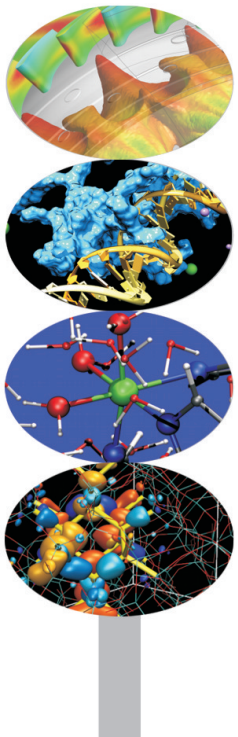
CINECA SCAI
SuperComputing Applications and Innovation

SAPIENZA
UNIVERSITÀ DI ROMA

Programmazione Avanzata

Vittorio Ruggiero

(v.ruggiero@cineca.it)



Roma, 10 Marzo 2014

CINECA

AGENDA

La mia parte di corso

- Architetture
- La cache ed il sistema di memoria
- Pipeline
- Profiling seriale
- Profiling parallelo
- Debugging seriale
- Debugging parallelo
- Esercitazioni

Outline

Introduzione

Architetture

Quanto possono variare le prestazioni?

Prodotto matrice-matrice (misurato in secondi)

Precisione	singola	doppia
Loop incorretto	7500	7300
senza ottimizzazione	206	246
con ottimizzazione (-fast)	84	181
codice ottimizzato	23	44
Libreria ACML (seriale)	6.7	13.2
Libreria ACML (2 threads)	3.3	6.7
Libreria ACML (4 threads)	1.7	3.5
Libreria ACML (8 threads)	0.9	1.8
Pgi accelerator	3	5
CUBLAS	1.6	3.2

Glossario

- Processore(fisico)
- Core
- Nodo di calcolo
- Thread
- Processore logico
- Processo

Processore/ CPU

- Central Processing Unit (CPU),è la parte di un computer che esegue i programmi.
- Un sistema di calcolo ad alte prestazioni (HPC) contiene molte CPU.
- Una CPU contiene uno o più core

Core

- È la parte del processore che veramente esegue i programmi. Può eseguire un solo comando alla volta.
- Alcuni processori possono apparentemente eseguire più comandi nello stesso momento (HypherThreading e Simultaneous MultiThreading).

Nodo di calcolo

- Un computer esegue un'immagine del sistema operativo.
- I programmi eseguiti su macchine a memoria condivisa (shared-memory) possono usare al massimo un solo nodo.
- Un Supercomputer è costituito da molti nodi interconnessi con una rete molto veloce (ad esempio InfiniBand).

Thread

- È la più piccola unità di lavoro che può essere gestita dal sistema operativo.

Processore logico

- Il sistema operativo vede un processore logico per ogni thread che può essere assegnato ad un core.
- Su un sistema dual-core dove ogni core può gestire 2 thread, il sistema operativo vedrà 4 processori logici.

Processo

- Un processo è un'istanza di un programma in esecuzione. Esso contiene il codice eseguibile e tutto quello che è necessario per definirne lo stato .
- Un processo può essere costituito da più thread di esecuzione che lavorano in contemporanea.

Prodotto matrice-matrice

- Fortran o C
- Prodotto riga per colonna $C_{i,j} = A_{i,k} B_{k,j}$
- Temporizzazione:
  - Fortran: date_and_time (> 0.001")
  - C: clock (>0.05")
- Per matrici quadrate di dimensione n
  - Memoria richiesta (doppia precisione) $\approx (3 * n * n) * 8$
  - Operazioni totali $\approx 2 * n * n * n$
    - Bisogna accedere a n elementi delle due matrici origine per ogni elemento della matrice destinazione
    - n prodotti ed n somme per ogni elemento della matrice destinazione
  - Flops totali = $2 * n^3 / tempo$
- Verificare sempre i risultati :-)

Sporchiamoci le mani

- Completiamo il loop principale del codice e verifichiamo:
  - Che prestazioni sono state ottenute?
  - C'è differenza tra Fortran e C?
  - Cambiano le prestazioni cambiando compilatore?
  - E le opzioni di compilazione?
  - Cambiano le prestazioni se cambio l'ordine dei loop?
  - Posso riscrivere il loop in maniera efficiente?

Misura delle prestazioni

- Stimare il numero di operazioni necessarie per l'esecuzione N_{Flop}
  - 1 FLOP equivale ad un'operazione di somma o moltiplicazione floating-point
  - Operazioni più complicate(divisione, radice quadrata, funzioni trigonometriche) vengono eseguite in modo più complesso e più lento
- Stimare il tempo necessario per l'esecuzione T_{es}
- Le prestazioni sono misurate contando il numero di operazioni floating-point eseguite per unità di tempo:

  $Perf = \frac{N_{Flop}}{T_{es}}$
  - l'unità di misura minima è 1 Floating-pointing Operation per secondo (FLOPS)
  - Normalmente si usano i multipli:
    - 1 MFLOPS= 10^6 FLOPS
    - 1 GFLOPS= 10^9 FLOPS
    - 1 TFLOPS= 10^{12} FLOPS

Makefile

- Per compilare
  - make
- Per pulire
  - make clean
- Per cambiare compilatore o opzioni
  - make "FC=ifort"
  - make "CC=icc"
  - make "OPT=fast"
- Per compilare in singola precisione
  - make "OPT=-DSINGLEPRECISION"
- Di default compila in doppia precisione

cat /proc/cpuinfo


```
processor       : 0
vendor_id       : GenuineIntel
cpu family      : 6
model           : 37
model name      : Intel(R) Core(TM) i3 CPU       M 330  @ 2.13GHz
stepping        : 2
cpu MHz         : 933.000
cache size      : 3072 KB
physical id     : 0
siblings        : 4
core id         : 0
cpu cores       : 2
apicid          : 0
initial apicid  : 0
fpu             : yes
fpu_exception   : yes
cpuid level     : 11
wp              : yes
wp              : yes
flags           : fpu vme de pse tsc msr pae mce cx8
bogomips        : 4256.27
clflush size    : 64
cache_alignment : 64
address sizes   : 36 bits physical, 48 bits virtual
...
```

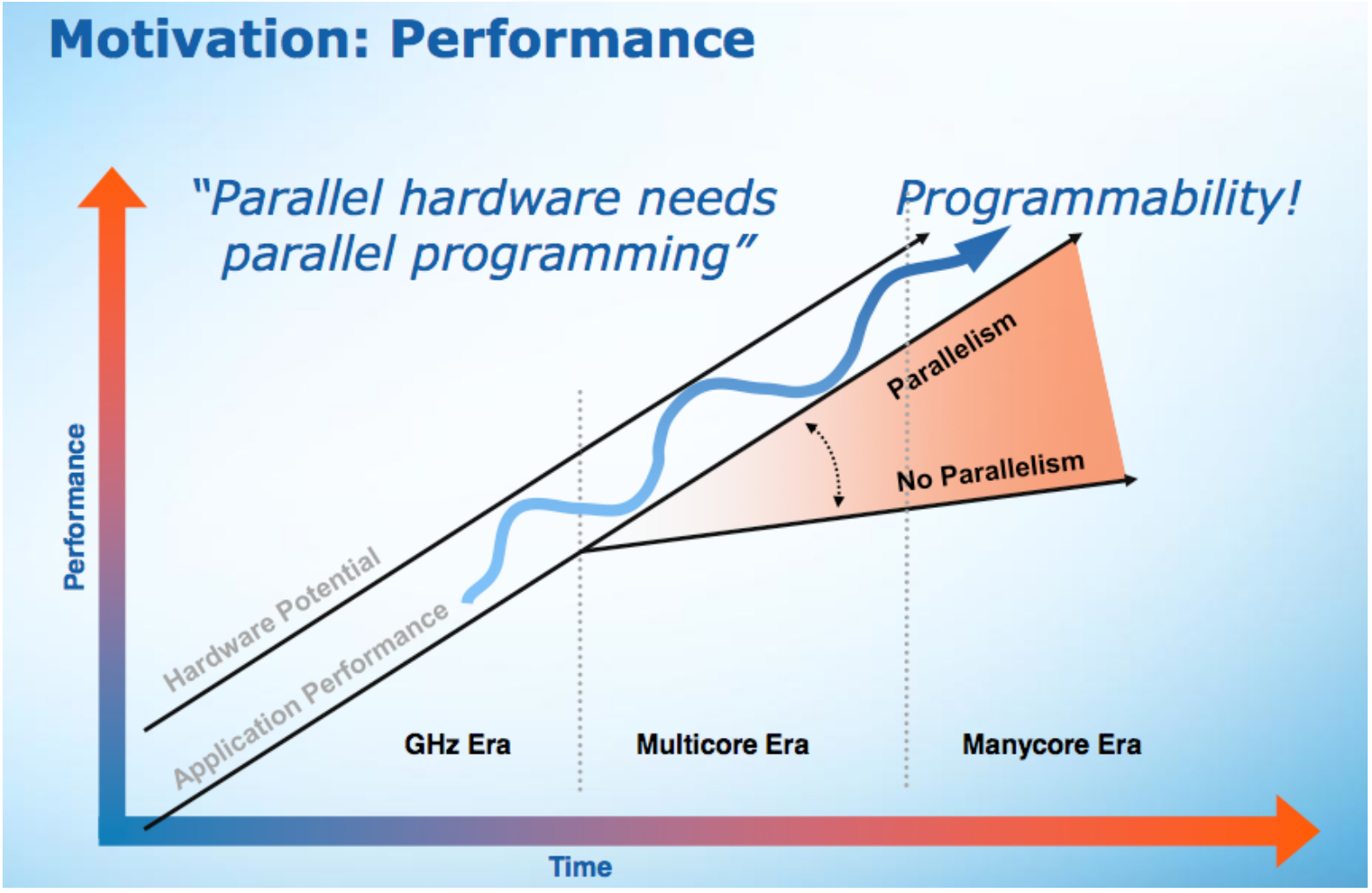
lscpu


```
Architecture:          x86_64
CPU op-mode(s):        32-bit, 64-bit
Byte Order:            Little Endian
CPU(s):                4
On-line CPU(s) list:   0-3
Thread(s) per core:    2
Core(s) per socket:    2
CPU socket(s):         1
NUMA node(s):          1
Vendor ID:             GenuineIntel
CPU family:            6
Model:                 37
Stepping:              2
CPU MHz:               933.000
BogoMIPS:              4255.78
Virtualization:        VT-x
L1d cache:             32K
L1i cache:             32K
L2 cache:              256K
L3 cache:              3072K
NUMA node0 CPU(s):     0-3
```

Prodotto matrice-matrice

Qualche considerazione sui risultati ottenuti?

Qual è il margine di manovra?



Software development

Problem

Solution method

Algorithms

Programming

Source code

```
#include <stdio.h>
#define SIZE 1000
main(int argc, char** argv) {
  int A[SIZE], B[SIZE], C[SIZE];
  int i;

  for (i=0; i<SIZE; i++) {
    B[i] = i;
    C[i] = SIZE-i;
  }

  /* Add B and C */
  for (i=0; i<SIZE; i++) {
    A[i] = B[i]*C[i];
  }
```

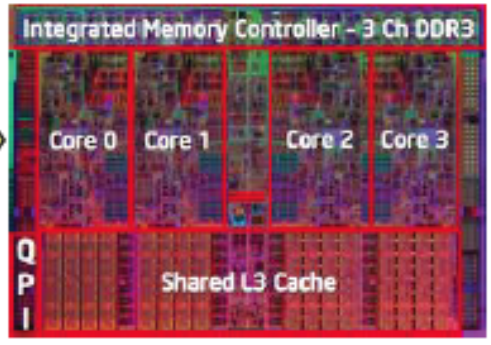
Compiling

Compiled and optimized code

```
.globl main
        .type    main,@function
main:
        pushl %ebp
        movl %esp,%ebp
        subl $12004,%esp
        pushl %edi
        pushl %esi
        pushl %ebx
        nop
        movl $0,-12004(%ebp)
.L2:
        cmpl $999,-12004(%ebp)
        jle .L6
        jmp .L3
        .align 4
.L6:
        movl -12004(%ebp),%eax
        movl %eax,%edx
        leal 0(,%edx,4),%eax
        leal -4000(%ebp),%edx
        movl -12004(%ebp),%ecx
        movl %ecx,%ebx
        leal 0(,%ebx,4),%ecx
        leal -8000(%ebp),%ebx
        movl -12004(%ebp),%esi
        movl %esi,%edi
        leal 0(,%edi,4),%esi
        leal -12000(%ebp),%edi
        movl (%ecx,%ebx),%ecx
        imull (%esi,%edi),%ecx
        movl %ecx,(%eax,%edx)
.L4:
        incl -12004(%ebp)
        jmp .L2
        .align 4
.L3:
        leal -12016(%ebp),%esp
        popl %ebx

.Lfe1:
        .size    main,.Lfe1-main
        .ident   "GCC: (GNU) 2.8.1"
```

Execution



Result

Output

```
Hierarchical code: Plummer model

 nbody      dtime       eps       theta     usequad     dtout       tstop
  1024    0.03125    0.0250        1.00       false    0.2500      2.0000

 tnow        Te+U        T/U       nttot       nbavg      ncavg    cputime
0.000     -0.2627     -0.4943      203185          84        114       0.00

           cm pos      0.0000     -0.0000      0.0000
           cm vel     -0.0000      0.0000      0.0000
           an vec      0.0057      0.0156     -0.0222

 tnow        Te+U        T/U       nttot       nbavg      ncavg    cputime
0.031     -0.2627     -0.4940      202260          81        116       0.01

           cm pos      0.0000     -0.0000      0.0000
           cm vel      0.0000     -0.0000      0.0000
           an vec      0.0057      0.0156     -0.0222

Time is:     9.5 seconds
```

Algoritmi

- Fondamentale è la scelta dell'algoritmo
  - algoritmo efficiente $\rightarrow$ buone prestazioni
  - algoritmo inefficiente $\rightarrow$ cattive prestazioni
- Se l'algortimo non è efficiente non ha senso tutto il discorso sulle prestazioni
- Regola d'oro
  - Curare quanto possibile la scelta dell'algoritmo prima della codifica, altrimenti c'è la possibilità di dover riscrivere!!!

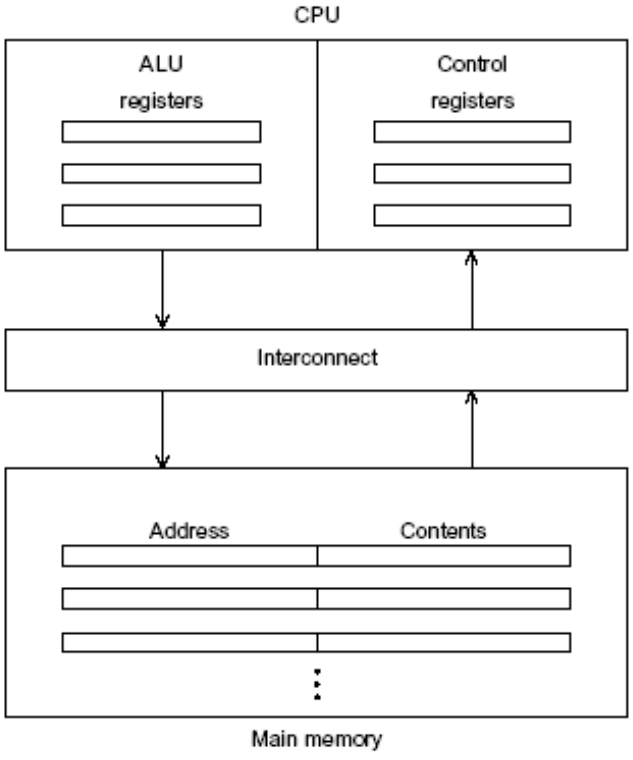
Outline

Introduzione

Architetture

Architettura di Von Neuman

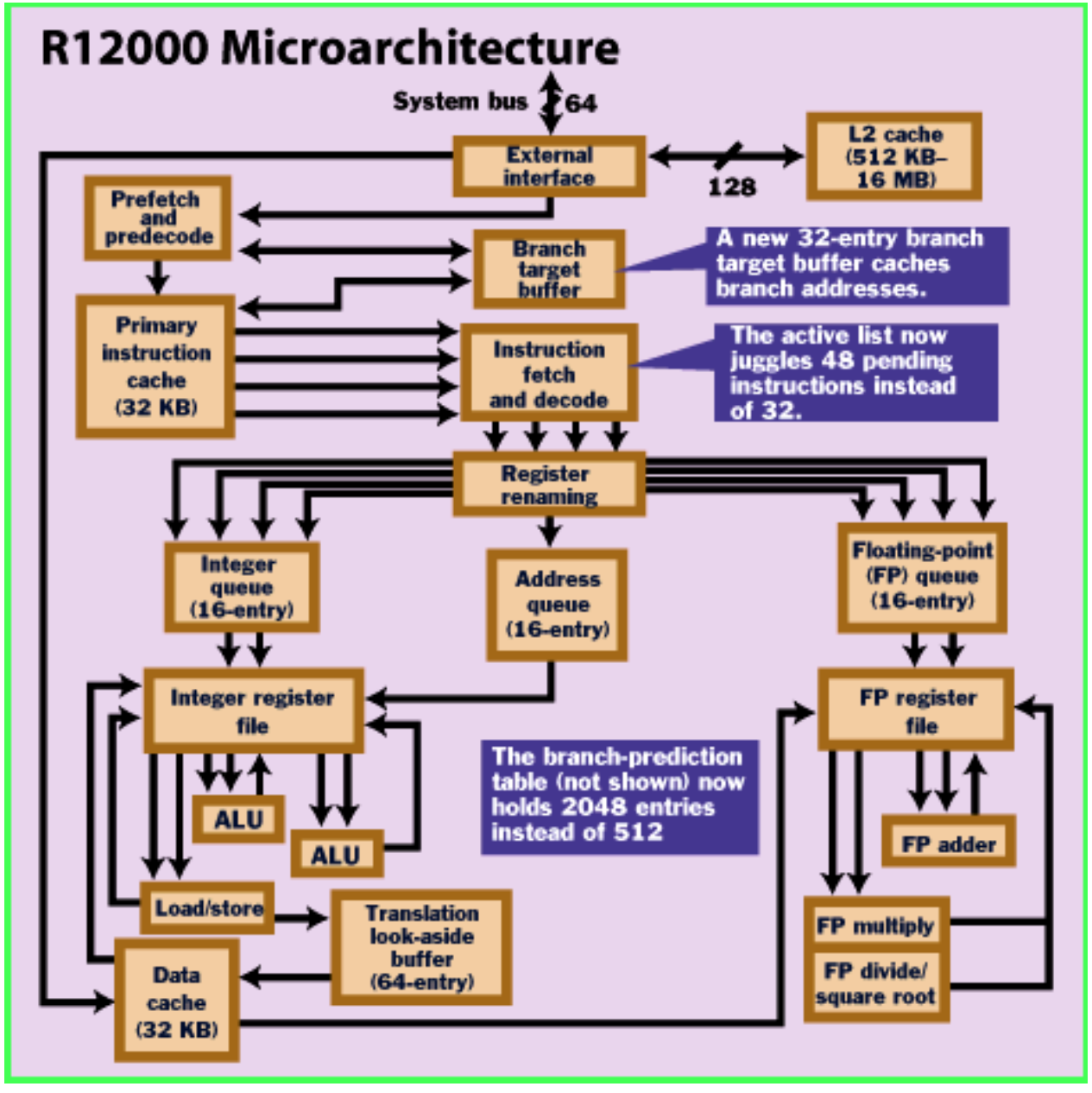
- CPU
  - Unità Logica Aritmetica (esegue le istruzioni)
  - Unità di controllo
  - Registri (memoria veloce)
- Interconnessione CPU RAM (Bus)
- Random Access Memory (RAM)
  - Indirizzo per accedere alla locazione di memoria
  - Contenuto della locazione (istruzione, dato)



Architettura di Von Neuman

- I dati sono trasferiti dalla memoria alla CPU (fetch o read)
- I dati sono trasferiti dalla CPU alla memoria (written to memory o stored)
- La separazione di CPU e memoria è conosciuta come la «von Neumann bottleneck» perché è il bus di interconnessione che determina a che velocità si può accedere ai dati e alle istruzioni.
- Le moderne CPU sono in grado di eseguire istruzioni almeno cento volte più velocemente rispetto al tempo richiesto per recuperare (fetch) i dati nella RAM

Implementazione reale



Evoluzione dei sistemi

Il «von Neumann bottleneck» è stato affrontato seguendo tre percorsi paralleli:

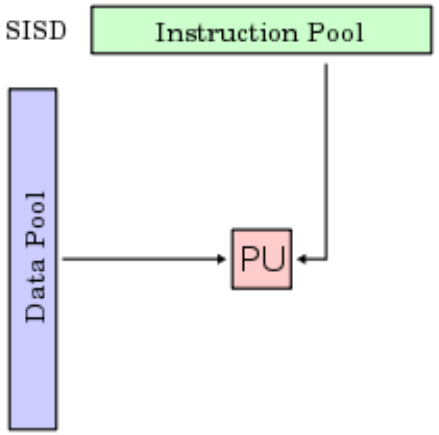
- Caching
  - Memorie molto veloci presenti sul chip del processore.
  - Esistono cache di primo, secondo e terzo livello.
- Virtual memory
  - Sistema sviluppato per fare in modo che la RAM funzioni come una cache per lo storage di grosse moli di dati.
- Instruction level parallelism
  - Tecnica utilizzata per avere più unità funzionali nella CPU che eseguono istruzioni in parallelo (pipelining ,multiple issue)

Classificazione di Flynn

- Racchiude le architetture dei computer in base alla molteplicità dell'hardware usato per manipolare lo stream di istruzioni e dati
  - SISD: single instruction, single data. Corrisponde alla classica achitettura di von Neumann, sistema scalare monoprocessore.
  - SIMD: single instruction, multiple data. Architetture vettoriali, processori vettoriali, GPU.
  - MISD: multiple instruction, single data. Non esistono soluzioni hardware che sfruttino questa architettura.
  - MIMD: multiple instruction, multiple data. Più processori/cores interpretano istruzioni diverse e operano su dati diversi.
- Le moderne soluzioni di calcolo sono date da una combinazione delle categorie previste da Flynn.

SISD

- Il classico sistema di von Neumann. Calcolatori con una sola unità esecutiva ed una sola memoria. Il singolo processore obbedisce ad un singolo flusso di istruzioni (programma sequenziale) ed esegue queste istruzioni ogni volta su un singolo flusso di dati.
- I limiti di prestazione di questa architettura vengono ovviati aumentando il bus dati ed i livelli di memoria ed introducendo un parallelismo attraverso le tecniche di pipelining e multiple issue.



SIMD

- La stessa istruzione viene eseguita in parallelo su dati differenti.
  - modello computazionale generalmente sincrono
- Processori vettoriali
  - molte ALU
  - registri vettoriali
  - Unità Load/Store vettoriali
  - Istruzioni vettoriali
  - Memoria interleaved
  - OpenMP, MPI
- Graphical Processing Unit
  - GPU completamente programmabili
  - molte ALU
  - molte unità Load/Store
  - molte SFU
  - migliaia di threads lavorano in parallelo
  - CUDA

MIMD

- Molteplici streams di istruzioni eseguiti simultaneamente su molteplici streams di dati
  - modello computazionale asincrono
- Cluster
  - molti nodi di calcolo (centinaia/migliaia)
  - più processori multicore per nodo
  - RAM condivisa sul nodo
  - RAM distribuita fra i nodi
  - livelli di memoria gerarchici
  - OpenMP, MPI, MPI+OpenMP

MIMD:Fermi Blue Gene/Q

Model:

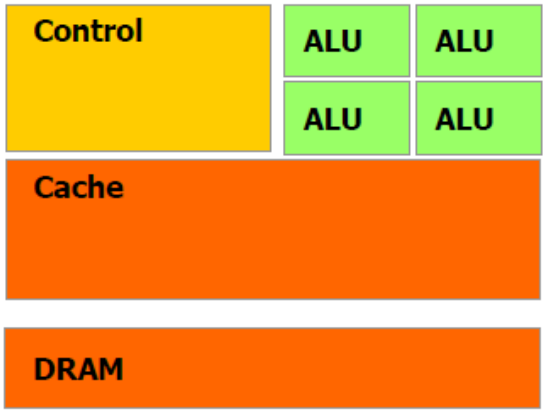
IBM-BlueGene /Q
Architecture: 10 BGQ Frame with 2 MidPlanes each
Front-end Nodes OS: Red-Hat EL 6.2
Compute Node Kernel: lightweight Linux-like kernel
Processor Type: IBM PowerA2, 16 cores, 1.6 GHz
Computing Nodes: 10.240
Computing Cores: 163.840
RAM: 16GB / node
Internal Network: Network interface
with 11 links ->5D Torus
Disk Space: more than 2PB of scratch space
Peak Performance: 2.1 PFlop/s

Cluster CPU GPU

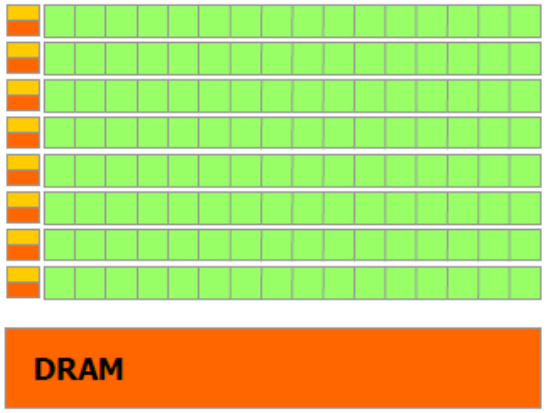
- Soluzioni ibride CPU multi-core + GPU many-core:
  - ogni nodo di calcolo è dotato di processori multicore e schede grafiche con processori dedicati per il GPU computing
  - notevole potenza di calcolo teorica sul singolo nodo
  - ulteriore strato di memoria dato dalla memoria delle GPU
  - OpenMP, MPI, CUDA e soluzioni ibride MPI+OpenMP, MPI+CUDA, OpenMP+CUDA, OpenMP+MPI+CUDA

CPU vs GPU

- Le CPU sono processori general purpose in grado di risolvere qualsiasi algoritmo
  - threads in grado di gestire qualsiasi operazione ma pesanti, al massimo 1 thread per core computazionale.
- Le GPU sono processori specializzati per problemi che possono essere classificati come «intense data-parallel computations»
  - controllo di flusso molto semplice (control unit ridotta)
  - molti threads leggeri che lavorano in parallelo



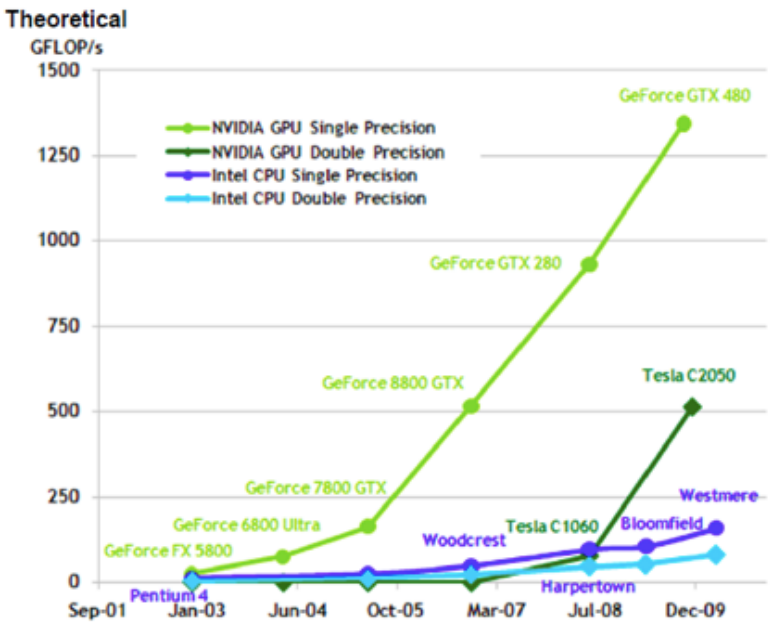
CPU



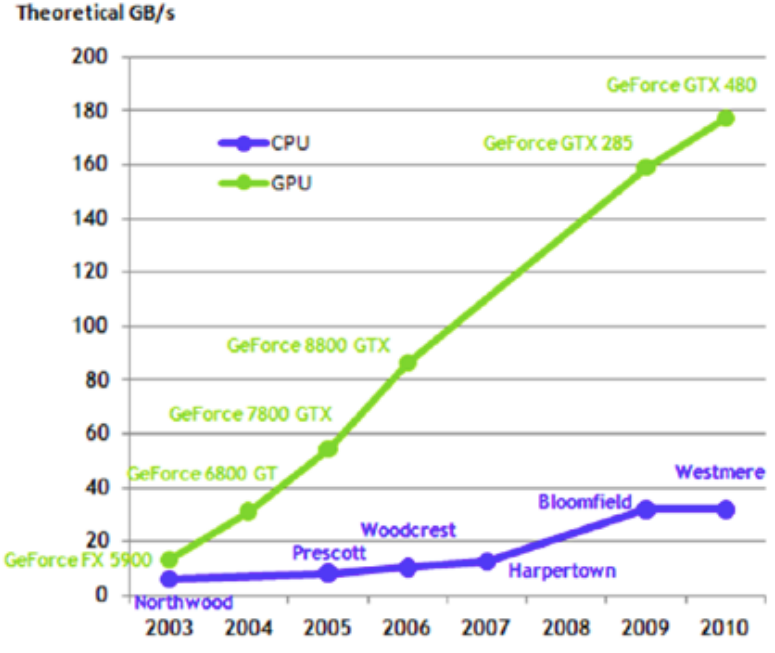
GPU

Multicore vs Manycore

- Una nuova direzione di sviluppo per l'architettura dei microprocessori:
  - incrementare la potenza di calcolo complessiva tramite l'aumento del numero di unità di elaborazione piuttosto che della loro potenza
  - la potenza di calcolo e la larghezza di banda delle GPU ha sorpassato quella delle CPU di un fattore 10.



Numero di operazioni in virgola mobile al secondo per la CPU e la GPU



Larghezza di banda della memoria

INTEL MIC

- Coprocessore Intel Xeon Phy
  - Basato su architettura Intel Many Integrated Core (MIC)
  - 60 core/1,053 GHz/240 thread
  - 8 GB di memoria e larghezza di banda di 320 GB/s
  - 1 TFLOPS di prestazioni di picco a doppia precisione
  - Istruzioni SIMD a 512 bit
  - Approcci tradizionali come MPI, OpenMP

Bandwidth

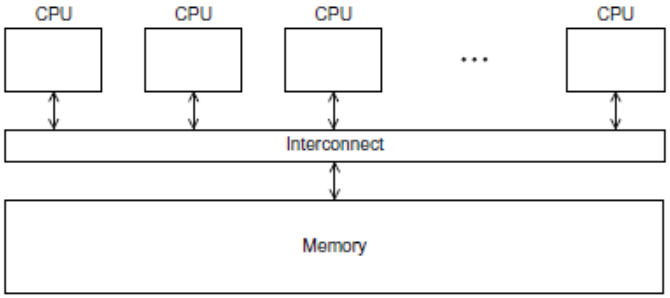
- La velocità con la quale è possibile trasferire i dati tra la memoria e il processore
- Si misura in numero di bytes che si possono traferire al secondo (Mb/s, Gb/s, etc..)
- A = B * C
  - leggere dalla memoria il dato B
  - leggere dalla memoria il dato C
  - calcolare il prodotto B * C
  - salvare il risultato in memoria, nella posizione della variabile A
- 1 operazione floating-point $\rightarrow$ 3 accessi in memoria

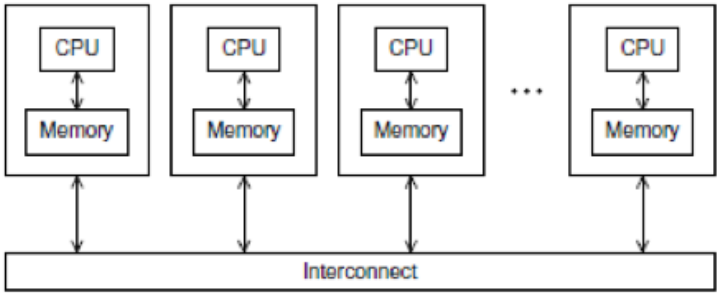
Stream

- Benchmark per la misura della bandwidth da e per la CPU
- Misura il tempo per
  - Copia a $\rightarrow$ c (copy)
  - Copia a*b $\rightarrow$ c (scale)
  - Somma a+b $\rightarrow$ c (add)
  - Somma a+b*c $\rightarrow$ d (triad)
- Misura della massima bandwith
- http://www.cs.virginia.edu/stream/ref.html

Condivisa Distibuita

- Le architetture MIMD classiche e quelle miste CPU GPU sono suddivise in due categorie
  - Sistemi a memoria condivisa dove ogni singolo core ha accesso a tutta la memoria
  - Sistemi a memoria distribuita dove ogni processore ha la sua memoria privata e comunica con gli altri tramite scambio di messaggi.



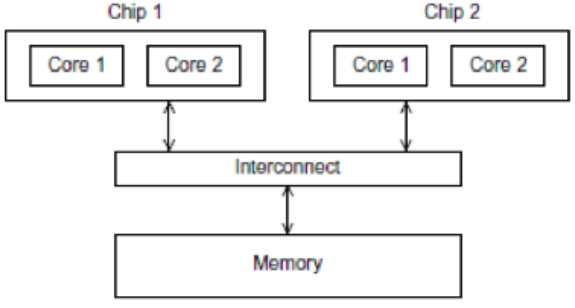


- I moderni sistemi multicore hanno memoria condivisa sul nodo e distribuita fra i nodi.

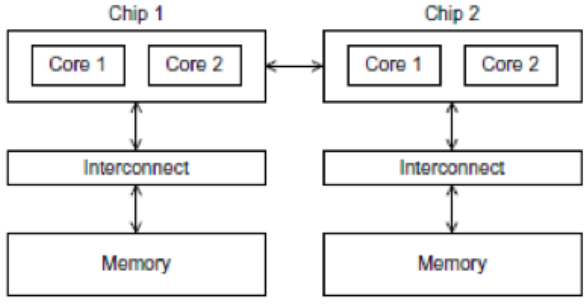
Memoria condivisa

Nelle architetture a memoria condivisa con processori multicore vi sono generalmente due tipologie di accesso alla memoria principale

- Uniform Memory Access dove tutti i cores sono direttamente collegati con la stessa priorità alla memoria principale tramite il sistema di interconnessione
- Non Uniform Memory Access dove ogni processore multicore può avere accesso privilegiato ad un blocco di memoria e accesso secondario agli altri blocchi.



UMA



NUMA

UMA NUMA

- Problemi principali:
  - se un processo o thread viene trasferito da una CPU ad un'altra, va perso tutto il lavoro speso per usare bene la cache
  - macchine UMA: processi "memory intensive" su più CPU contendono per il bus, rallentandosi a vicenda
  - macchine NUMA: codice eseguito da una CPU con i dati nella memoria di un'altra portano a rallentamenti
- Soluzioni:
  - binding di processi o thread alle CPU
  - memory affinity

RETI

- Tutti i sistemi caratterizzati da elevate potenze di calcolo sono composti da diversi nodi, a memoria condivisa sul singolo nodo e distribuita fra i nodi
  - i nodi sono collegati fra di loro da topologie di interconnessione più o meno complesse e costose
- Le reti di interconnessione commerciali maggiormente utilizzate sono
  - Gigabit Ethernet : la più diffusa, basso costo, basse prestazioni
  - Infiniband : molto diffusa, elevate prestazioni, costo elevato (50% del costo di un cluster)
  - Myrinet : sempre meno diffusa dopo l'avvento di infiniband, vi sono comunque ancora sistemi HPC molto importanti che la utilizzano
- Reti di interconnessione di nicchia
  - Quadrics
  - Cray

www.top500.org

Rank	Site	System	Cores	Rmax (TFlop/s)	Rpeak (TFlop/s)	Power (kW)
1	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
2	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
3	DOE/NNSA/LLNL United States	**Sequoia** - BlueGene/Q, Power BQC 16C 1.60 GHz, Custom IBM	1,572,864	17,173.2	20,132.7	7,890
4	RIKEN Advanced Institute for Computational Science (AICS) Japan	K computer, SPARC64 VIIIfx 2.0GHz, Tofu interconnect Fujitsu	705,024	10,510.0	11,280.4	12,660
5	DOE/SC/Argonne National Laboratory United States	**Mira** - BlueGene/Q, Power BQC 16C 1.60GHz, Custom IBM	786,432	8,586.6	10,066.3	3,945
6	Swiss National Supercomputing Centre (CSCS) Switzerland	**Piz Daint** - Cray XC30, Xeon E5-2670 8C 2.600GHz, Aries interconnect , NVIDIA K20x Cray Inc.	115,984	6,271.0	7,788.9	2,325
15	CINECA Italy	**Fermi** - BlueGene/Q, Power BQC 16C 1.60GHz, Custom IBM	163840	1788.9	2097.2	822