Corso di Laurea in Informatica

Dipartimento di Scienze Fisiche, Informatiche e Matematiche

UNIVERSITÀ DEGLI STUDI
DI MODENA E REGGIO EMILIA

Corso
Calcolo Parallelo
Esercitazioni

Titolare del corso: prof. Luca Zanni (luca.zanni@unimore.it)

AA 2018/2019

Presentazione del modulo di esercitazioni.
La necessità di sistemi ad alte prestazioni.
Tipi di parallelizzazione ed architetture parallele

Slide e testi di riferimento

• Slide originariamente create e gentilmente concesse dal prof. **Andrea Bertoni**.

• Modifiche a cura del prof. **Luca Zanni**, dott. **Paolo Burgio** e dott. **Roberto Cavicchioli**

TESTI UTILI

T.M.R. Ellis, I.R. Philips, T.M. Lahey, Fortran 90 Programming, Addison Wesley (1994)

W. Gropp, E. Lusk, A. Skjellum, Using MPI, The MIT Press (2000)

P. S. Pacheco, Parallel Programming with MPI, Morgan Kaufmann Pub. (1996)

M. Herlihy, N. Shavit, The Art of Multiprocessor Programming, Elsevier (2008)

A. Grama, G. Karypis, V. Kumar, A. Gupta, An introduction to Parallel Computing: Design and Analysis of Algorithms, Addison-Wesley (2003)

J.L. Hennessy, D.A. Patterson, Computer Architecture: A Quantitative Approach, 3rd edition, Morgan Kaufmann Pub. (2002)

D.E. Culler, J.P. Singh, A. Gupta, Parallel Computer Architecture, Morgan Kaufmann (1998)

Cosa imparerete

- Cos'è un calcolatore parallelo? Quali sono le tipologie più diffuse?
- Come è fatto un calcolatore parallelo?
- Come valuto la performance di un sistema parallelo?
- Devo implementare un algoritmo: mi conviene farlo su un sistema parallelo? Di che tipo?
- Quali strumenti posso utilizzare (librerie parallele, solutori paralleli…)?
- Librerie **MPI**.
- Leggere e capire i MANUALI di sistemi paralleli, software parallelizzato e software per la gestione di sistemi paralleli.
- Paradigma e librerie **OPENMP**
- Programmazione GP-GPU**, CUDA** e **OpenCL**

ESEMPI DI SISTEMI PARALLELI NELLE VICINANZE…

cluster di PC
(lo potete fare in casa)

Proc: qualsiasi, meglio uguali.
Nodo Master
Dischi locali + Master
Sistema Operativo: Linux
Rete veloce (almeno Gbit)
Lib parallele: MPICH

IBM SP3
(laboratorio CSAI UniMoRe)

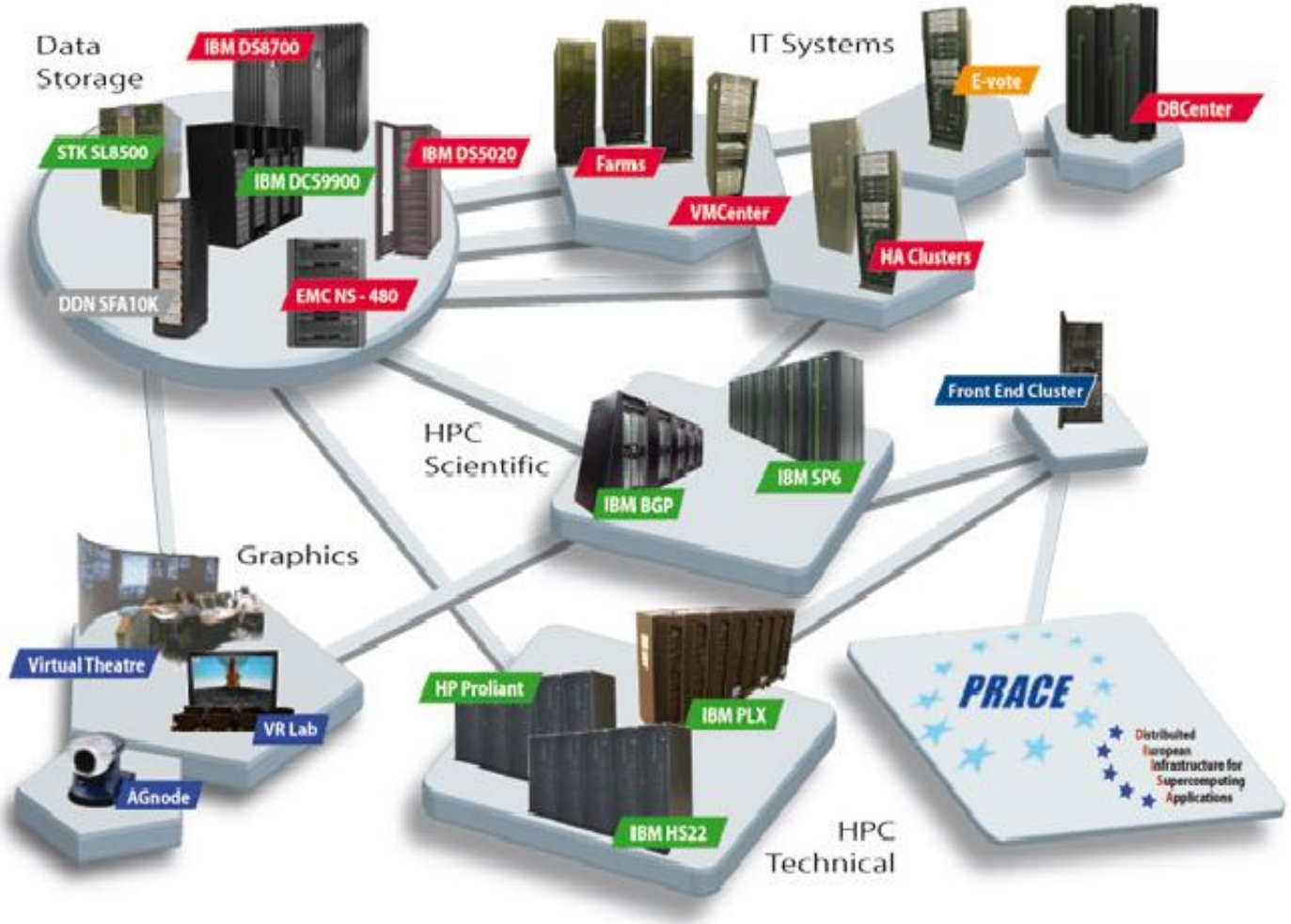
Processori: Power3 375 MHz
6 Nodi, Memoria 4 Gb / nodo
Dischi 36 Gb per nodo
Sistema Operativo AIX 4.3
Anno: 2000
(sostituito da un cluster Linux)

Cluster IBM
(Centro S3 INFM-CNR)

Processori: Xeon ~ 3 GHz
12 (32) Nodi biproc
Memoria ~ 8 Gb / nodo
Dischi 36 Gb per nodo
Sistema Operativo Linux
Anno: 2002 (dismesso)

ESEMPI DI SISTEMI PARALLELI NELLE VICINANZE…

CINECA (Casalecchio di Reno, Bologna)
PLX-GPU, SP6



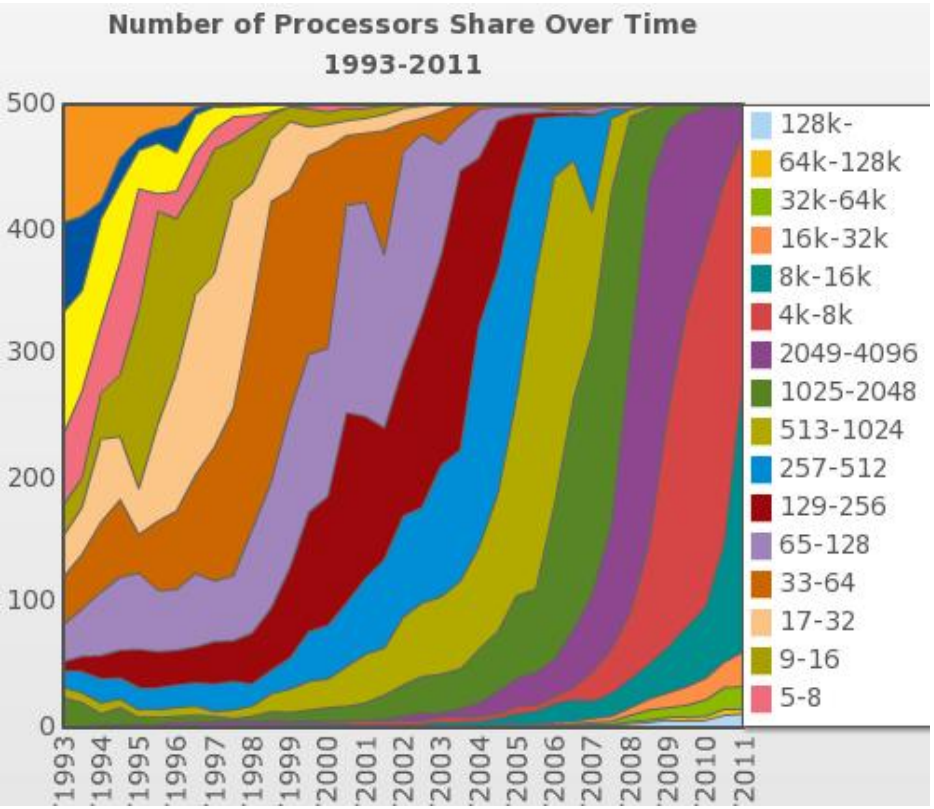
ESEMPI DI SISTEMI PARALLELI NELLE VICINANZE…

CINECA (Casalecchio di Reno, Bologna)
PLX-GPU, SP6

Rank	Site	System	Cores	R_{max}	R_{peak}
54	CINECA / SCS - SuperComputing Solution Italy	iDataPlex DX360M3, Xeon 2.4, nVidia GPU, Infiniband IBM	3072	142.7	293.27
60	eni Italy	HP ProLiant SL390s G7 Xeon 6C X5650, Infiniband Hewlett-Packard	15360	131.2	163.43
116	CINECA Italy	Power 575, p6 4.7 GHz, Infiniband IBM	5376	78.68	101.07
129	Energy Company (A) Italy	BladeCenter HS22 Cluster, Xeon QC X55xx 2.66 GHz, Infiniband IBM	10224	73.64	109.03

ESEMPI DI SISTEMI PARALLELI

TOP500.ORG

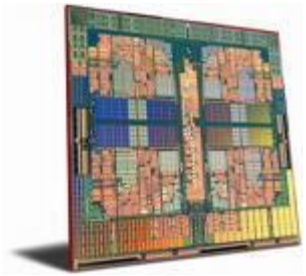


Rank	Site	Computer/Year Vendor	Cores	R_{max}	R_{peak}	Power
1	RIKEN Advanced Institute for Computational Science (AICS) Japan	K computer, SPARC64 VIIIfx 2.0GHz, Tofu interconnect / 2011 Fujitsu	548352	8162.00	8773.63	9898.56
2	National Supercomputing Center in Tianjin China	Tianhe-1A - NUDT TH MPP, X5670 2.93Ghz 6C, NVIDIA GPU, FT-1000 8C / 2010 NUDT	186368	2566.00	4701.00	4040.00
3	DOE/SC/Oak Ridge National Laboratory United States	Jaguar - Cray XT5-HE Opteron 6-core 2.6 GHz / 2009 Cray Inc.	224162	1759.00	2331.00	6950.60
4	National Supercomputing Centre in Shenzhen (NSCS) China	Nebulae - Dawning TC3600 Blade, Intel X5650, NVidia Tesla C2050 GPU / 2010 Dawning	120640	1271.00	2984.30	2580.00
5	GSIC Center, Tokyo Institute of Technology Japan	TSUBAME 2.0 - HP ProLiant SL390s G7 Xeon 6C X5670, Nvidia GPU, Linux/Windows / 2010 NEC/HP	73278	1192.00	2287.63	1398.61
6	DOE/NNSA/LANL/SNL United States	Cielo - Cray XE6 8-core 2.4 GHz / 2011 Cray Inc.	142272	1110.00	1365.81	3980.00
7	NASA/Ames Research Center/NAS United States	Pleiades - SGI Altix ICE 8200EX/8400EX, Xeon HT QC 3.0/Xeon 5570/5670 2.93 Ghz, Infiniband / 2011 SGI	111104	1088.00	1315.33	4102.00
8	DOE/SC/LBNL/NERSC United States	Hopper - Cray XE6 12-core 2.1 GHz / 2010 Cray Inc.	153408	1054.00	1288.63	2910.00
9	Commissariat a l'Energie Atomique (CEA) France	Tera-100 - Bull bullx super-node S6010/S6030 / 2010 Bull SA	138368	1050.00	1254.55	4590.00
10	DOE/NNSA/LANL United States	Roadrunner - BladeCenter QS22/LS21 Cluster, PowerXCell 8i 3.2 Ghz / Opteron DC 1.8 GHz, Voltaire Infiniband / 2009 IBM	122400	1042.00	1375.78	2345.50
11	National Institute for Computational Sciences/University of Tennessee United States	Kraken XT5 - Cray XT5-HE Opteron Six Core 2.6 GHz / 2011 Cray Inc.	112800	919.10	1173.00	3090.00
12	Forschungszentrum Juelich (FZJ) Germany	JUGENE - Blue Gene/P Solution / 2009 IBM	294912	825.50	1002.70	2268.00
13	Moscow State University - Research Computing Center Russia	Lomonosov - T-Platforms T-Blade2/1.1, Xeon X5570/X5670 2.93 GHz, Nvidia 2070 GPU, Infiniband QDR / 2011 T-Platforms	33072	674.11	1373.06	

TOP500 List - June 2011

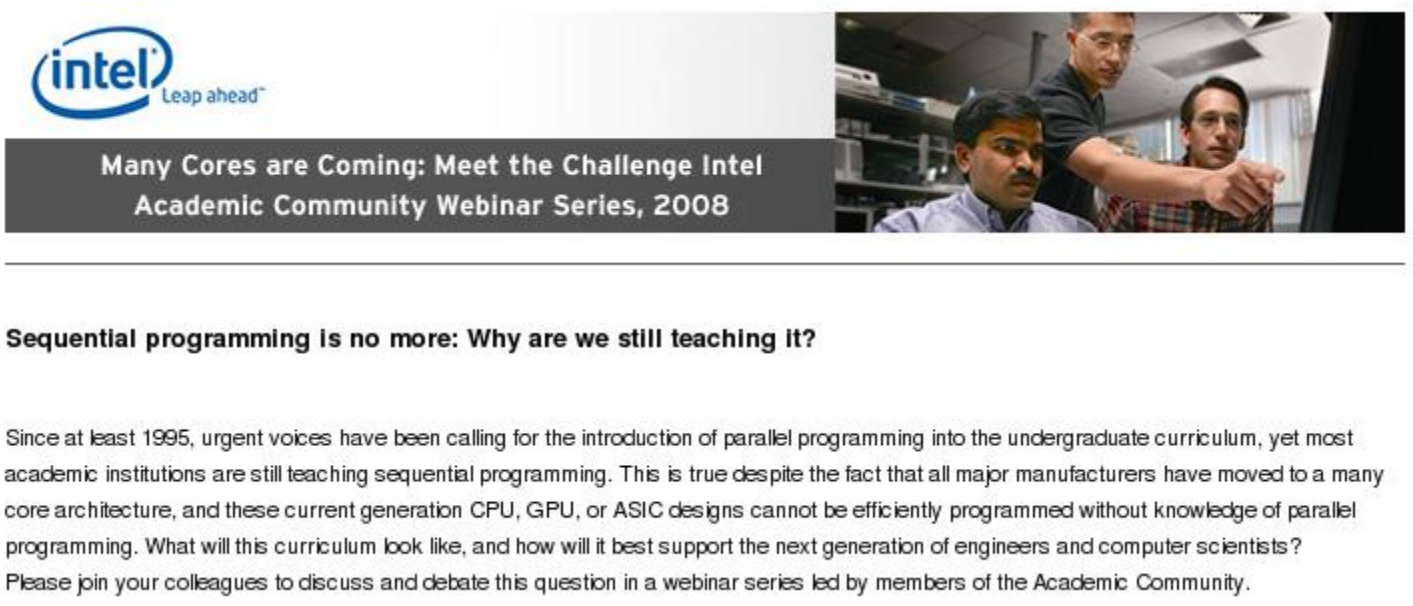
… ma anche nei vostri PC

Intel Core2 quad core



AMD Opteron quad core

NVIDIA Tesla GPU



LE ARCHITETTURE PARALLELE: MOTIVAZIONI

Domanda: Perché si sviluppano sistemi paralleli ?
Risposta: Necessita di elevate prestazioni di calcolo.

Chi ne ha bisogno:

- Ricercatori (fisici, chimici, biologi…)
- Industrie (modeling di sistemi e dispositivi, data mining)
- ma anche… Sport Team (LunaRossa, Ferrari), Entertainment (effetti speciali, video texture), Security (data mining, pattern recognition), ecc.

L'incremento di prestazioni introdotto da un sistema parallelo può essere visto come:

SCALE-UP : nello stesso tempo si risolve un problema più grande

SPEED-UP : lo stesso problema si risolve in minor tempo

l'uso di sistemi paralleli rappresenta l'unica possibilità per affrontare le grandi simulazioni del mondo fisico (Grand Challenges)

LE ARCHITETTURE PARALLELE: MOTIVAZIONI

Domanda: Ma perché sistemi paralleli e non un sistema seriale molto potente ?

Risposta 1: I processori hanno dei limiti intrinseci (tempo di switching dei dispositivi, velocità di propagazione del segnale, heat dissipation, ecc.). Ad esempio nel 2001 si dovettero allocare 2 stage dei circa 20 presenti nella pipeline del Pentium 4 per la propagazione del segnale nel chip.

Risposta 2: Spesso è più economico costruire sistemi multiprocessore, eventualmente basati su CPU a basso costo già in commercio.

Cerchiamo di essere più quantitativi sui costi…

Premessa: un microprocessore si produce partendo da un singolo cristallo di silicio (figura a sinistra) ed "affettandolo", producendo *wafer* circolari (figura a destra) che vengono sezionati in *die* quadrati.

ANALISI DELLE PRESTAZIONI

La “velocità” di un sistema è misurata da:

tempo di esecuzione
(quanto impiega ad eseguire un lavoro)
throughput
(quanto lavoro esegue in un dato tempo)

Quale tempo dobbiamo considerare ? Guardiamo l’output del comando time di unix:

```
[user@system]$ time ./executable.x
   running…
   …end.
52.45u 10.04s 1:24  53%
[user@system]$
```

otteniamo:

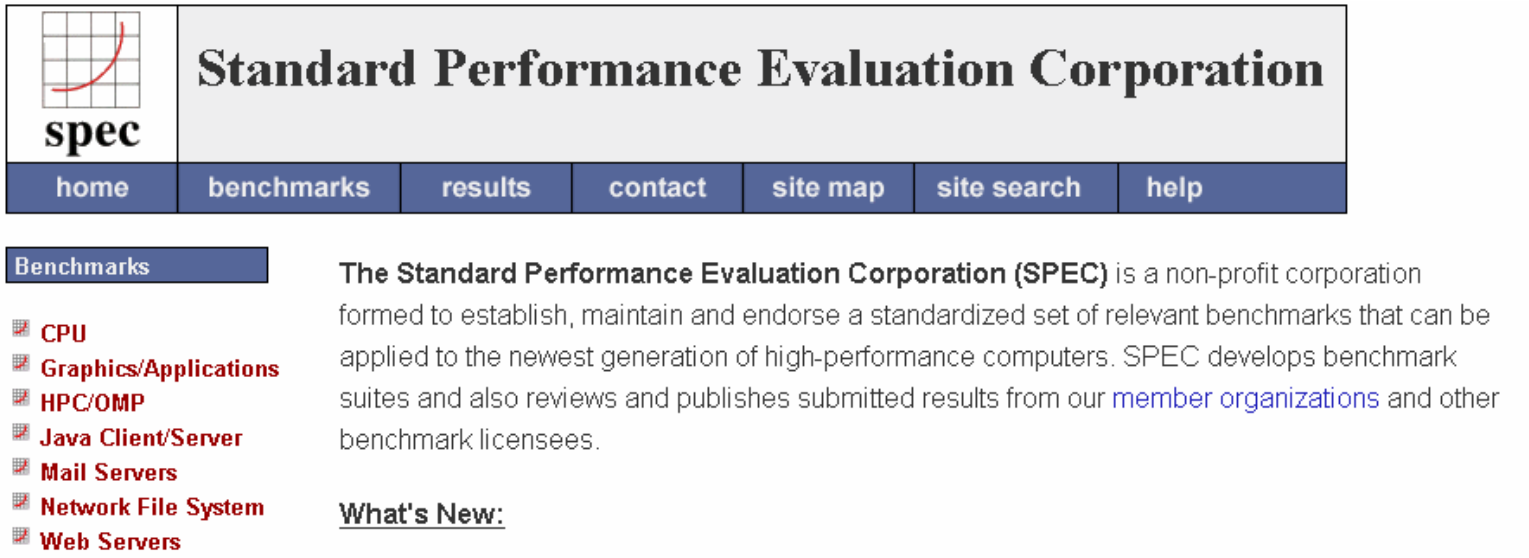
tempo di CPU d’utente (u: user time)
tempo di CPU di sistema (s: system time)
tempo effettivo trascorso (minuti:secondi)
percentuale del tempo di CPU sul tempo effettivo trascorso (quanto ha lavorato la CPU)

ANALISI DELLE PRESTAZIONI

Che programma usiamo per valutare le prestazioni di un sistema ?
La scelta migliore è usare il codice specifico che dovrà girare sul sistema. (APE)
…ma spesso non lo conosciamo a priori o sono molti i codici che useremo:
ci serve una valutazione indipendente dal programma.

Benchmark

Programmi reali (gcc, TeX, Spice), Kernel (LivermooreLoops, Linpack),
Benchmark elementari (Puzzle, Quicksort), Benchmark artificiali (varie raccolte)



ANALISI DELLE PRESTAZIONI

Ovviamente per il benchmark di un sistema di calcolo si dovrà scegliere un test mirato per la stima della potenza di calcolo della **CPU**. Inoltre sono utili test per **memoria** e **dischi**. Meno significativi quelli mirati a schede video o applicazioni multimedia.

Nel caso si voglia stimare la prestazione di un sistema parallelo si dovranno utilizzare benchmark appositi.

Una caratteristica chiave da valutare in un sistema parallelo è la SCALABILITA' (scalability)

Questa ha vari aspetti:

1) possibilità (ed economicità) di aggiungere altri PE
2) incremento delle prestazioni generali (benchmark) all'aggiunta di PE
3) incremento delle prestazioni di un codice all'aggiunta di PE

come vedremo, l'ultimo punto dipende principalmente dal codice

ANALISI DELLE PRESTAZIONI

legge di Amdahl: l'incremento di prestazioni derivante dalla miglioria di una parte di un sistema (o codice) è limitato dalla frazione di tempo in cui esso viene usato

ci insegna che è meglio rendere veloce (nel nostro caso: parallelizzare) la parte più usata

te_{old} : total exec time without the enhancement
te_{new} : total exec time with the enhancement
$ts = (te_{old} / te_{new})$: total speed-up
pe_{old} : (partial) exec time of the original component =1
pe_{new} : (partial) exec time of the enhanced component
$ps = (pe_{old} / pe_{new})$: (partial) speed-up of the enhanced component
f : fraction of time in which the component is used

ps si suppone noto

$$te_{new} = te_{old} [(1 – f) + (f / ps)]$$

$$ts = te_{old} / te_{new} = 1 / [(1 – f) + (f / ps)]$$

ANALISI DELLE PRESTAZIONI

legge di Amdahl: ESEMPIO

supponiamo che un sottosistema (o una subroutine di un codice) sia responsabile del 30% del tempo di esecuzione e supponiamo di essere in grado di incrementarne le prestazioni di un fattore 10

$pe_{old} = 1$
$pe_{new} = 0.1$
$f = 0.3$

$$ts = te_{old} / te_{new} = 1 / [(1 – f) + (f / ps)]$$

$$ts = 1 / [(1 – 0.3) + (0.3 / 10)]$$

$$ts = 1.37$$

ANALISI DELLE PRESTAZIONI

principio di località:

- i programmi tendono a riusare i dati usati di recente (località temporale)
- i programmi tendono ad usare insieme dati vicini (località spaziale)

Questo è il principio alla base dell'incremento di prestazioni ottenuto con l'introduzione di gerarchie di memoria (cache).

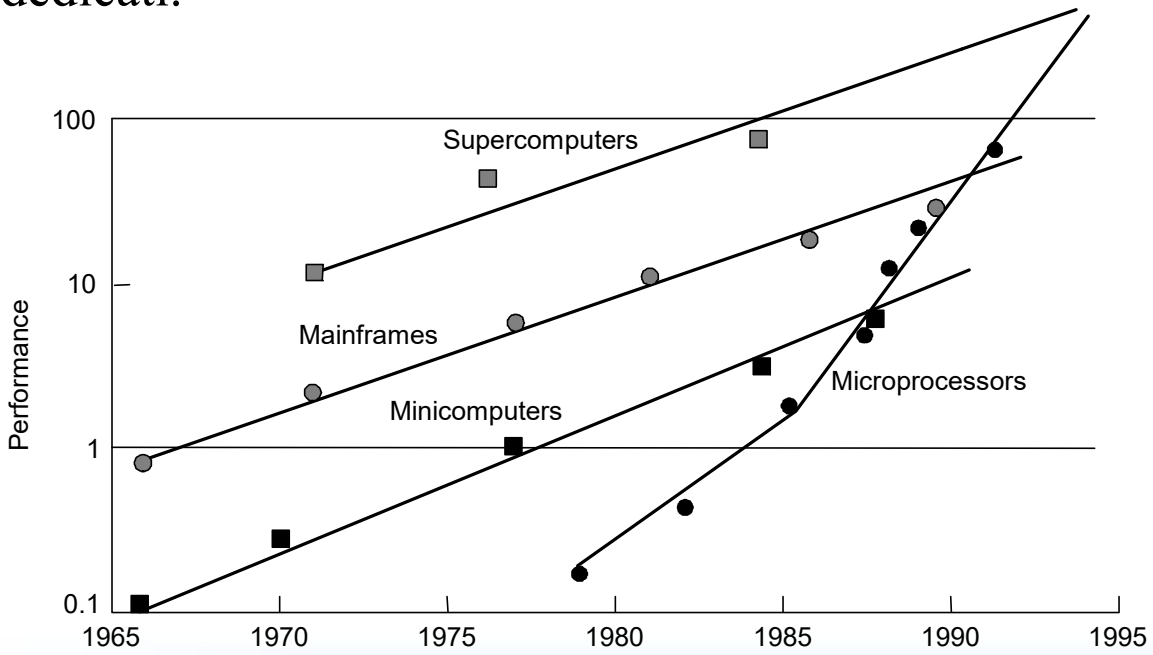
parallelismo:

- parallelismo **a livello di sistema**: vengono usati processori (dischi, memorie) multipli per incrementare il throughput totale, ad esempio per calcoli ad elevate prestazioni o server ad alto carico
- parallelismo **a livello di istruzione**: pipelining
- parallelismo **a livello di architettura** digitale: set-associative cache istanziate in parallelo, carry-lookahead

Noi ci occuperemo essenzialmente del parallelismo a livello di sistema.

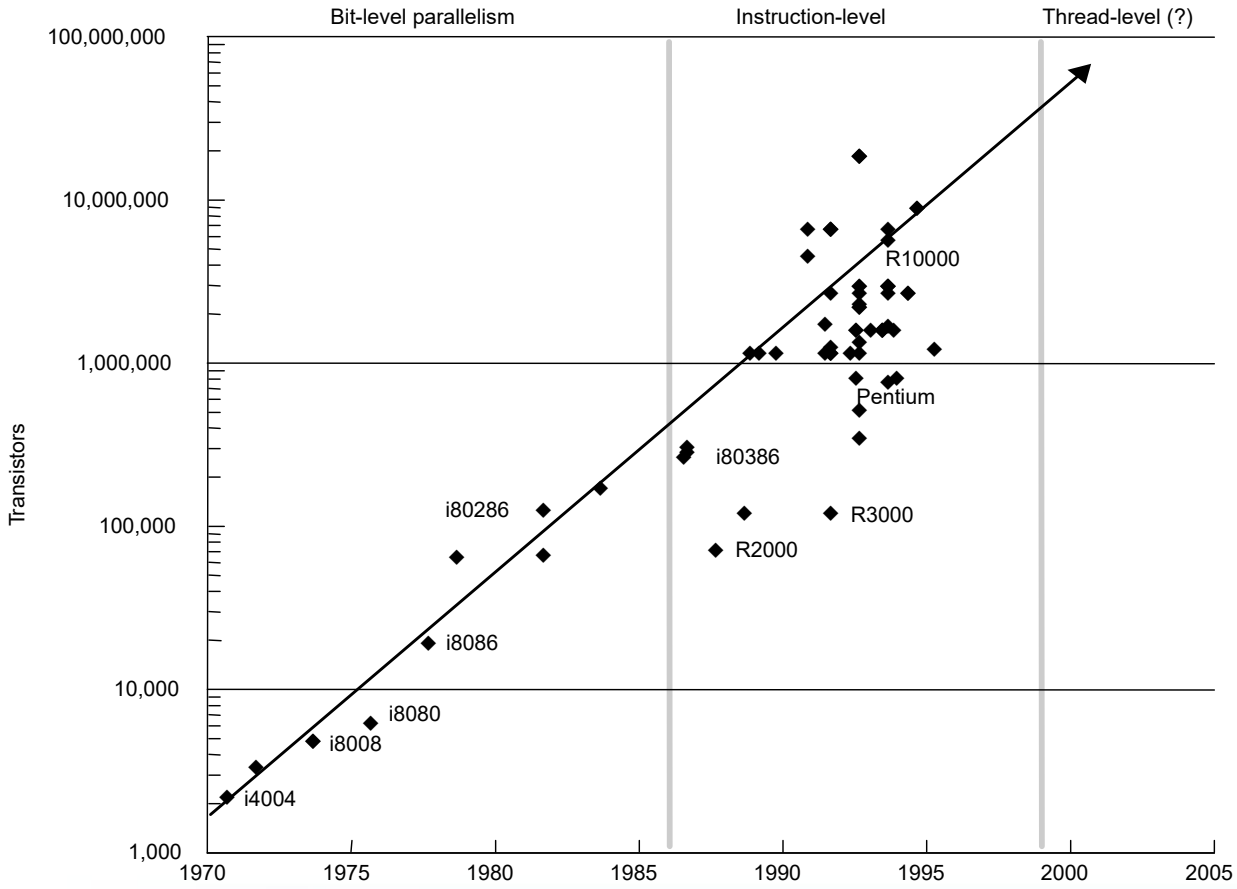
Tassonomia delle architetture parallele

Cominciamo ad occuparci del parallelismo a livello di sistema esaminando le tipologie possibili per gli elaboratori paralleli.
Il campo delle architetture parallele è ancora molto giovane ed è difficile fare previsioni sulle architetture del futuro anche prossimo. Possiamo analizzare il trend della tecnologia: fino a 10 anni fa sembrava che la potenza dei singoli microprocessori dovesse superare il trend di supercomputer tradizionali. Non sempre ciò si è rivelato esatto. Poi si sono diffusi i cluster basati su processori non dedicati.



Tassonomia delle architetture parallele

Lo sfruttamento del parallelismo ha visto varie fasi: ora siamo entrati in quella in cui si comincia a sfruttare il thread-level parallelism.



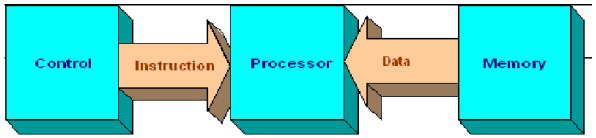
Tassonomia delle architetture parallele

La classificazione tradizionale delle architetture parallele è quella di Flynn (1966).

	single instruction stream	multiple instruction stream
single data stream	**SISD**	**MISD**
multiple data stream	**SIMD**	**MIMD**

Tassonomia delle architetture parallele

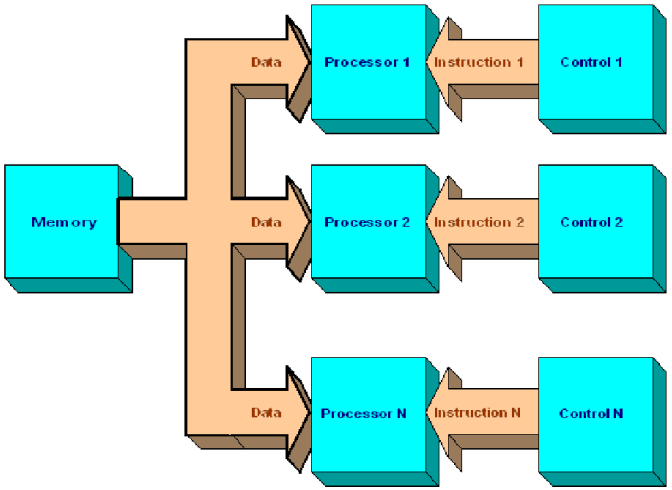
	single instruction stream	multiple instruction stream
single data stream	**SISD**	
multiple data stream		



SISD : sono le architetture a singolo processore

Tassonomia delle architetture parallele

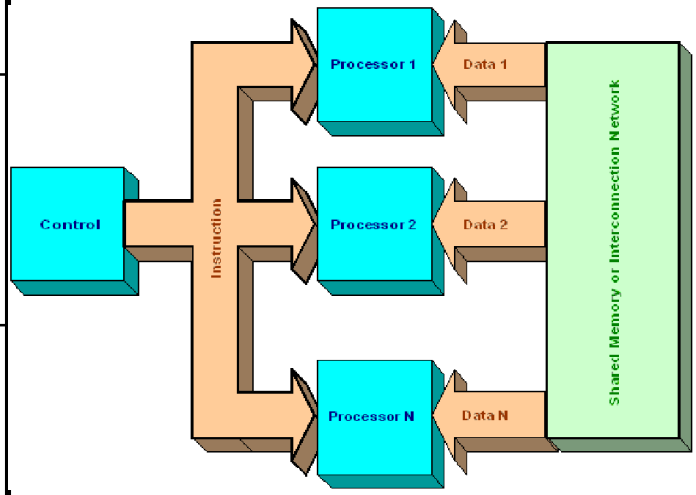
	single instruction stream	multiple instruction stream
single data stream		**MISD**
multiple data stream		



MISD : architetture Multiple Instruction Single Data, ridondanza.

Tassonomia delle architetture parallele

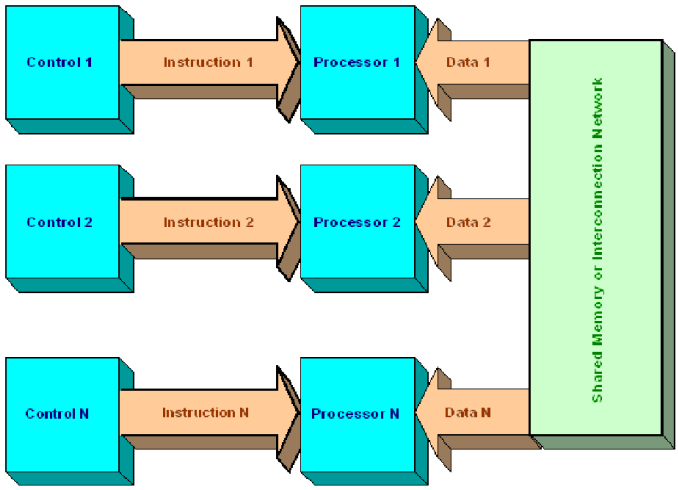
	single instruction stream	multiple instruction stream
single data stream		
multiple data stream	**SIMD**	



SIMD : nelle architetture Single Instruction Multiple Data la stessa istruzione è eseguita (eventualmente da vari PE) su dati differenti.
es.: Multimedia extensions. Processori vettoriali.

Tassonomia delle architetture parallele

	single instruction stream	multiple instruction stream
single data stream		
multiple data stream		**MIMD**

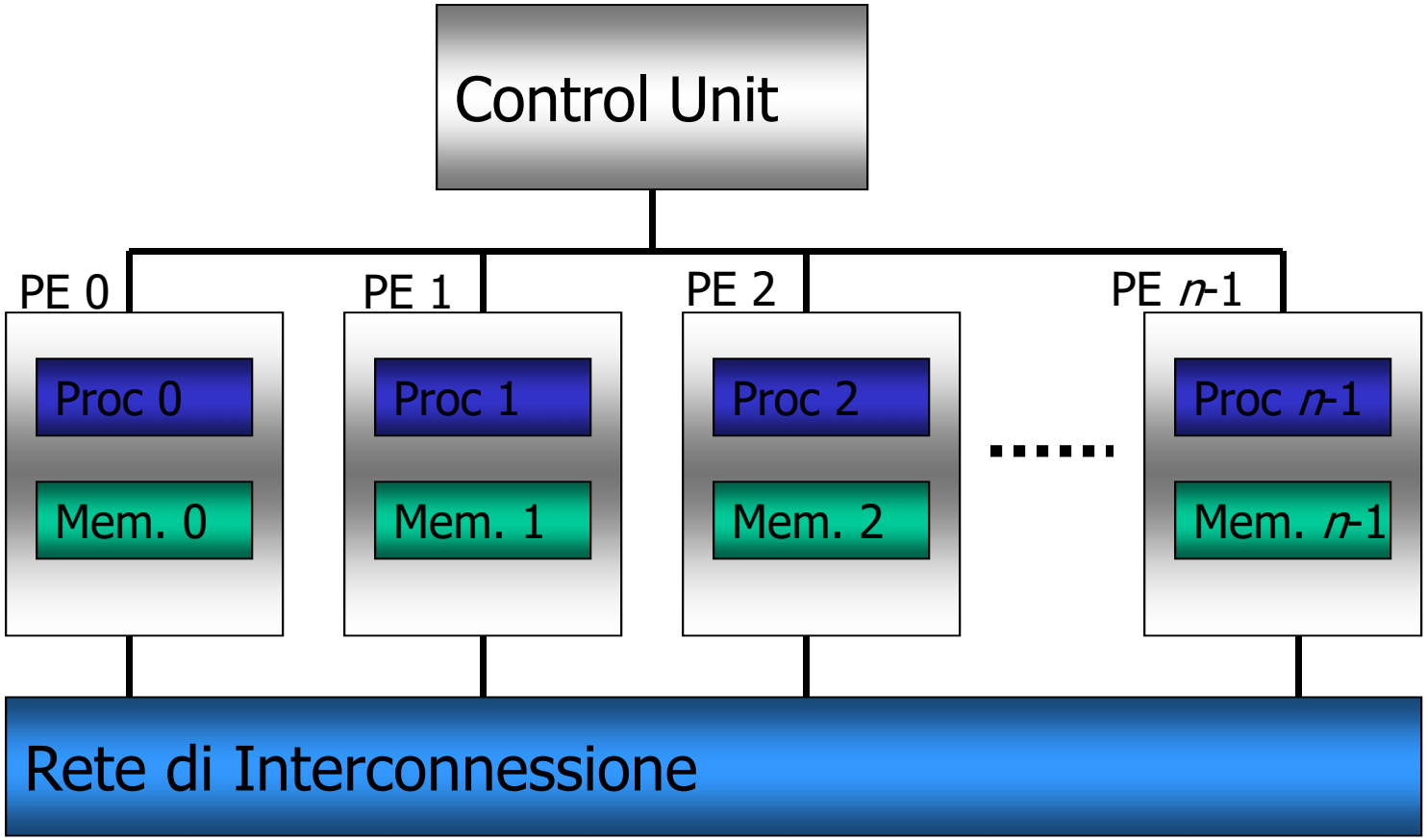


MIMD : ogni processore ha proprie istruzioni e propri dati da elaborare. E' la classe di gran lunga più diffusa che vedremo in dettaglio.

Le architetture SIMD

Le architetture SIMD

Ogni PE ha la propria data-memory mentre lo stream delle istruzioni è condiviso



PE = *Processing Element*

Le architetture SIMD

Nella categoria delle macchine SIMD rientrano o possono rientrare:

1. **Supercomputer vettoriali** (i registri vettoriali mediante cui è possibile effettuare computazioni algebriche elementari in un unico ciclo macchina "assomigliano" ai PE di un'architettura SIMD)

2. **Sistemi a parallelismo massiccio**, costituiti da molti (ma semplici) processori quali:
   - TMC Connection Machine (65.536 processori)
   - AMD DAP (4.096 processori)
   - MarsPar MP-1 (16.384 processori)

3. **Macchine SIMD *special-purpose***:
   - IBM GF-11 (11 Gflops peak)
   - INFN APE-100 (6 Gflops peak, nel 1991)

4. **Le estensioni (o i coprocessori) per le applicazioni multimediali**:
   - Alpha MAX
   - Intel MMX

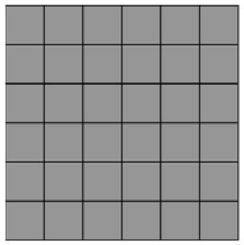
Le architetture SIMD

La programmazione di una architettura SIMD è un modello data-parallel sincrono

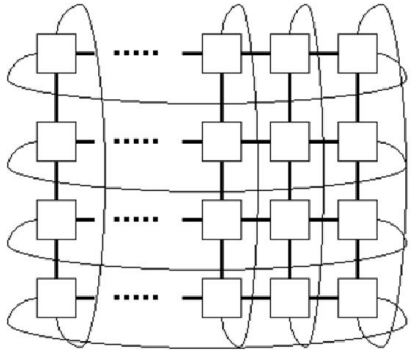
(1) richiede come primo passo una distribuzione del dominio dei dati tra tutti i PE

- Si ha una *granularità fine* (*fine grain*) o *finissima* della distribuzione dei dati in quanto il numero delle unità di elaborazione è, in genere, elevato e ciascun nodo è molto semplice,
- Per far sì che la programmazione SIMD sia efficiente, la distribuzione del dominio dei dati va effettuata in modo tale che la struttura del problema (pattern prevalente di comunicazione) sia "mappata" sulla rete di interconnessione fisica in modo → **1:1**
- Ogni deviazione dal mapping 1:1 comporta considerevoli perdite di tempo nello scambio dei dati (servono step multipli o passando attraverso l'unità di controllo, molto più lenta)

Le topologie di interconnessione delle architetture SIMD sono **regolari** (griglie, torus, ipercubi, mesh multidimensionali) o, nel caso di sistemi dedicati, vengono create in base alla struttura stessa del problema



Dominio dei dati



Rete di Interconnessione toroidale

Le architetture SIMD

La programmazione di una architettura SIMD è un modello data-parallel sincrono

(2) risulta semplificata per il fatto che vi è un solo flusso di controllo e non vi sono processi indipendenti che vengono eseguiti in modo asincrono

- Tutte le operazioni vengono effettuate in modalità LOCK-STEP (pertanto, non vi è bisogno di meccanismi di sincronizzazione espliciti e soggetti a rischio di errore, quali monitor, semafori, ecc.),

- Vi è scambio di dati tra i processori, ma:
  - di tipo collettivo e non solo tra coppie di nodi
  - è realizzato in modo molto efficiente (in un tempo di ordini di grandezza inferiore rispetto a comunicazioni in MIMD),

- Processori differenti non possono eseguire istruzioni differenti nello stesso ciclo di clock.

Problemi di efficienza del parallelismo SIMD

1. **Costrutti condizionali**

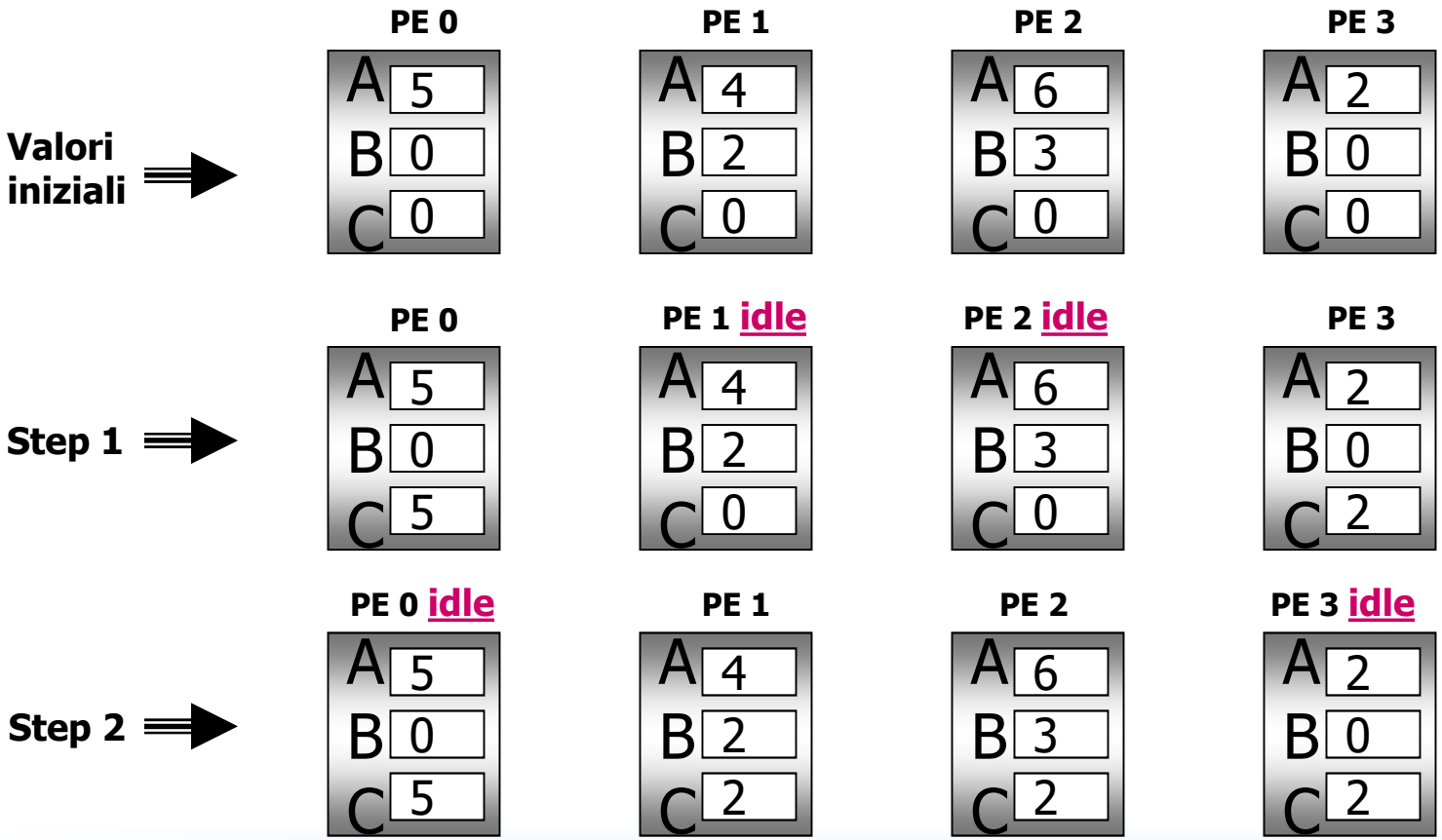
2. **Necessità di processori virtuali**

3. **Mapping non 1:1 del dominio dei dati → problemi nelle comunicazioni** (come nella soluzione di Poisson: topologie cartesiane)

Problemi di efficienza del parallelismo SIMD
costrutti condizionali

Nelle istruzioni condizionali, il codice per ogni condizione deve essere eseguito **sequenzialmente**.
Ciò introduce la necessità di mettere in idle i PE.

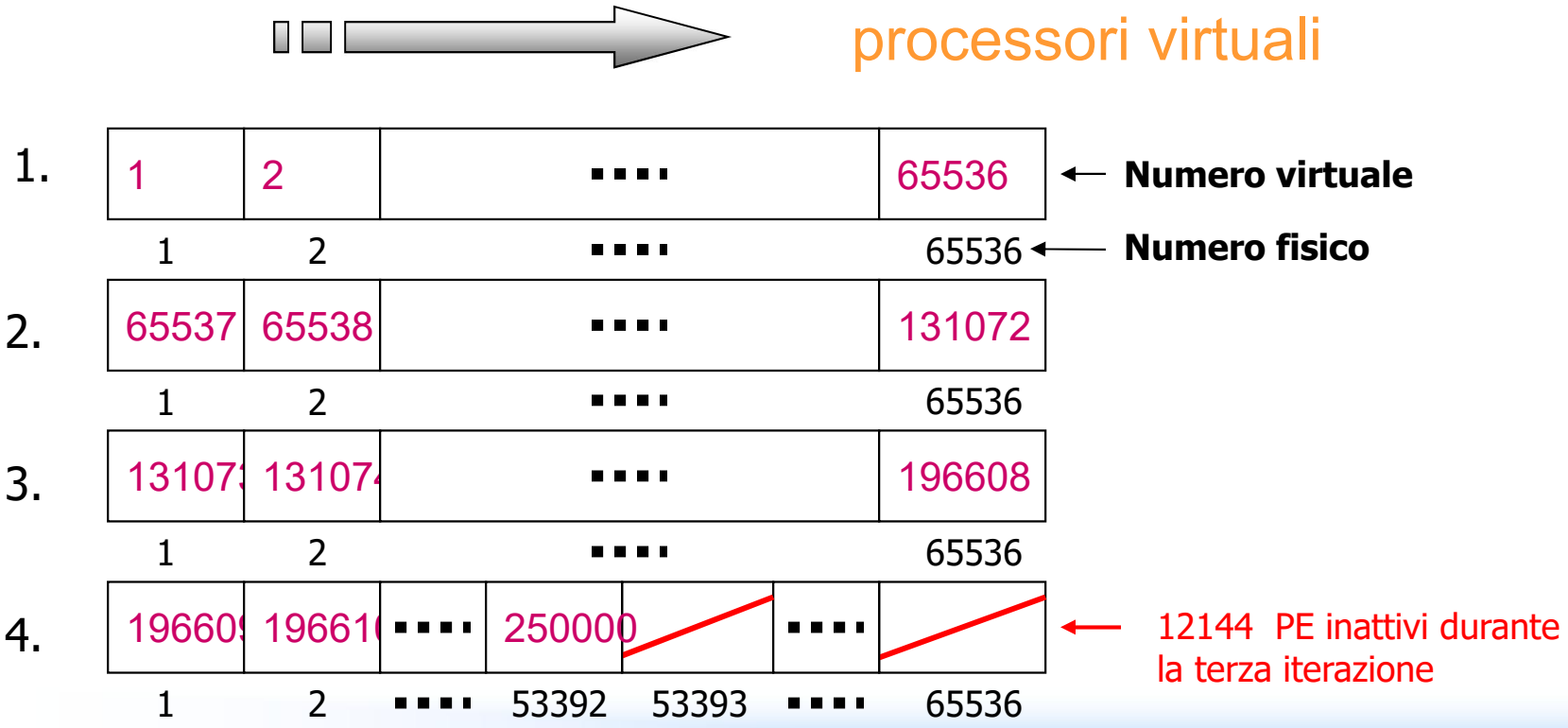
```
If (B==0)
      C=A;
else
      C=A/B;
```



Problemi di efficienza del parallelismo SIMD
processori virtuali

L' elementarità dei processori SIMD fa sì che ciascuno di essi esegua operazioni molto semplici ad ogni ciclo di clock: Il numero di PE potrebbe non essere sufficiente.

Esempio: gestire immagini con 500x500 pixel = 250.000 processori. Come faccio se ne ho "solo" 65536 ?
Se il programma richiede un PE per ogni pixel (es. smoothing) dovrò fare 4 iterazioni per ogni istruzione.



Problemi di efficienza del parallelismo SIMD
mapping del dominio dei dati

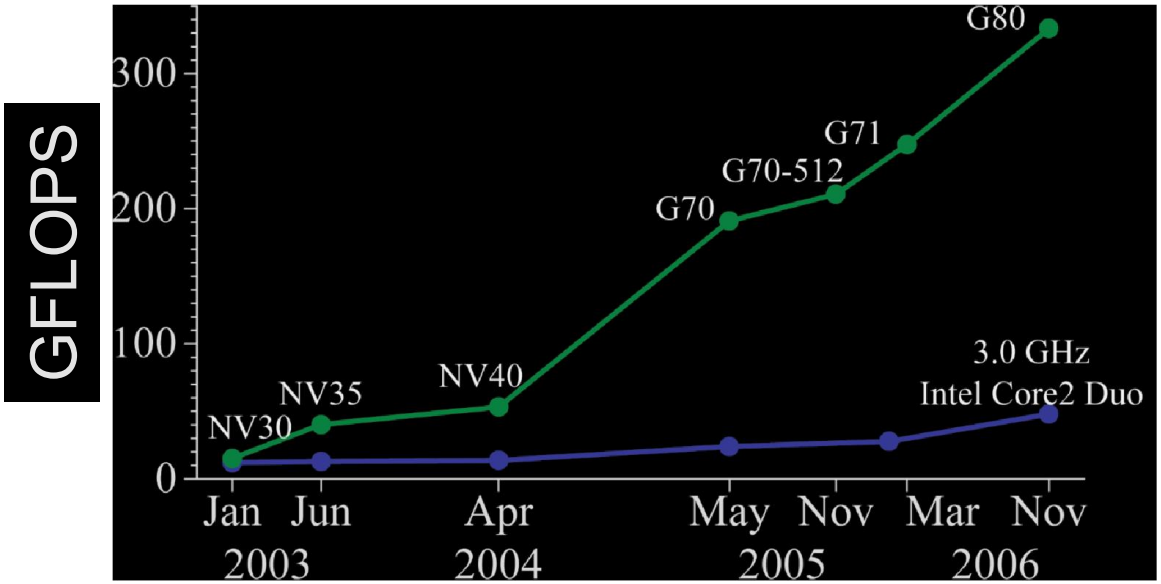
La scelta del mapping del dominio dei dati è legata al modello e alla **topologia di interconnessione** dei PE.

Le COMUNICAZIONI che rispettano la topologia fisica non creano conflitti e sono poco costose

- Anche le comunicazioni vengono eseguite in modalità *LOCK-STEP*
- Le comunicazioni non sono nodo-nodo, ma avvengono con uno scambio di dati (generalmente di dimensioni limitate) tra tutte le unità di elaborazione o tra un sottoinsieme di queste
- Le comunicazioni avvengono tra nodi vicini e quindi il costo è dell'ordine di un istruzione aritmetico/logica. Anche nei sistemi con comunicazioni più complesse, il loro costo è di diversi ordini di grandezza inferiore a quello che si ha nei sistemi MIMD.
- Ogni deviazione dalla regolarità causa considerevoli perdite di tempo, in quanto lo scambio dei dati deve avvenire:
  - o attraverso passi multipli
  - o (se disponibile) attraverso la più lenta struttura di interconnessione universale

GPGPU: nuove architetture SIMD (D.Kirk NVIDIA & uiuc.edu)

- A quiet revolution and potential build-up
  - Calculation: 367 GFLOPS vs. 32 GFLOPS
  - Memory Bandwidth: 86.4 GB/s vs. 8.4 GB/s
  - Until last year, programmed through graphics API



G80 = GeForce 8800 GTX
G71 = GeForce 7900 GTX
G70 = GeForce 7800 GTX
NV40 = GeForce 6800 Ultra
NV35 = GeForce FX 5950 Ultra
NV30 = GeForce FX 5800

  - GPU in every PC and workstation – massive volume and potential impact

Parallel Computing on a GPU

- NVIDIA GPU Computing Architecture
  - Via a separate HW interface
  - In laptops, desktops, workstations, servers

GeForce 8800

- 8-series GPUs deliver 50 to 200 GFLOPS on compiled parallel C applications

- GPU parallelism is doubling every year
- Programming model scales transparently

Tesla D870

- Programmable in C with CUDA tools
- Multithreaded SPMD model uses application data parallelism and thread parallelism

Tesla S870

Conclusione: le architetture SIMD

Per finire, elenchiamo i **pregi** delle macchine SIMD

1. **SIMD** richiede meno hardware dei **MIMD** perché vi è una sola **CU** e molti **PE**.
   D'altro canto, nei **MIMD** è possibile utilizzare processori general-purpose e quindi per le leggi dell'economia di scala i processori **MIMD** potrebbero essere allo stesso tempo più potenti e più economici dei **PE** delle architetture **SIMD**.

2. **SIMD** richiede meno memoria dei **MIMD** perché si deve memorizzare una sola copia del programma.
   MIMD memorizza il programma ed il sistema operativo in ogni nodo

3. **SIMD** è naturalmente predisposta ad una programmazione *data-parallel*. In particolare, con frequenti sincronizzazioni (*fine grain*)

4. **SIMD** richiede un tempo di STARTUP inferiore nelle comunicazioni con i vicini perché (essendovi un clock globale) la comunicazione è paragonabile ad un trasferimento register-to-register

Le architetture MIMD

Le architetture MIMD

Le caratteristiche che hanno decretato il successo delle architetture MIMD sono:

flessibilità:

- possono far girare un **singolo codice** sfruttando il proprio parallelismo,
- possono fare girare **più codici** contemporaneamente (multi-user),
- possono funzionare come una **combinazione** delle precedenti modalità (è quello che succede in realtà algli elaboratori dei grossi centri di calcolo);

economicità:

- sono spesso costruite partendo da numerosi processori **economici** già disponibili sul mercato,
- sono facilmente **espandibili**,
- sistemi operativi e sowtware di gestione sono spesso **free**;

semplicità:

- in generale non occorrono compilatori specifici ma bastano **librerie** di tipo Shared-Memory o Message-Passing,
- abbiamo visto come è semplice **programmare** queste macchine usando un linguaggio di programmazione comune (C, C++, Fortran, Fortran90).

Le architetture MIMD

Per sfruttare una architettura MIMD con n processori occorre avere n processi da distribuire: il programmatore dovrà esplicitamente parallelizzare l'algoritmo e creare gli stream di istruzioni multipli.

Il parallelismo che si sfrutta in questo modo è il cosiddetto tread-level parallelism.

Oggi le architetture MIMD rappresentano la grande maggioranza delle architetture parallele in uso.

Le architetture MIMD

Le architetture MIMD si suddividono, a loro volta, in:

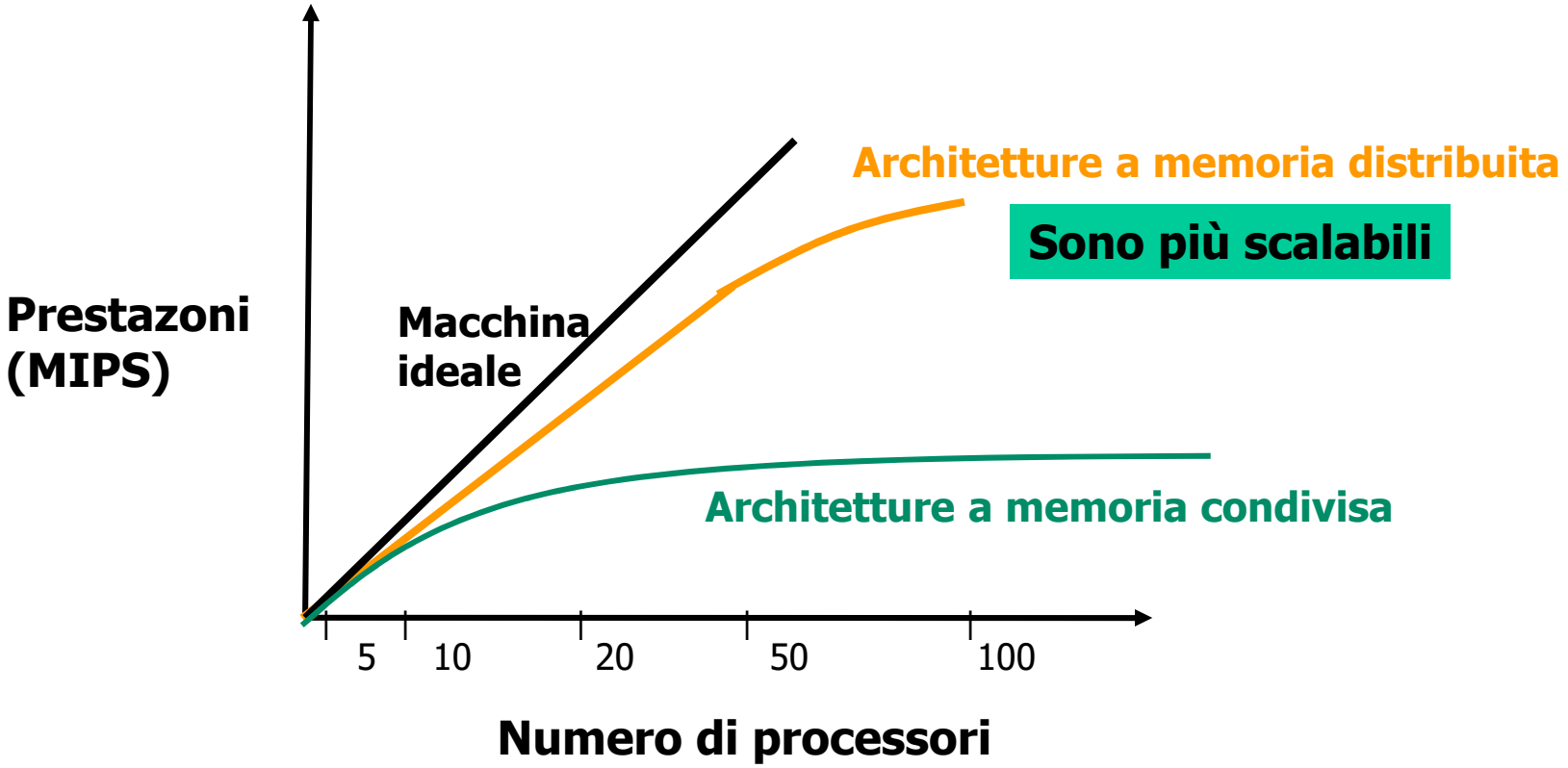
architetture a memoria condivisa

• tutti i processori condividono lo stesso spazio di indirizzamento della memoria (un dato indirizzo rappresenta, per tutti, la stessa locazione),
• non vuole necessariamente dire che la memoria è fisicamente una memoria singola centralizzata,
• di solito il numero di processori è limitato dalla complessità architetturale,
• sono rappresentate, ad esempio, dai server bi/quadri processore **SMP** (Symmetric MultiProcessors).

architetture a memoria distribuita

• un processore può indirizzare direttamente solo la propria memoria locale,
• di solito il collo di bottiglia per le prestazioni parallele è rappresentato dalla ridotta velocità delle comunicazioni fra i nodi,
• essendo l'accesso alla memoria indipendente da parte di ogni nodo non si hanno conflitti di memory access.
• sono rappresentate, ad esempio, dai **cluster** (multicomputer),

Scalabilità delle Architetture MIMD



Scalabilità delle Architetture MIMD

Per capire la difficoltà nella scalabilità di un codice all'aumentare del numero di processori ricordiamo la legge di Amdahl e facciamo un esempio.

DOMANDA: Supponendo di avere a disposizione 100 processori e di volere ottenere un incremento di prestazioni di un fattore 80, quale frazione del codice originale può restare seriale ?

RISPOSTA:Ricordo la legge di Amdahl $ts = te_{old} / te_{new} = 1 / [(1 – f) + (f / ps)]$

e sostituisco i dati del problema

$$80 = 1 / [(1 – f) + (f / 100)]$$

$$f = 0.9975$$

ottengo che la frazione del codice che può essere non parallela equivale allo

0.25% del totale !!

Vantaggi e svantaggi dei diversi meccanismi di comunicazio

SHARED-MEMORY COMMUNICATION

- Compatibility with the well-understood mechanisms in use in centralized multiprocessors, which all use shared-memory communication.

- Ease of programming when the communication patterns among processors are complex or vary dynamically during execution. Similar advantages simplify compiler design.

- The ability to develop applications using the familiar shared-memory model, focusing attention only on those accesses that are performance critical.

- Lower overhead for communication and better use of bandwidth when communicating small items. This arises from the implicit nature of communication and the use of memory mapping to implement protection in hardware, rather than through the I/O system.

- The ability to use hardware-controlled caching to reduce the frequency of remote communication by supporting automatic caching of all data, both shared and private.

Vantaggi e svantaggi dei diversi meccanismi di comunicazio

MESSAGE-PASSING COMMUNICATION

- The hardware can be simpler, especially by comparison with a scalable sharedmemory implementation that supports coherent caching of remote data.

- Communication is explicit, which means it is simpler to understand; in shared memory models, it can be difficult to know when communication is occurring and when it is not, as well as how costly the communication is.

- Explicit communication focuses programmer attention on this costly aspect of parallel computation, sometimes leading to improved structure in a multiprocessor program.

- Synchronization is naturally associated with sending messages, reducing the possibility for errors introduced by incorrect synchronization.

- It makes it easier to use sender-initiated communication, which may have some advantages in performance,

Vantaggi e svantaggi dei diversi meccanismi di comu

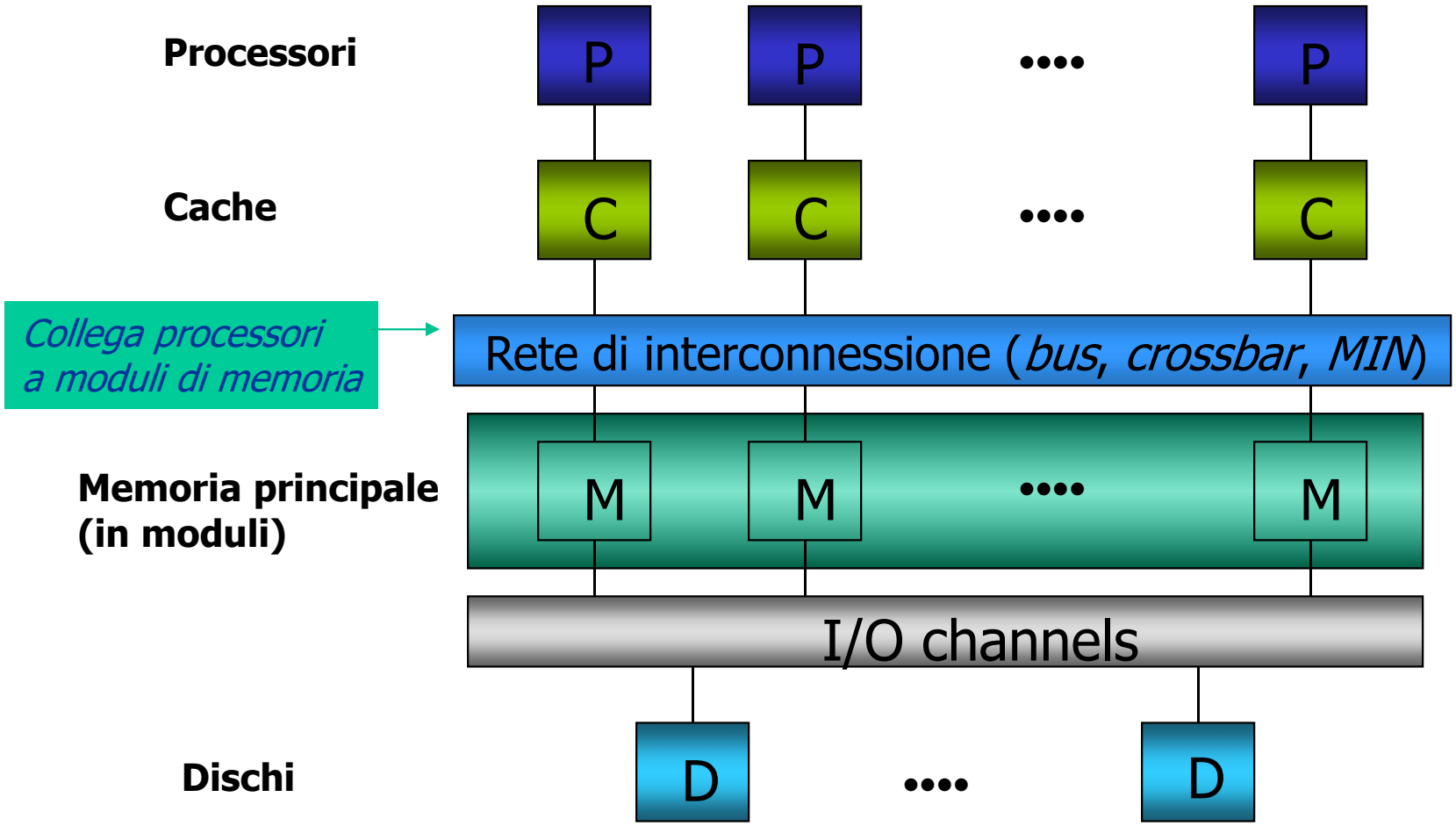
Il meccanismo di comunicazione non deve neccessariamente rispettare l'hardware del sistema.

meccanismo di comunicazione \ hardware	shared memory	distributed memory
shared-memory communication	directly writes on memory	difficult ! needs extra hw or sw
message-passing communication	easy ! needs extra hw or sw	sends messages to other nodes

Architetture MIMD a Memoria Condivisa (shared memory)

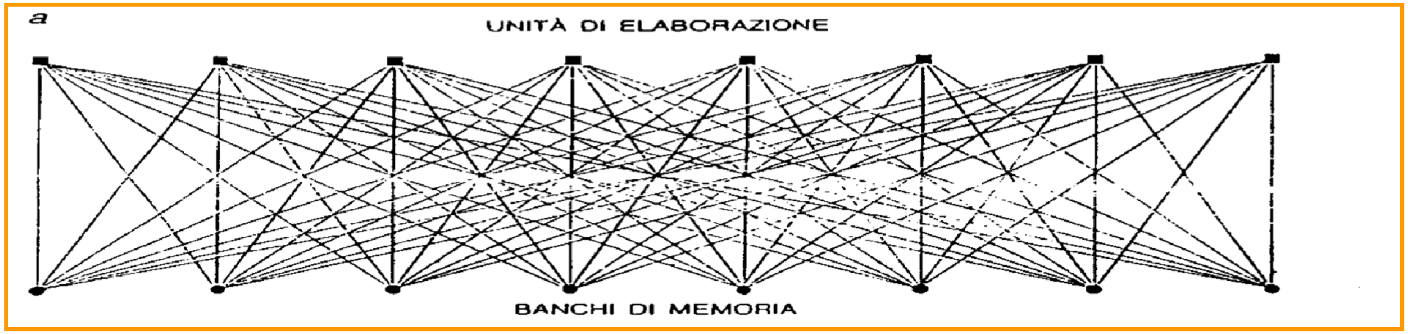
Multiprocessor

Modello Architetture MIMD shared memory

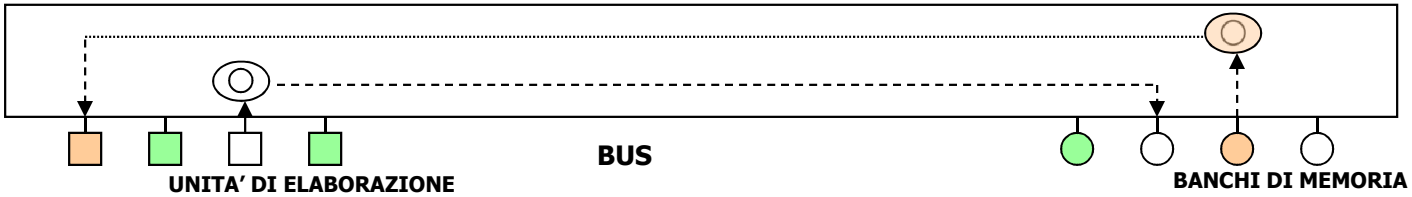


MIMD shared memory: rete di interconnessione

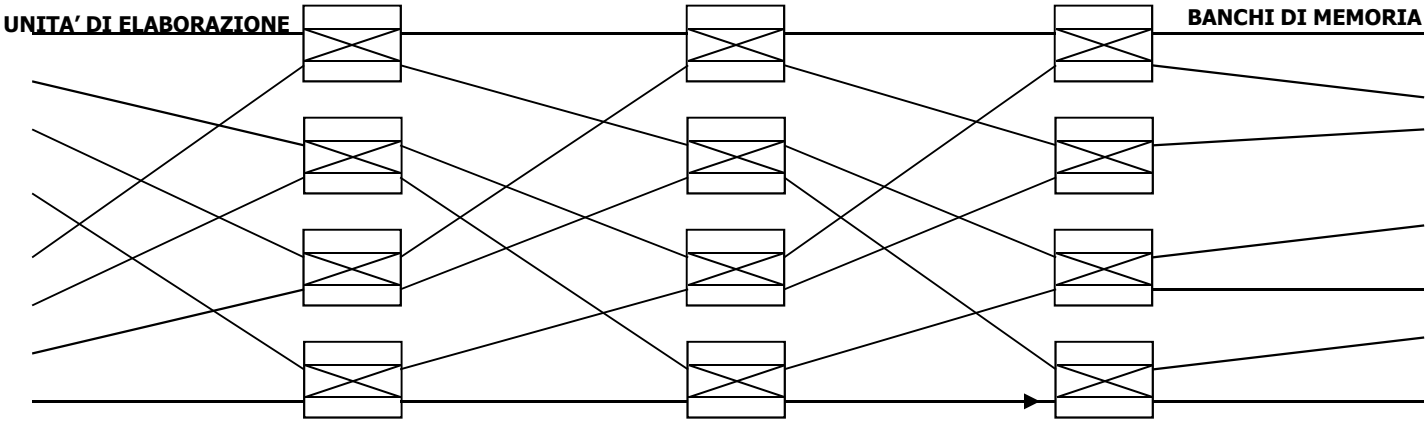
A



B

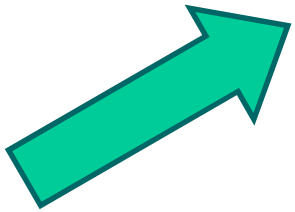


C

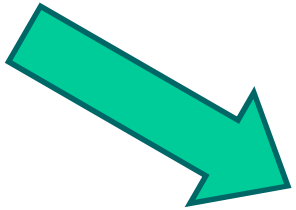


Crossbar Switch ("Rete Omega")

MIMD shared memory:
due modelli
di accesso alla memoria

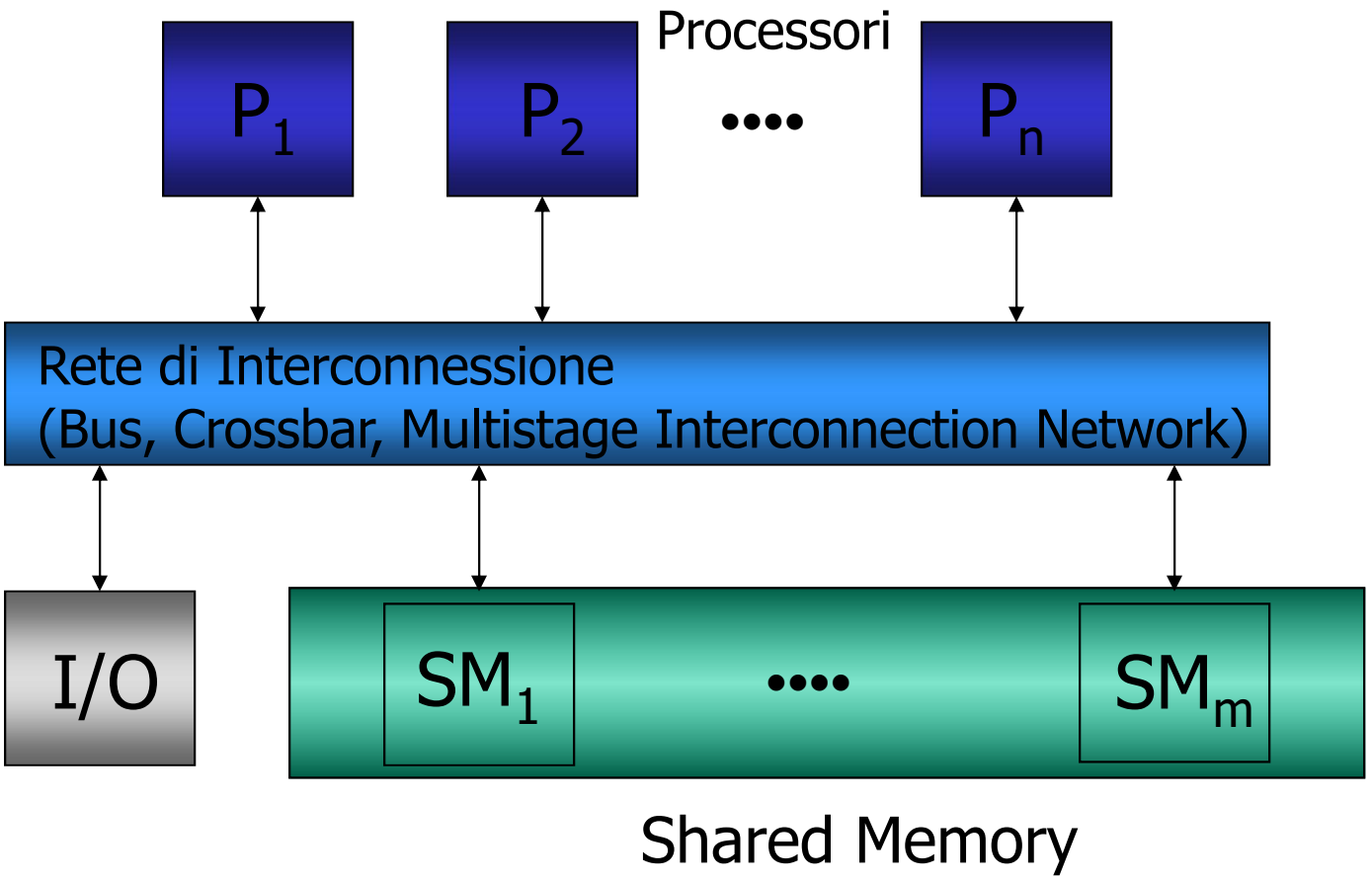


Uniform Memory Acess
(UMA)

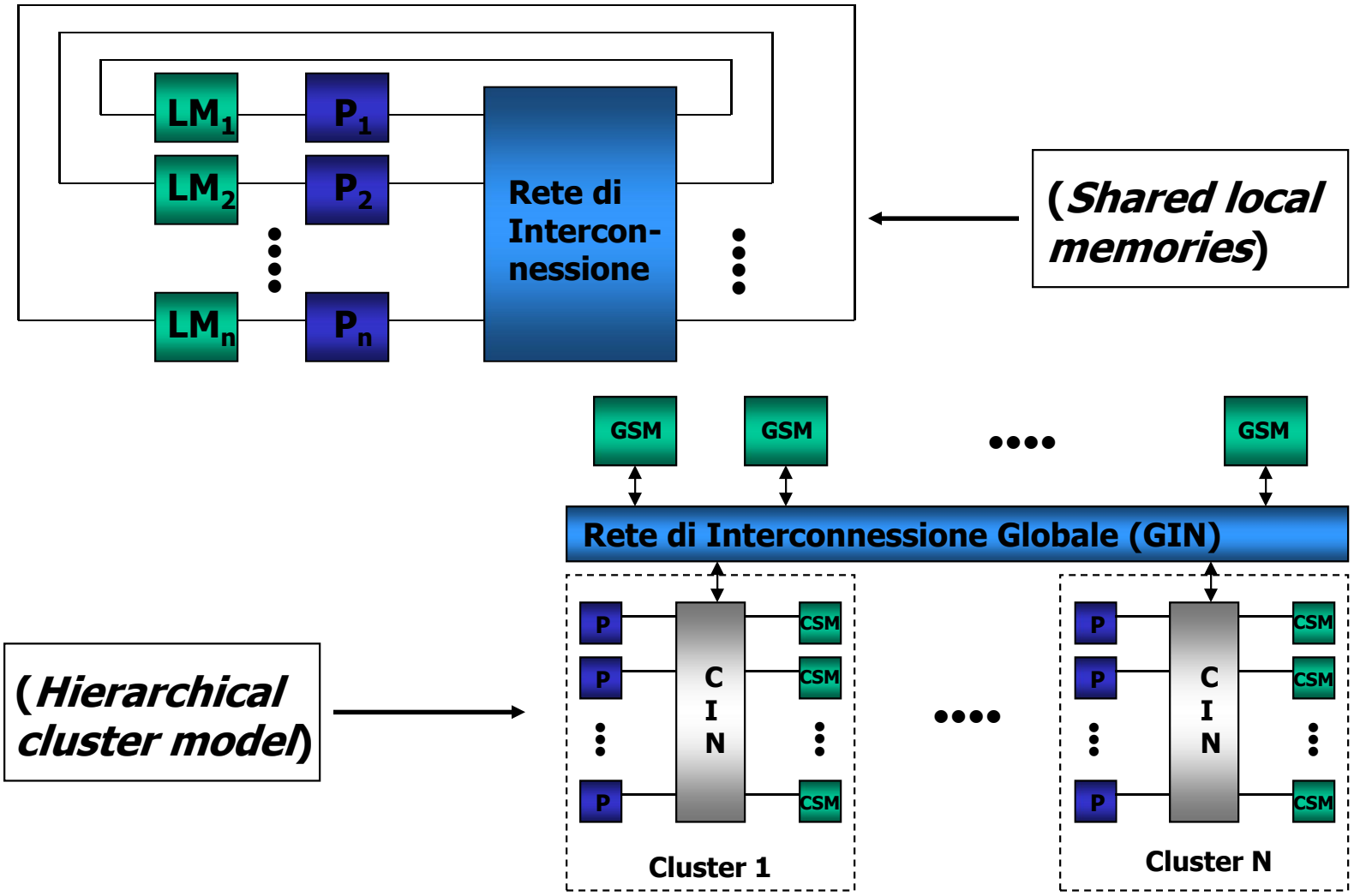


Non-Uniform Memory Acess
(NUMA)

MIMD shared memory: modello UMA



MIMD shared memory: modello NUMA



Esempio: Intel Core 2 Duo quad-core (Kentsfield)

- Commercializzato nel novembre 2006
- Singolo socket
- Due chip Core 2 Duo connessi da un FSB da 1066 Mhz
- Architettura UMA

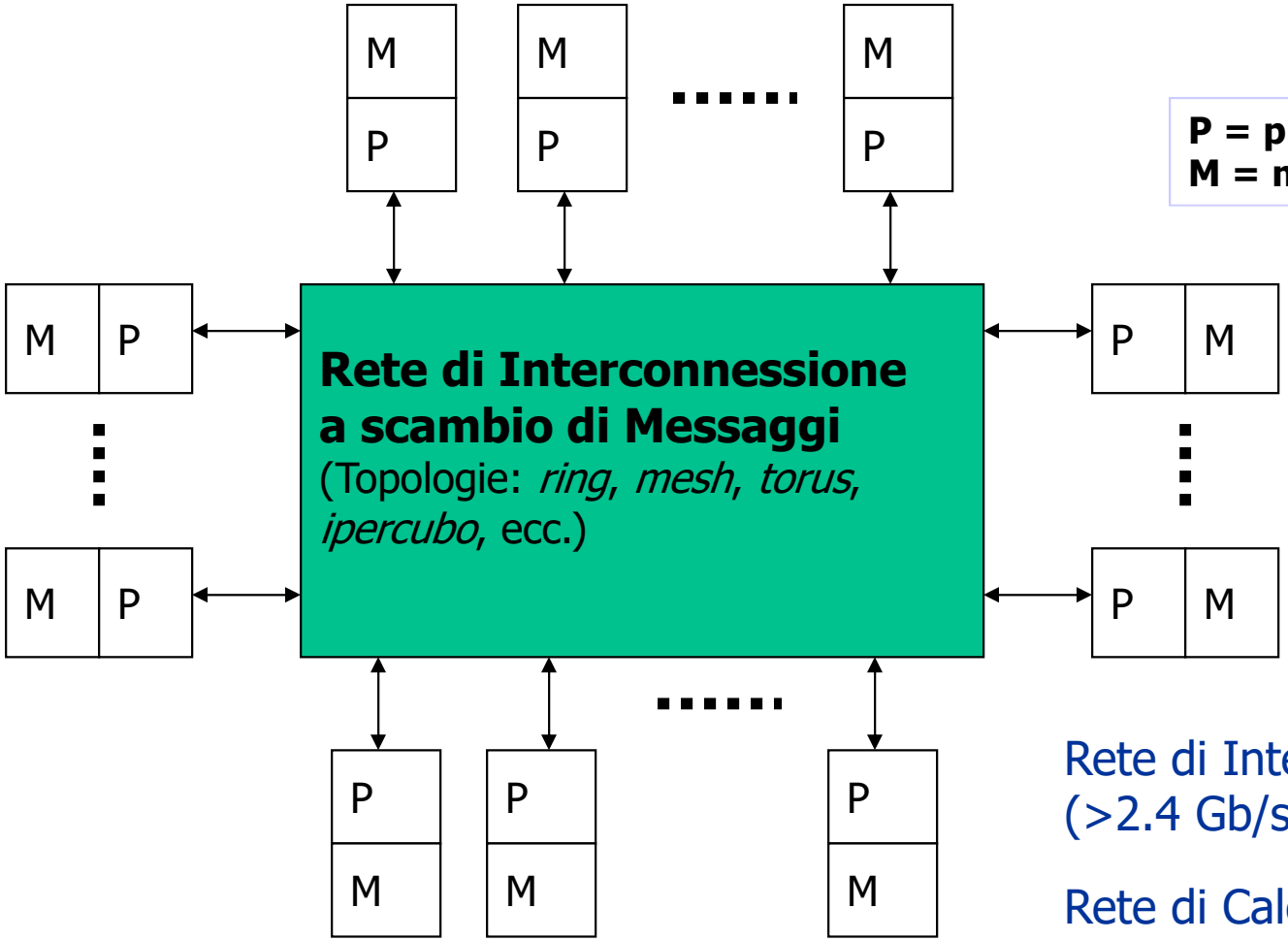
Esempio: multi processore AMD Opteron

- Commercializzato dal 2003 (K10 dal 2007 ma rimandato)
- Socket multiplo. Comunicazione HyperTransport.
- Barcelona: Due chip Opteron-like dual-core sullo stesso package
- Architettura NUMA

Architetture MIMD a Memoria Distribuita

Multicomputer

Modello di architettura MIMD a memoria distribuita



P = processore
M = memoria locale

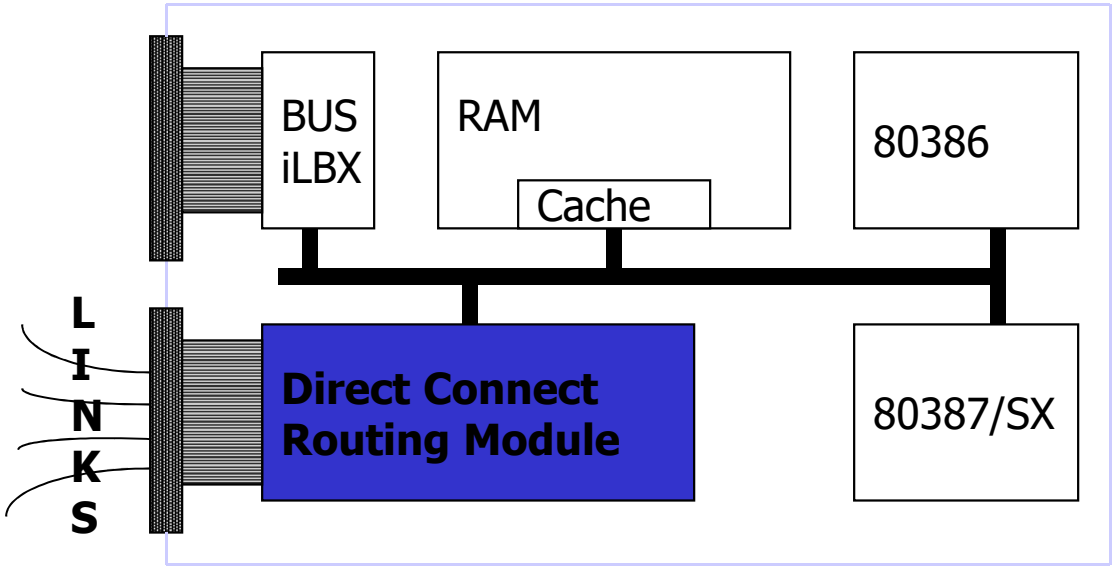
Rete di Interconnessione (>2.4 Gb/s)

Rete di Calcolatori LAN (1.2 Gb/s)

Modello di architettura MIMD a memoria distribuita

- Composte da un numero variabile di elementi di elaborazione (detti anche *nodi*, in quanto composti da più componenti)
- I nodi sono tutti connessi mediante una rete di interconessione diretta
- Ciascun nodo contiene uno o più processori, una memoria locale, una cache, un canale di comunicazione con altri nodi e, nelle architetture più moderne, un canale di I/O (prevalentemente verso un disco)

Esempio di nodo di II generazione: Intel iPSC/2 (1987)



Caratteristiche dell'architettura MIMD a memoria distribuita

- Potenzialmente molto più scalabili
- Non vi è memoria condivisa tra i diversi nodi
- Ciascun nodo esegue indipendentemente insiemi multipli di istruzioni utilizzando differenti insiemi di dati, memorizzati su spazi differenti
- I nodi scambiano informazioni attraverso la rete di interconnessione utilizzando qualche politica *message-passing*
- Tre sono le reti di interconnessione (commercialmente) più diffuse: **Ipercubi**, **Mesh**, **Torus**.

Ipercubi

- Intel iPSC/1, **iPSC/2**, iPSC/860
- Ncube, Ncube-2

Mesh 2D

- Intel Delta, Intel Paragon
- Basati su Transputer: Meiko Computing Surface, Parsytec

Torus 3D

- Cray T3D