Architetture di processori

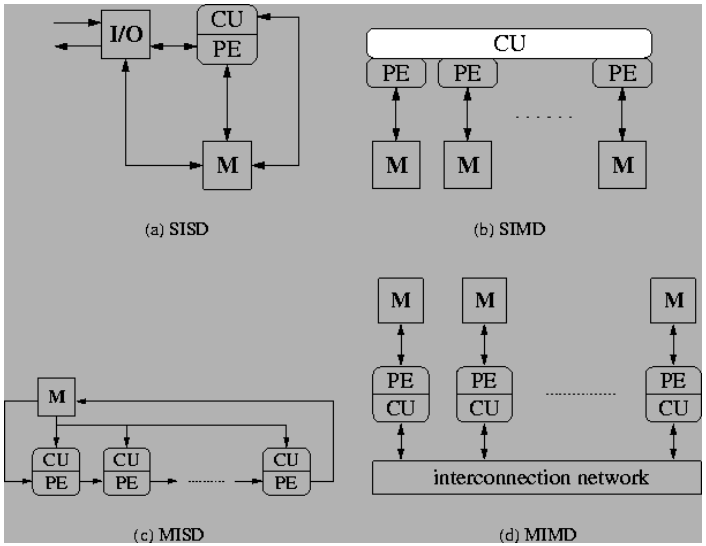
Classificazione di Flynn

SISD - Single Instruction Single Data

Le istruzioni sono eseguite sequenzialmente su un solo insieme di dati
Le macchine sequenziali comuni appartengono a questa classe che include anche le macchine **RISC** (*reduced instructions set)*, con pipelines più o meno complicate.
Calcolatori con pochi CPU appartengono in pratica pure a questa classe
I **DSP** (*digital signal processors*) appartengono in generale a questa classe.

MISD- Multiple Instructions Single Data

Esiste solo sulla carta!

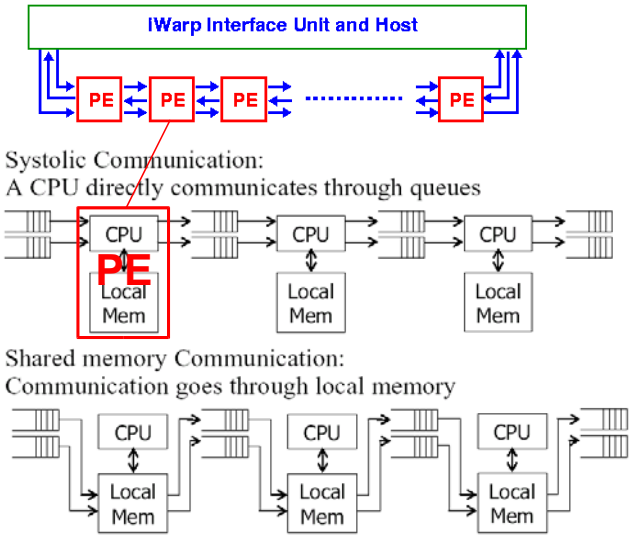


M = memory **CU** = control unit **PE** = processing element

SIMD - Single Instruction Multiple Data

Questi sistemi hanno in generale un gran numero di processori, più di 1024, che eseguono **in sincronismo** la stessa operazione su flussi paralleli di dati.

I **sistemi sistolici** possono considerarsi un sottoinsieme di questa classe.
Sono sistemi di processori interconnessi a pochi processori vicini. Ciascun processore esegue un numero limitato di operazioni su dati che si spostano da un processore al seguente. In generale le operazioni sono le stesse per ogni processore o per piccoli gruppi di processori. Come le macchine SIMD i calcoli sono eseguiti in sincronismo, alternando calcolo e comunicazione col vicino.
Il calcolo è controllato da un supervisore che alimenta i processori e gestisce la memoria.
Esempio: **Intel iWARP**.



Esempio: calcolo di un polinomio di ordine *n*

La regola di Horner per il calcolo di un polinomio è

$$y=((((a_n x+a_{n-1})x+a_{n-2})x+a_{n-3})x.....a_1)x+a_0$$

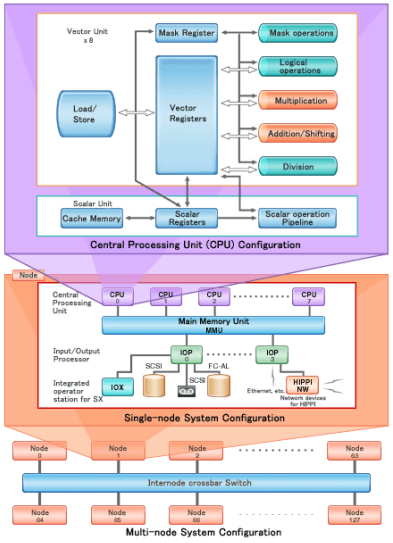
Il sistema è organizzato in modo che il primo processore moltiplica il coefficiente a_n per x e passa il risultato al processore successivo che aggiunge a_{n-1} ; il risultato passa al processore successivo che lo moltiplica per x e passa il risultato al processore successivo che aggiunge a_{n-2} .... e così via

Il sistema produce un polinomio per ogni ciclo, con un ritardo (latenza).

Un'altra sottoclasse importante è quella dei **processori vettoriali**.
La procedura è sequenziale, ma i dati sono un "vettore": la parte vettoriale del processo è **SIMD**.

Uno dei *supercomputers* rapidi del momento è un processore **vettoriale** NEC SX-6 con 5120 processori raggruppati in 640 nodi.
Ha una capacità di calcolo di 40 Tflop.
E' installato al Earth Simulation Center (ESC) a Yokohama.

Lo schema a lato rappresenta un processore NEC SX-6 con 128 nodi.



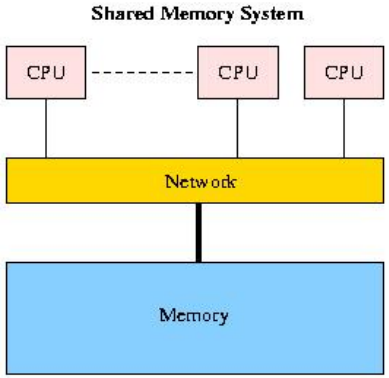
MIMD- Multiple Instructions Multiple Data

Questi sistemi eseguono istruzioni diverse su flussi diversi di dati

SM-MIMD - Shared Memory

Tutti i processori accedono alla stessa memoria che possiede un sistema di indirizzi globale.

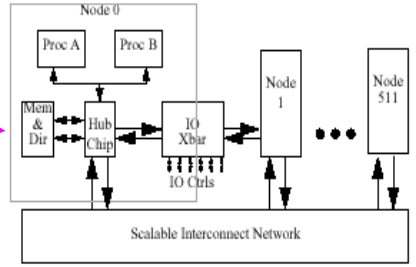
I processori comunicano tra di loro attraverso un commutatore (*switch*) e la memoria.



DM-MIMD - Distributed Memory,
Ogni processore ha la sua memoria.
I processori comunicano tra di loro attraverso un commutatore, *crossbar switch* o altro

I *clusters* o ***farms*** sono un sottoinsieme di questa classe

Un altro sottoinsieme di questa classe sono i **ccNUMA** *(cash coherent Non Uniform Memory Access)*
Ciascun processore ha la sua memoria. Un nodo di più processori può avere una **memoria collettiva**. I tempi di accesso alla memoria dipendono dalla "distanza" tra i processori. La coerenza delle memorie è assicurata da *hardware* e/o *software*



Beowulf

I *clusters* o *farms* di PC sotto Linux, sono spesso identificati col nome di Beowulf, dal nome di un progetto iniziato nel 1994 dalla NASA al Goddard Space Flight Center.

***Massively Parellel Processor*, MPP**

Indica un sistema costituito da un numero elevato, più di 128, processori uguali che cooperano all'esecuzione di un programma.

Accelerated Strategicic Computing Initiative, **ASCI**

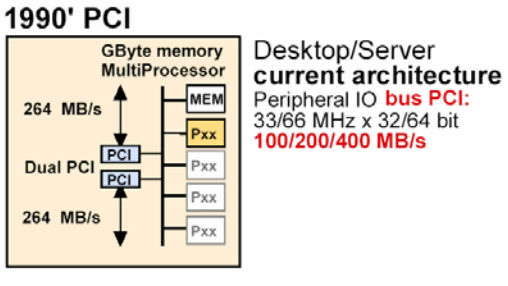
Iniziativa del governo degli Stati Uniti iniziata intorno al 1994 che ha stimolato l'industria dei *supercomputers*.

GRID

Progetto per utilizzare la potenza di calcolo installata (disponibile) in luoghi distanti connessi da Internet

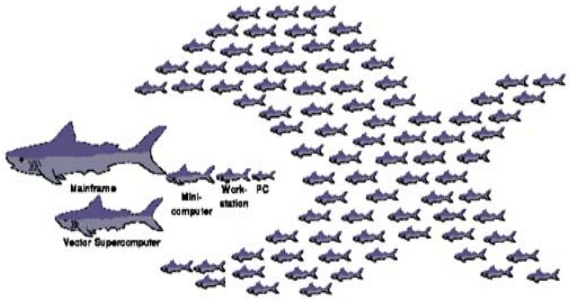
Il processore universale oggi è il PC

Come evolverà?



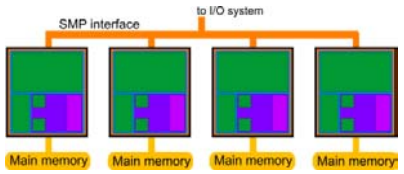
Due linee di sviluppo:

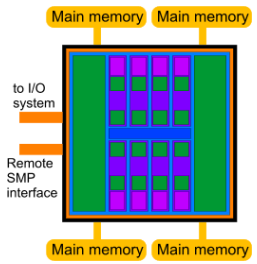
- **processori con molte unità di processo (cores) in un solo chip**
- **interconnessioni tra processori in rete via DMA (InfiniBand)**

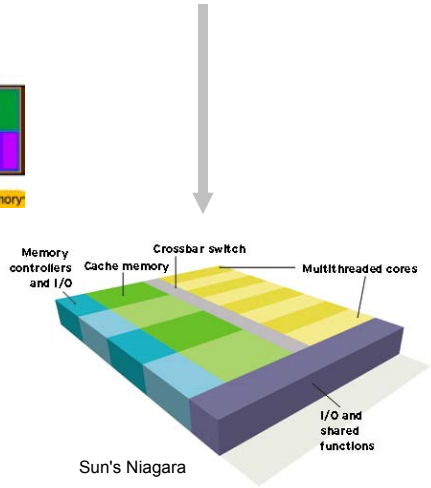


NOW Project
(http://now.cs.berkeley.edu)

da un SMP (*symmetric multiprocessor*) a un *chip* con molte unità di processo (*cores*)

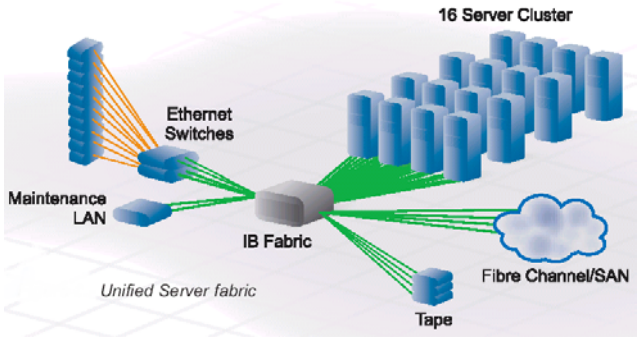






Sun's Niagara

200X: Infiniband as cluster interconnect?



SGI's Columbia 10240 processor installation at NASA Ames has Intel Itanium 2-processor chips, an **InfiniBand** interconnect by Voltaire and runs Linux. It has made 42,7 LINPACK teraflops

Parallelismo nei programmi

Granularità

Fine *(fine grain)*

Il problema può essere trattato da un programma semplice che è eseguito in parallelo da ciascun processore

- Assenza di connessioni tra un flusso e i vicini: es. processi vettoriali. La logica di parallelismo è intrinseca all'*hardware*

- Necessità di connessioni coi flussi vicini: es. calcolo dei valori di un campo scalare o vettoriale
  La logica di parallelismo fa parte del sistema operativo. Talvolta è necessario avere un processore come supervisore

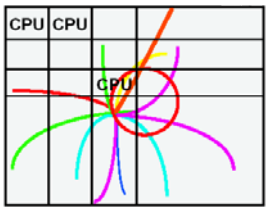
La generazione di tracce con metodi di Monte Carlo è *fine grain*

Grossolana *(coarse grain)*

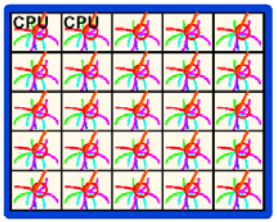
Il problema richiede sottoprogrammi massicci e solo questi sono eseguiti in parallelo.

Se ci sono necessità di comunicazione coi flussi vicini, queste possono essere soddisfatte dalla memoria comune, come in SM-MIMD, o da un protocollo di comunicazione (*message passing*), come in DM-MIMD I metodi di programmazione tengono conto parzialmente di queste necessità. Esempi PVM (*Parallel Virtual Machine*) e MPI (*Message Passing Interface*).Tuttavia rimane necessario frammentare manualmente certe parti dei programmi per preparare l' esecuzione parallela o per migliorare l'efficienza.

Nel caso dei *clusters* o *farms* usati in fisica delle alte energie ogni processore tratta un evento in indipendenza completa, senza comunicazioni coi vicini: è un parallelismo *very coarse grain*



Massive parallel system
ONE event, ALL processors
- Low latency
- Complex I/O
- Parallel programming



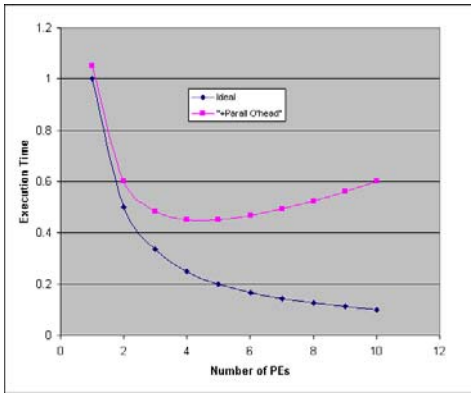
Farm of processors
ONE event, ONE processor
- High latency (larger buffers)
- Simpler I/O
- Sequential programming

S. Cittolin, CERN CMS

Speedup è il fattore secondo il quale un programma eseguito da *n* processori è più rapido di un programma eseguito da un solo processore.

Legge di Amdahl:

$$T(n) = \beta T(1) + (1- \beta)\frac{T(1)}{n}$$



Parallel Overhead

Quando si divide un programma in canali (*threads*) paralleli non si ottiene lo *speedup* teorico

Referenze

Per architetture di processori:
A. J. van der Steen, J. J. Dongarra, Overview of Recent Supercomputers,
http://www.top500.org/html/orsc/2005

Per documentazione su PVM e MPI:
http://nereida.deioc.ull.es/html/pvm.html

Field Programmable Gate Array, FPGA

Forniscono elementi di trattamento logico, e in certi casi numerico, nel processo di acquisizione dati, senza dover ricorrere a logica progettata espressamente per la funzione richiesta.

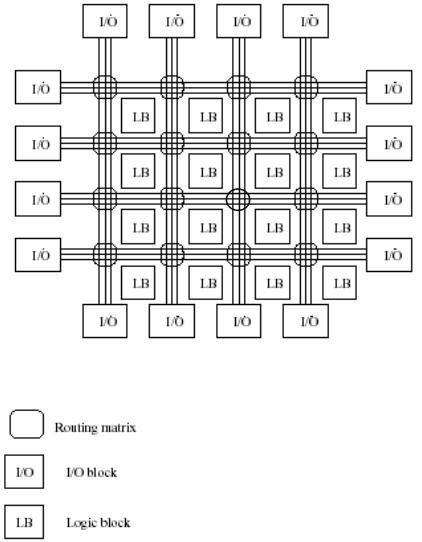
Sono di solito realizzate in tecnologia CMOS e funzionano a frequenza di orologio fino a centinaia di Mhz.

La funzioni realizzate più comunemente sono:

- Filtri digitali, tanto FIR quanto IIR
- *Pattern recognition* elementare (locale)
- Calibrazione, statica o dinamica

Una FPGA semplice contiene:

- unità di ingresso/uscita,
- piccole matrici per interconnessioni programmabili
- blocchi per funzioni logiche anche complesse



FPGA

Realizzazioni tipiche usando FPGAs:

Prototipi di logica complessa che in seguito sarà realizzata in ASICs in gran serie

Logica che si deve produrre in un numero grande ma ancora insufficiente per giustificare un ASIC

Logica complessa della quale non si è sicuri e che si pensa di dover cambiare in seguito, riprogrammando

Processori speciali che eseguono a ripetizione una funzione particolare, per es. coprocessori per PC

Referenze FPGA

S. Haas, ELEC 2005, Electronics in High Energy Physics: http://humanresources.web.cern.ch/humanresources/external/training/tech/special/ELEC2005.asp#Winter

S.Brown and J. Rose, Architecture of FPGAs and CPLDs: A Tutorial, IEEE Design and Test of Computers, Vol. 12 No. 2, 1996, pp42-57, http://www.eecg.toronto.edu/~jayar/pubs/brown/survey.pdf