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.

simplist

P_1 P_2

/* Assume initial value of A and flag is 0 */

```
A = 1;                   while (flag == 0); /* spin idly */
flag = 1;                print A;
```

Cache lines and Data Locality

Cache Lines

- The chunks of memory handled by the cache are called cache lines.
- The size of these chunks is called the cache line size.
  Common cache line sizes are 32, 64 and 128 bytes.
- A cache can only hold a limited number of lines, determined by the cache size.
  - For example, a 64 kilobyte cache with 64-byte lines has 1024 cache lines.

Data Locality

- *Temporal locality* means that the program reuses the same data that it recently used, and that therefore is likely to be in the cache.
- *Spatial locality* means that the program uses data close to recently accessed locations. Since the processor loads a chunk of memory around an accessed location into the cache, locations close to recently accessed locations are also likely to be in the cache.

MESI protocol

4 states of a cache line:

- **M - Modified**

The data in the cache line is modified and is guaranteed to only reside in this cache. The copy in main memory is not up to date, so when the cache line leaves the modified state the data must be written back to main memory.

- **E - Exclusive**

The data in the cache line is unmodified, but is guaranteed to only reside in this cache.

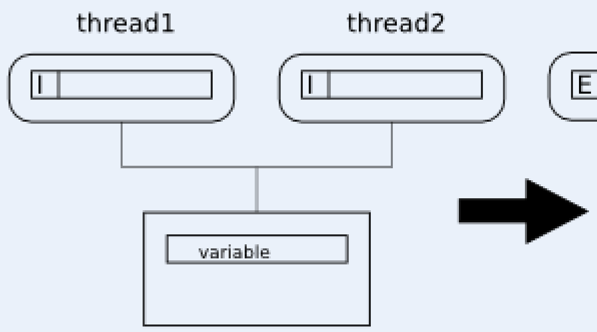
- **S - Shared**

The data in the cache line is unmodified, and there may also be copies of it in other caches.

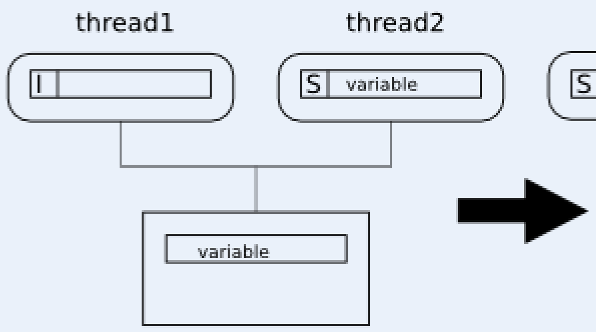
- **I - Invalid**

The cache line does not contain valid data.

Example

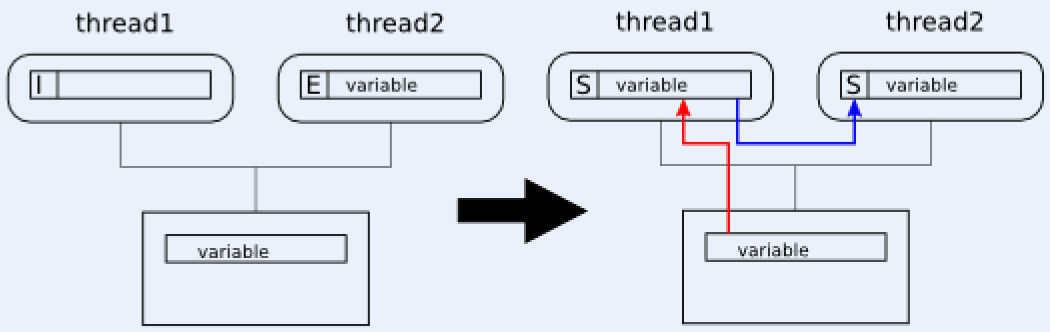


If a thread reads data not present in any cache, it will fetch the line into its cache in exclusive state (E)

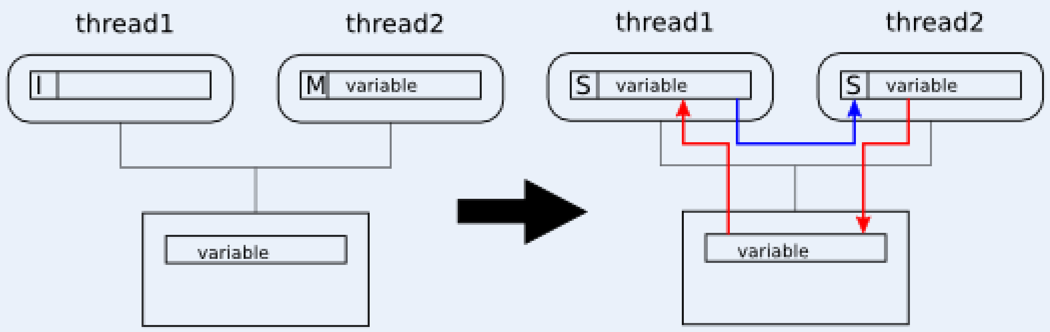


If a thread reads from a cache line that is in shared state (S) in another thread's cache, it fetches the cache line into its cache in shared state (S)

Example

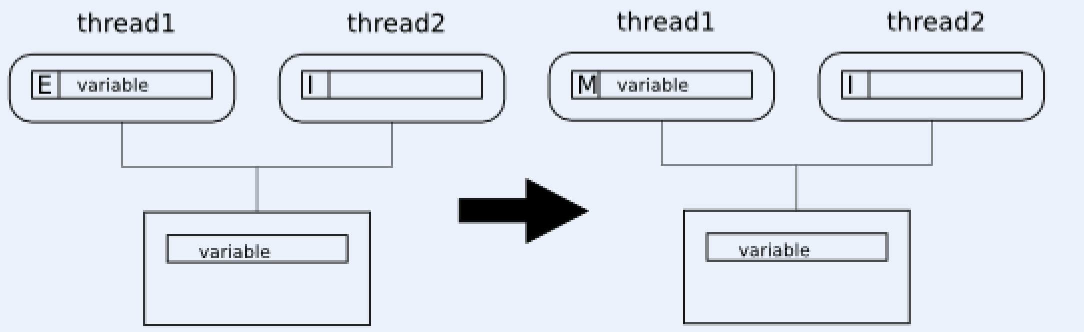


If a thread reads from a cache line that is in exclusive state (E) in another thread's cache, it fetches the cache line to its cache in shared state (S) and downgrades the cache line to shared state (S) in the other cache

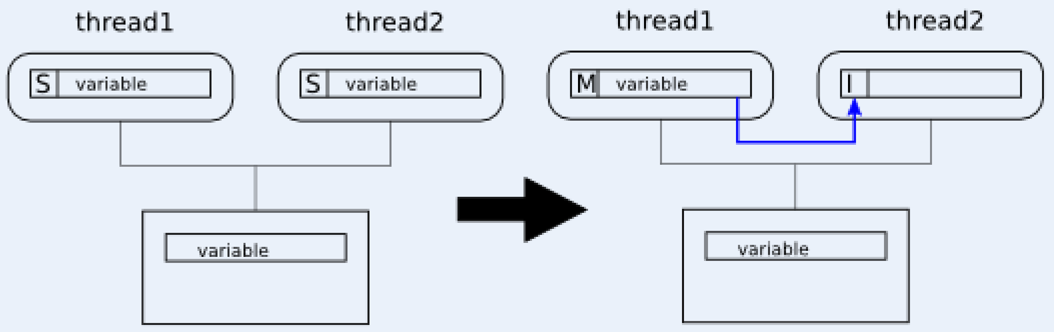


If a thread reads from a cache line that is in modified state (M) in another thread's cache, the other cache must first write-back its modified version of the cache line and downgrade it to shared state (S). The thread doing the read can then add the cache line to its cache in shared state (S)

Example

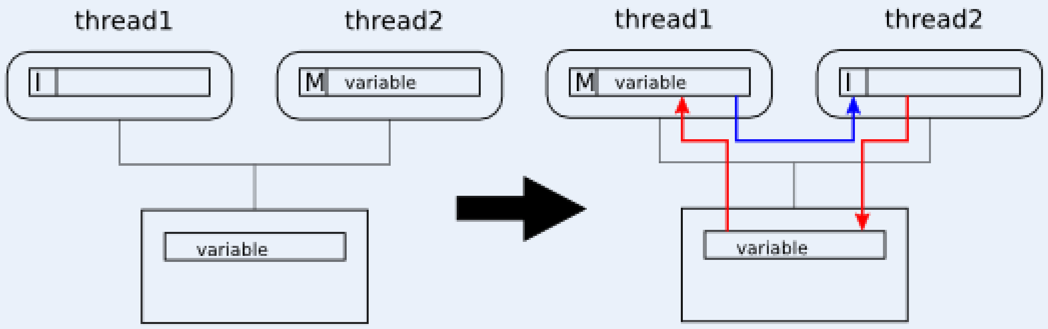


When a thread has a cache line in exclusive (E) or modified state (M) it can write to it with very low overhead, since it knows that no other thread can have a copy of the line that needs to be invalidated. A write to an exclusive cache line makes it modified (M)



When a thread writes to a cache line that it has in shared state (S) it must upgrade the line to modified state (M). In order to do this it must invalidate (I) any copies of the line in other caches, so that they do not retain an outdated copy

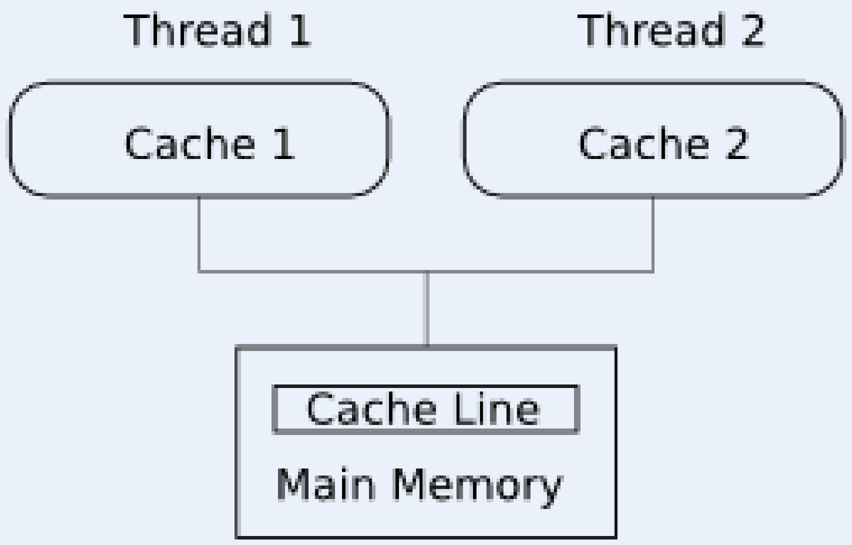
Example



When a thread writes to a cache line that it does not have in its cache, it must fetch the line and invalidate (I) it in all other caches. If another thread has a modified (M) copy of the cache line it must first write it back before the thread doing the write can fetch it.

OBS:
In reality there are other access types to consider !!!
e.g. prefetches and cache line flushes

Example



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 constrange gradul de reordonare a instructiunilor al compilatorului sau al hardware-ului.
- Contract intre programator si sistem.

(Consistenta secventiala) Sequential Consistency

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

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

Shared Memory

Avantaje:

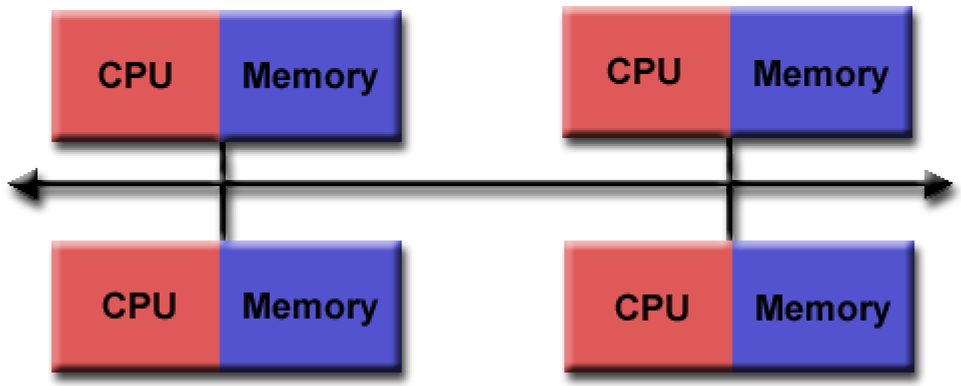
- *Global address space*
- *Partajare date rapida si uniforma*

Dezavantaje:

- Lipsa scalabilitatii
- Sincronizare in sarcina programatorului
- Costuri mari

Arhitecturi cu Memorie Distribuita/*Distributed Memory*

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



Arhitecturi cu memorie distribuita

Avantaje:

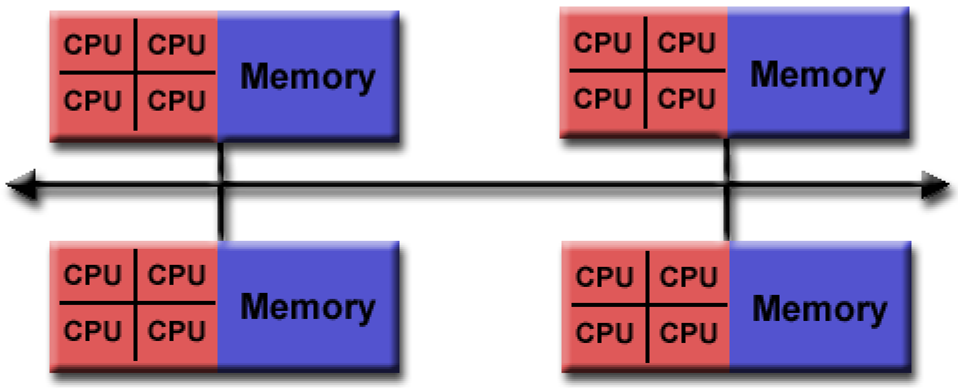
- Memorie scalabila – odata cu cresterea nr de procesoare
- Cost redus – retele

Dezavantaje:

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

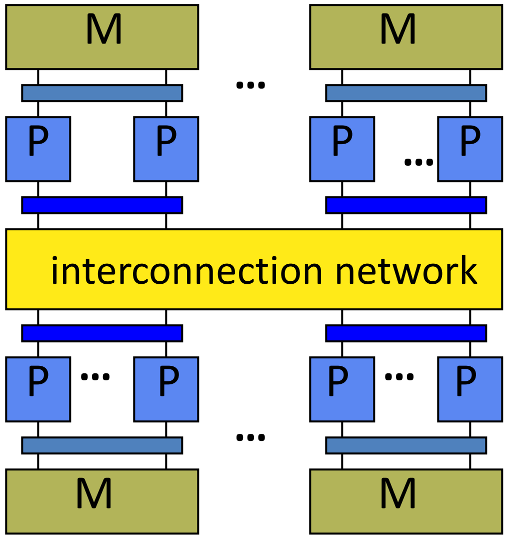
Hybrid Distributed-Shared Memory

- Retea de SMP-uri



SMP(Symmetric multiprocessing) 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



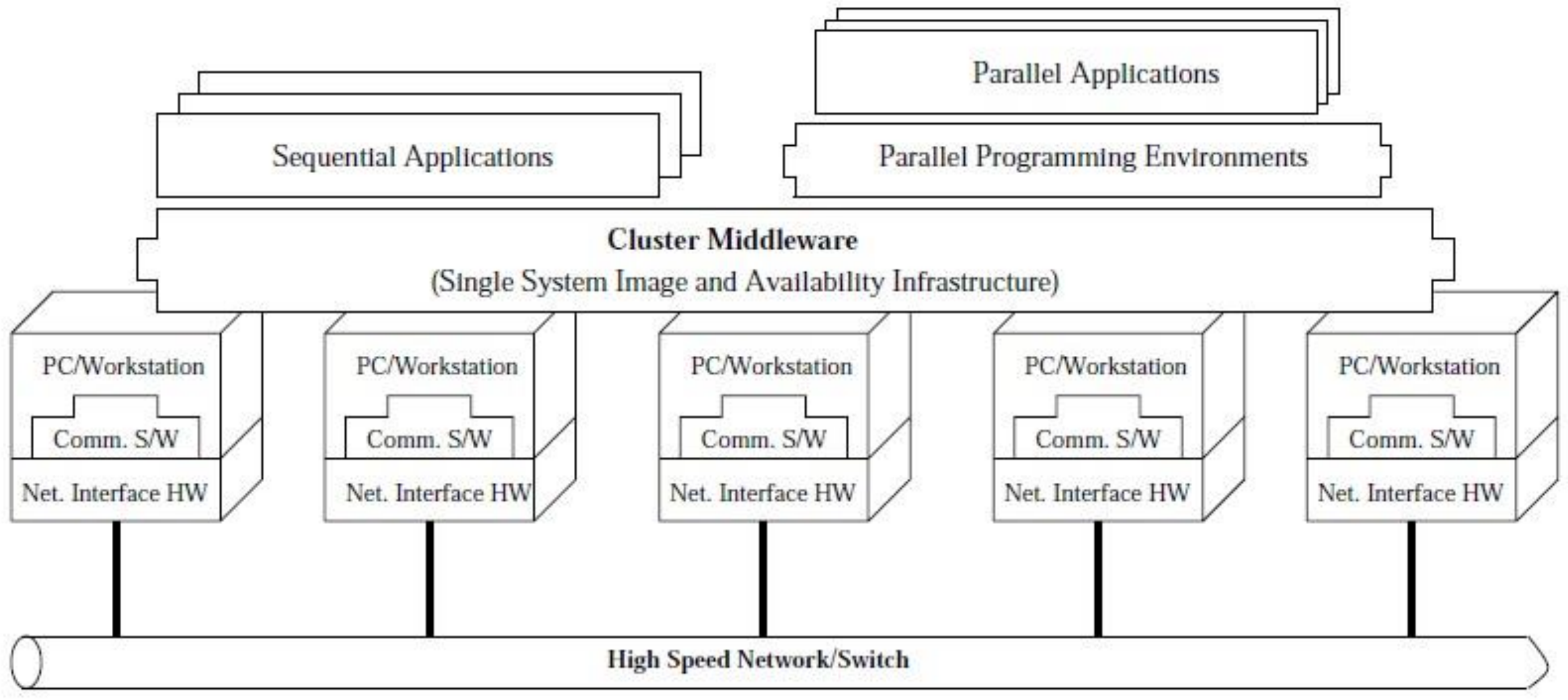
MPP(Massively Parallel Processor)

- 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



Parallel Applications
Sequential Applications
Parallel Programming Environments
Cluster Middleware
(Single System Image and Availability Infrastructure)
PC/Workstation
Comm. S/W
Net. Interface HW
High Speed Network/Switch

Consistency in distributed systems

- In a distributed system (*collection of independent computers that appear to their users as a single coherent system*), data is typically replicated across multiple nodes to improve availability and fault tolerance.
- Consistency is a fundamental property of distributed systems that ensures that all replicas of a shared data store have the same value.
- Maintaining consistency across all replicas can be challenging, especially in the presence of concurrent updates and network delays.

Techniques to implement Sequential Consistency in distributed systems

Two-Phase Locking:

Two-phase locking is a technique used to ensure that concurrent access to shared data is prevented.

In this technique, locks are used to control access to shared data.

Each process acquires a lock before accessing the shared data.

Once a process acquires a lock, it holds the lock until it has completed its operation.

Two-phase locking ensures that no two processes can access the same data at the same time, which can prevent inconsistencies in the data.

Timestamp Ordering:

Timestamp ordering is a technique used to ensure that operations are performed in the same order across all nodes. In this technique, each operation is assigned a unique timestamp.

The system ensures that all operations are performed in the order of their timestamps.

Timestamp ordering ensures that the order of operations is preserved across all nodes.

Quorum-based Replication:

Quorum-based replication is a technique used to ensure that data is replicated across different nodes.

In this technique, the system ensures that each write operation is performed on a subset of nodes.

To ensure consistency, the system requires that a majority of nodes agree on the value of the data.

Quorum-based replication ensures that the data is replicated across different nodes, and inconsistencies in the data are prevented.

Vector Clocks:

Vector clocks are a technique used to ensure that operations are performed in the same order across all nodes.

In this technique, each node maintains a vector clock that contains the timestamp of each operation performed by the node.

The system ensures that all operations are performed in the order of their vector clocks.

Vector clocks ensure that the order of operations is preserved across all nodes.

Sequential Consistency in distributed systems

Challenges

Network Latency: Communication between different nodes in a distributed system can take time, and this latency can result in inconsistencies in the data.

Node Failure: In a distributed system, nodes can fail, which can result in data inconsistencies.

Concurrent Access: Concurrent access to shared data can lead to inconsistencies in the data.

Replication: Replication of data across different nodes can lead to inconsistencies in the data.

Best Practices for Implementing Sequential Consistency

Design for Global Order: Use global timestamps or logical clocks to order operations across processes.

Implement Synchronization Mechanisms: Utilize locks, barriers, or distributed consensus algorithms to coordinate operations and enforce the global order.

Utilize Consensus Algorithms: Apply consensus protocols like Two-Phase Commit (2PC) or Three-Phase Commit (3PC) to agree on the order of transactions.

Ensure Fault Tolerance: Incorporate redundancy and failover mechanisms to maintain consistency during failures.

Optimize Performance: Balance synchronization overhead with performance requirements by batching operations or using efficient consensus algorithms.

Monitor and Test Consistency: Continuously monitor for consistency and perform regular testing to validate that operations adhere to sequential consistency.

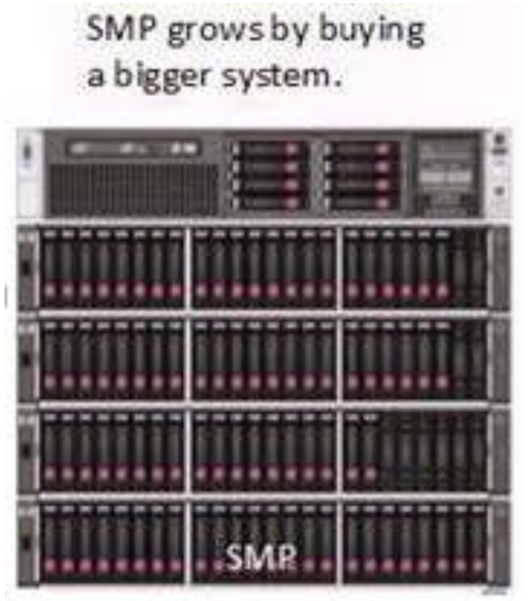
Use Logical Clocks: Implement Lamport timestamps or vector clocks to provide a partial ordering of events.

Scalabilitatea sistemelor de calcul

- 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*)


SMP grows by buying
a bigger system.
SMP

MPP grows by adding
to the existing system.
PDW (MPP)

Performanta

- FLOP= floating point operation
- FLOPS: metrica de performanta => floating point operations per second
  - pentru un calculator
    - FLOPS= cores*cycles_per_sec*FLOPs_per_cycle

- 32-bit (*FP32*) single precision and 64-bit (*FP64*) double precision operations

- Problema: daca un procesor este evaluat la nivel k MFLOPS si sunt p procesoare, este performanta totala de ordin k*p MFLOPS?
- Problema: daca un calcul necesita 100 sec. pe un procesor se va putea face in 10 sec. pe 10 procesoare?

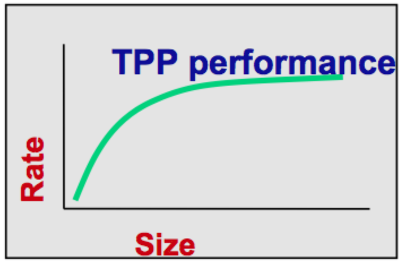
- Cauze care pot afecta performanta
  - Fiecare proc. –unitate independenta
  - Interactiunea lor poate fi complexa
  - *Overhead ...*

  !!! memory accesses and branches are comparatively much more expensive than FLOPS

- *Need to understand performance space*

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*
    - (CPU speed in GHz) x (no of CPU cores) x (CPU instruction per cycle) x (no of CPUs per node) X(no of nodes)
    - Nu poate fi obtinuta in practica.
  - 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



UBB CLUSTER – IBM Intelligent Cluster
http://hpc.cs.ubbcluj.ro/

- Hybrid architecture
  - HPC system +
  - private cloud

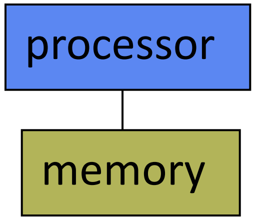
SUMARIZARE

Vedere actuala asupra tipurilor de arhitecturilor paralele

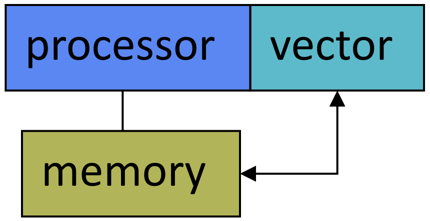
Parallel Architecture Types

imagini preluate de la course pres. Introduction to Parallel Computing CIS 410/510, Univ. of Oregon

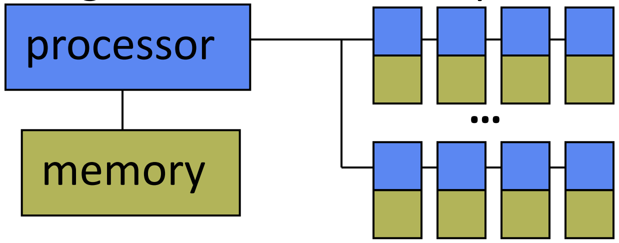
- Uniprocessor
  - Scalar processor



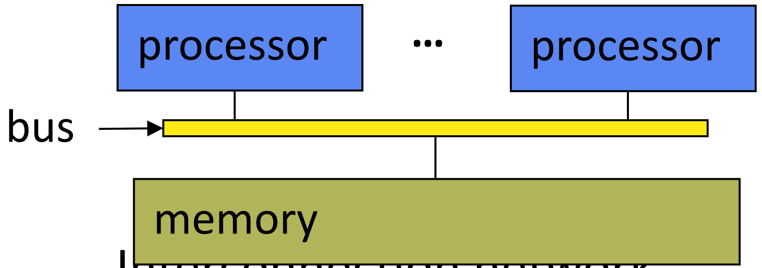
  - Vector processor



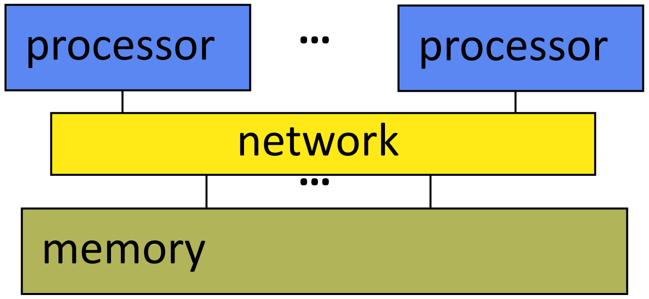
  - Single Instruction Multiple Data



- Shared MemoryMultiprocessor (SMP)
  - Shared memory address space
  - Bus-based memory system

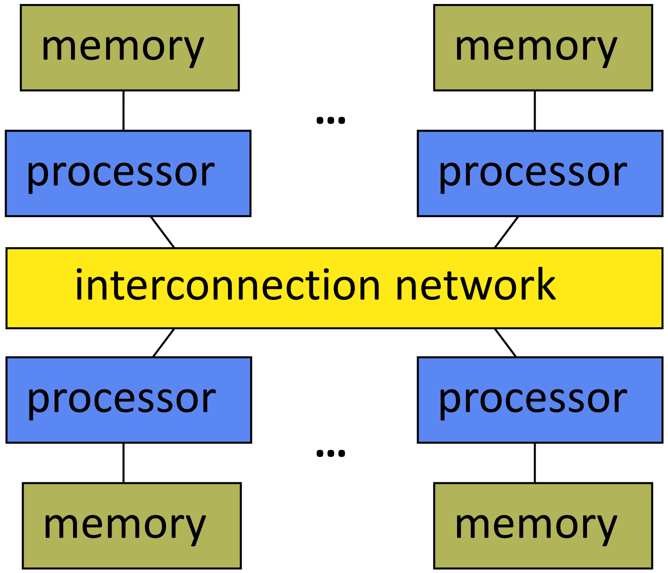


  - Interconnection network



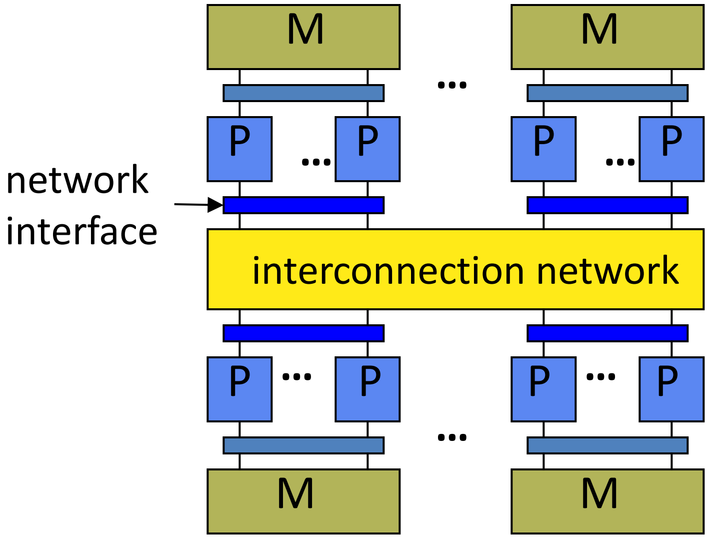
Parallel Architecture Types (2)

- Distributed Memory Multiprocessor
  - Message passing between nodes



  - Massively Parallel Processor (MPP)
    - many, many processors
    - fast interconnection

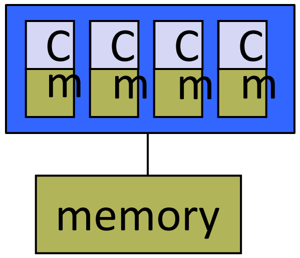
- Cluster of SMPs
  - Shared memory addressing within SMP node
  - Message passing between SMP nodes



  - Can also be regarded as MPP if processor number is large and the communication is fast

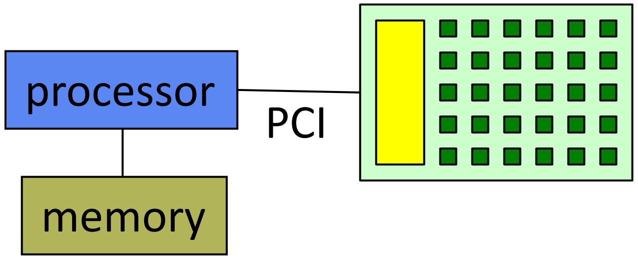
Parallel Architecture Types (3)

- Multicore
  - Multicore processor

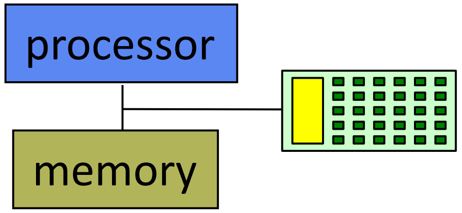


cores can be hardware multithreaded (hyperthread)

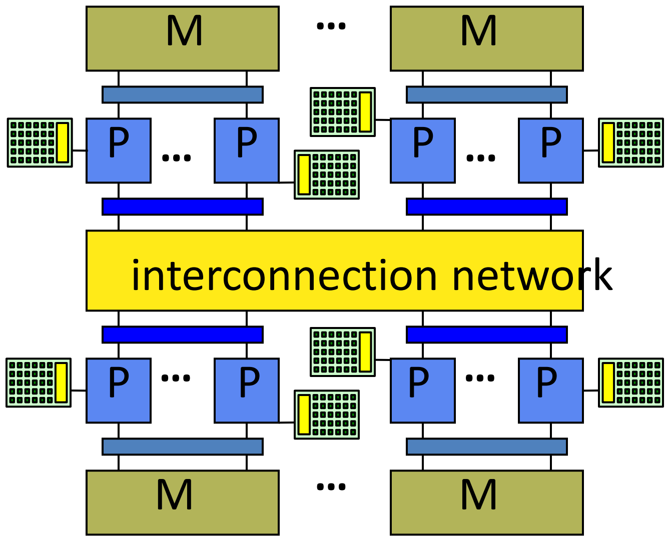
  - GPU accelerator



  - "Fused" processor accelerator



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


M
…
M
P
…
P
P
…
P
interconnection network
P
…
P
P
…
P
M
…
M