POLITECNICO DI MILANO

Introduzione al Calcolo Parallelo

Algoritmi e Calcolo Parallelo

Daniele Loiacono

Riferimenti

- Questo materiale deriva dalle slide del prof. Lanzi per il corso di Informatica B, A.A. 2009/2010

- Il materiale presente in queste slide è preso dai seguenti tutorial:
  - Introduction to Parallel Computing
    Blaise Barney, Lawrence Livermore National Laboratory
    https://computing.llnl.gov/tutorials/parallel_comp/
  - Anche pubblicato come Dr.Dobb's "Go Parallel"
    Introduction to Parallel Computing: Part 2
    Blaise Barney, Lawrence Livermore National Laboratory

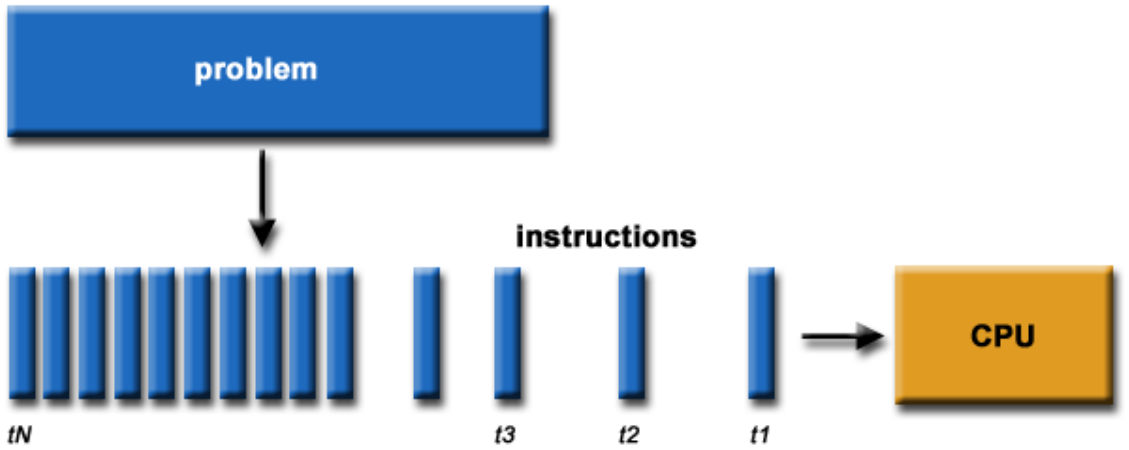
Calcolo Sequenziale

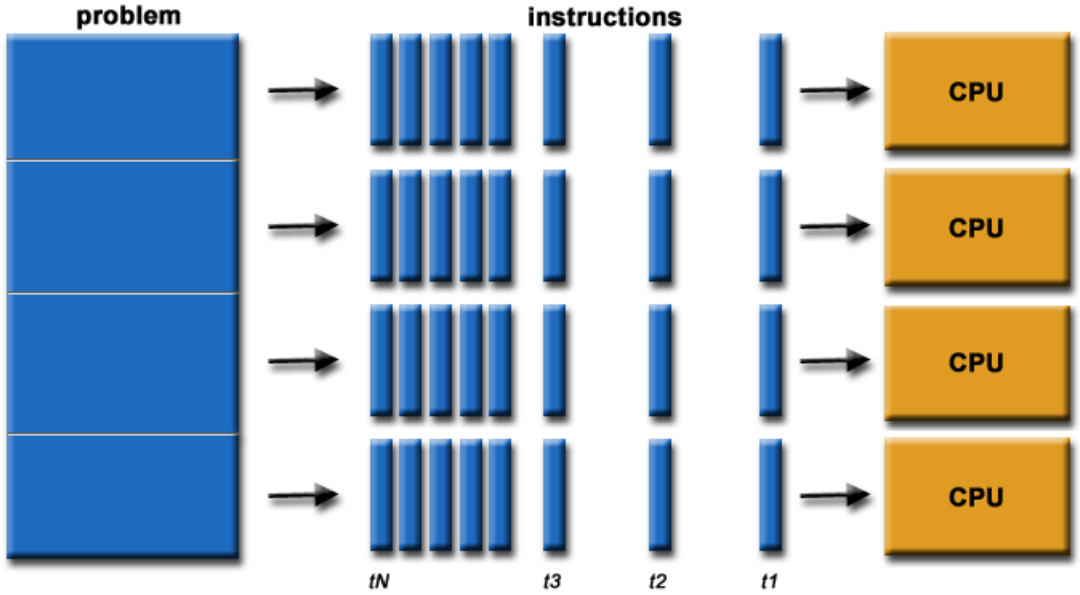
- Tradizionalmente i programmi sono stati scritti per un modello di computazione sequenziale (Von Neuman)
- Per esere eseguiti su un computer con una singola CPU
- Un problema viene spezzato in sequenze (discrete) di istruzioni che sono eseguite in sequenza (una dopo l'altra)
- In un dato istante di tempo solo un'istruzione è in esecuzione

Che Cos'è il Calcolo Parallelo?

- Con il termine calcolo parallelo si intende l'uso di più unità di computazione (più CPU) per risolvere problemi
- Un problema viene decomposto in parti che posso essere risolte in maniera concorrente (in parallelo)
- Ogni parte è spezzata in sequenze di istruzioni da eseguire su una singola CPU
- Le istruzioni di ognuna delle parti sono eseguite simultaneamente su CPU differenti.

Calcolo Sequenziale vs Calcolo Parallelo





Quali Assunzioni?

- Risorse di Calcolo
  - Un singolo calcolatore con più processori
  - Un numero arbitrario di calcolatori connessi in rete
  - Una combinazione dei due
- Problema da Risolvere
  - Può essere suddiviso in parti che possono essere risolte contemporaneamente, in parallelo
  - Esecuzioni di più istruzioni in un certo istante di tempo
  - Se risolto con più CPU in parallelo richiede meno tempo rispetto a risolverlo in maniera sequenziale

Il Mondo è Parallelo!

Quali Applicazioni? (Ieri)

- Storicamente il calcolo parallelo è stato visto come un paradigma costoso e di alto livello
- È stato quindi utilizzato principalmente per risolvere problemi scientifici e ingegneristici di alto livello:
- Alcuni esempi
  - Problemmi atmosferici e ambientali
  - Fisica Atmosphere, Earth, Environment
  - Scienze della vita
  - Chimica
  - Geologia, fenomeni sismici
  - Ingegneria meccanica
  - Ingegneria elettronica
  - …

Quali Applicazioni? (Oggi)

- Le applicazioni commerciali forniscono le motivazioni principali per lo sviluppo di calcolatori sempre più veloci
- È necessario processare grandi quantità di dati con tecniche sofisticate e velocemente
- Alcuni esempi
  - Database, data mining
  - Oil exploration
  - Motori di ricerca Web
  - Elaborazione immagini medicali e diagnosi
  - Design farmaceutico
  - Modelli finanziari ed economici
  - Grafica avanzata e realtà virtuale
  - …

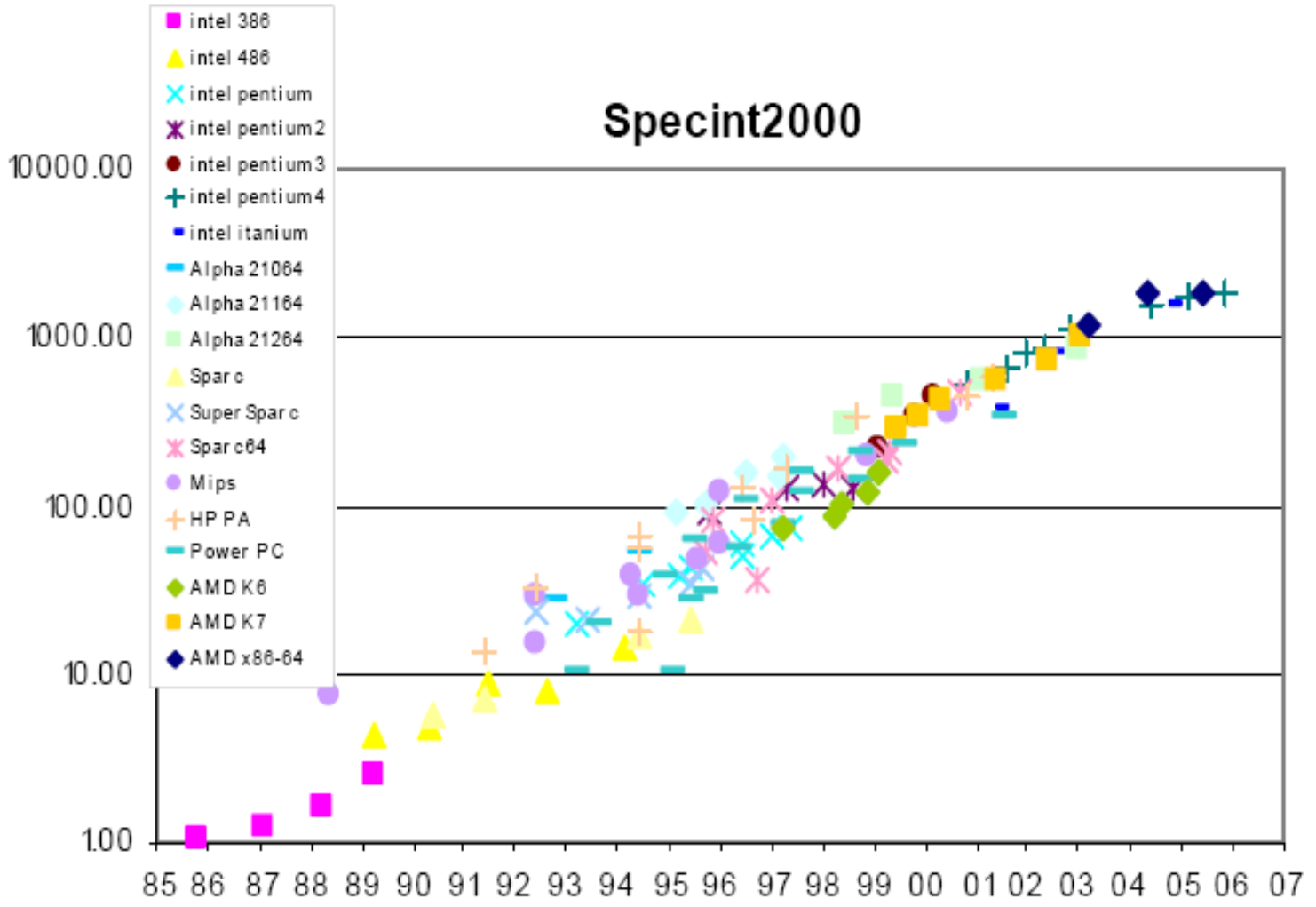
Perchè?

- Risparmiare tempo e denaro
- Risolvere problemi molto (troppo) difficili
  - Esistono problemi che sono troppo grossi o complessi per poter essere risolti utilizzando un singolo calcolatore (o sarebbe impraticabile)
  - Grand Challenge
- Dare la possibilità di fare più cose contemporaneamente
- Utilizzare risorse distribuite su tutto il pianeta (SETI@HOME)

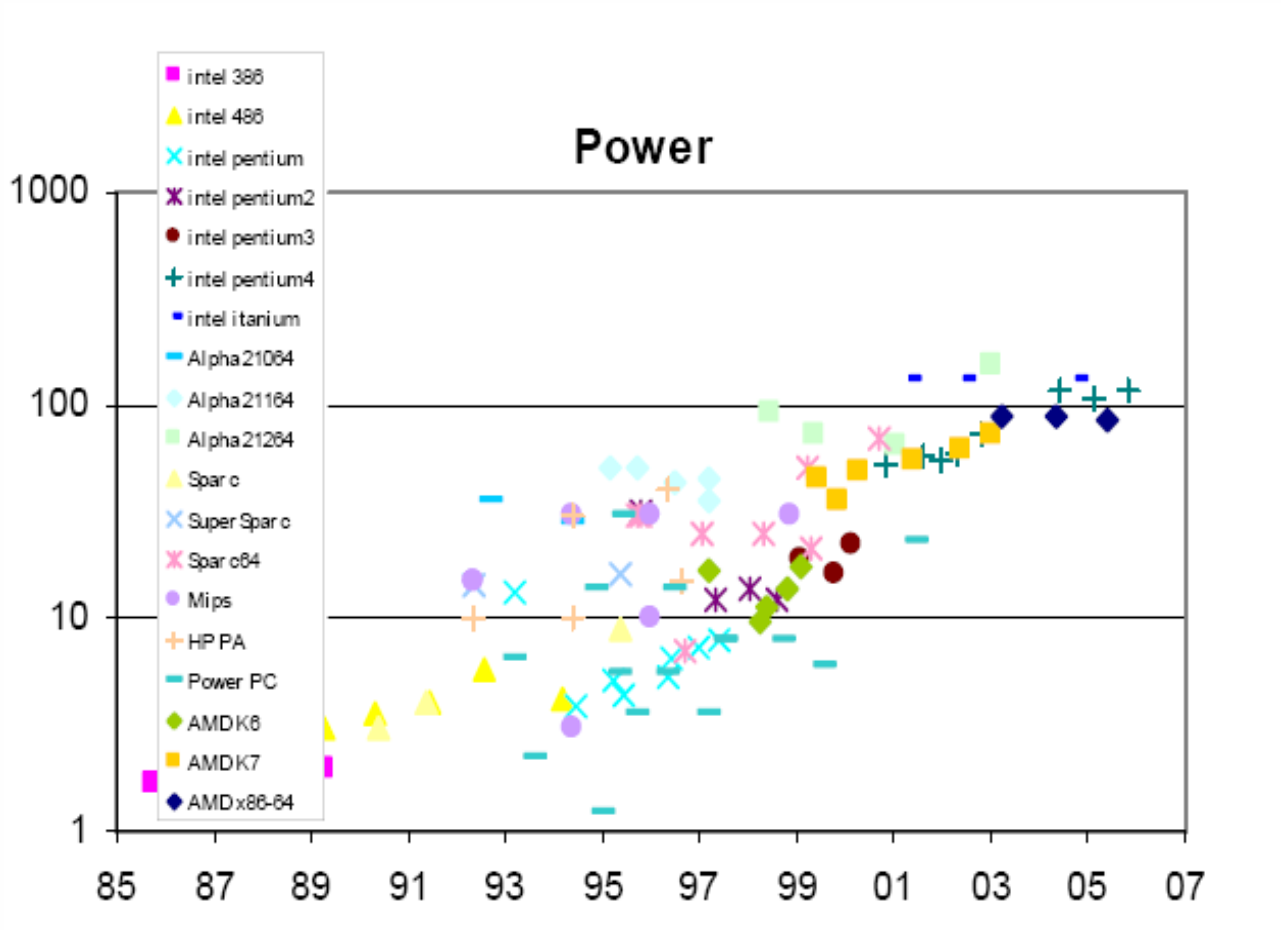
Limitazioni del Calcolo Seriale

- Ci sono diverse ragioni che limitano la costruzione di calcolatori sequenziali sempre più veloci
- Velocità di trasmissione
  - La velocità di un calcolatore sequenziale dipende dalla velocità di trasferimento dei dati
  - Il limite è la velocità della luce (30 cm/nanosecondo) e i limiti del rame (9 cm/nanosecondo).
  - Velocità superiori richiedono maggiore vicinanza fra le unità che processano l'informazione
- Limiti della miniaturizzazione
- Limitazioni economiche

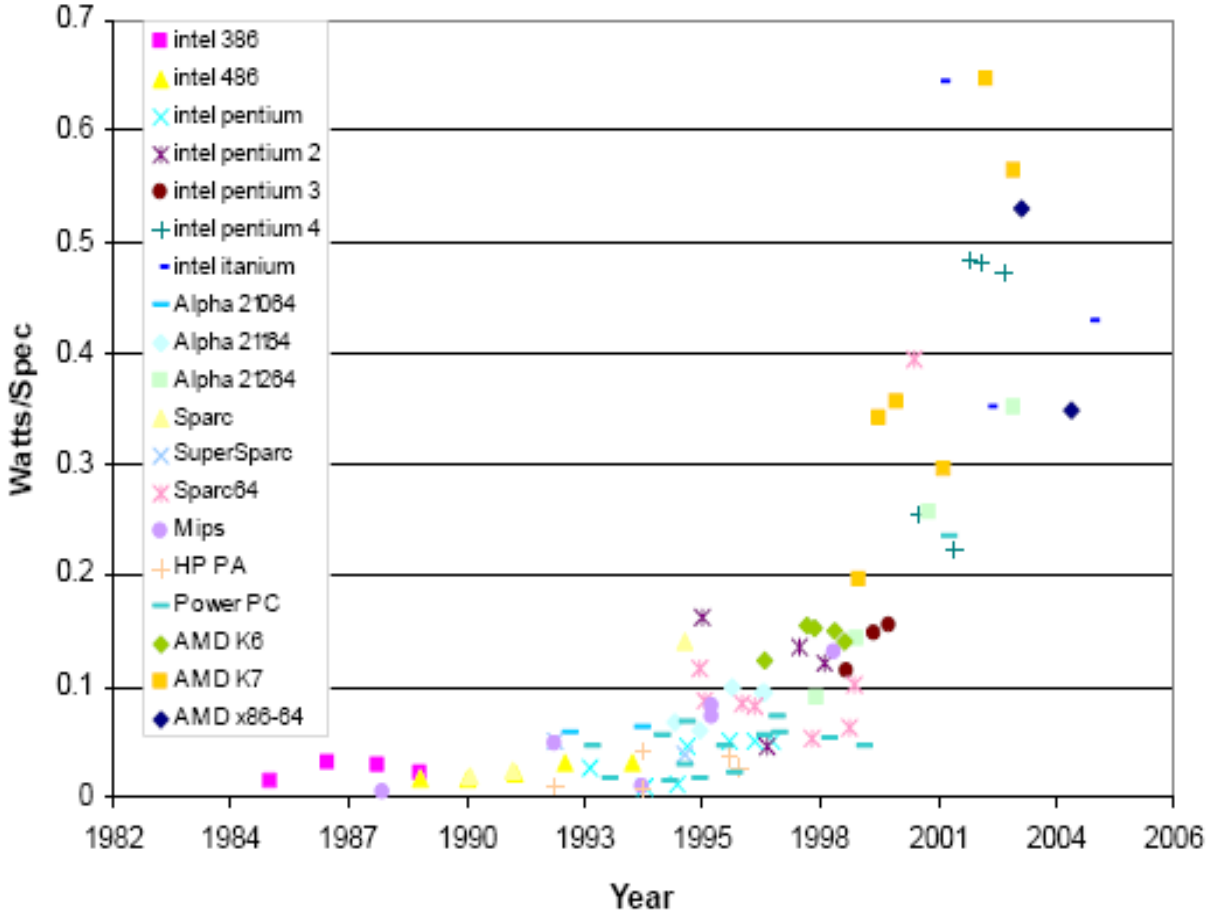
Prestazione degli (uni)Processori



Consumo di potenza (W)



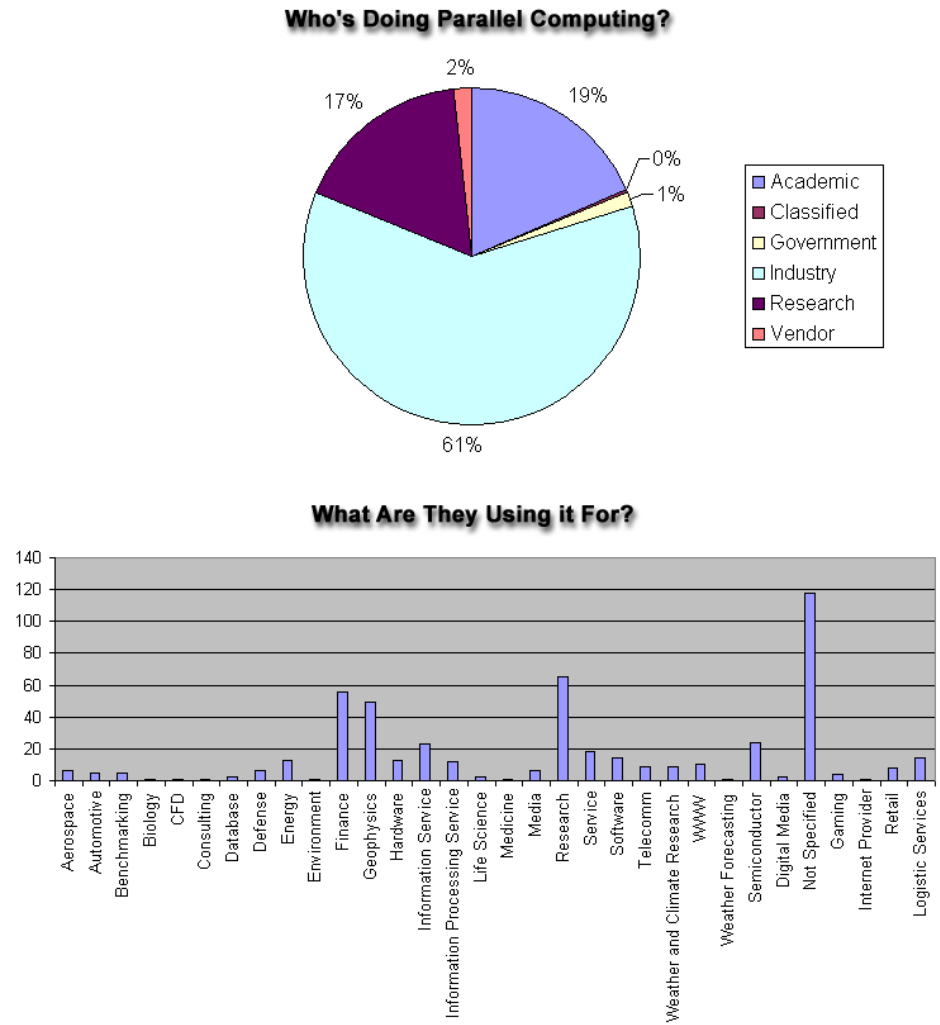
Efficienza (Potenza/Prestazioni)



Investimenti sul Calcolo Parallelo

- Anni 80
  - Espansione dei processori superscalari
  - Miglioramenti del 50% nelle prestazioni
  - Transistor utilizzati per creare parallelismo implicito
- Anni 90
  - Ritorni si riducono
  - Sfruttare al meglio il parallelismo implicito
  - Prestazioni al disotto delle attese
  - Progetti ritardati e cancellati
- Anni 2000
  - Inizio dell'era dei multicore
  - Parallelismo esplicito

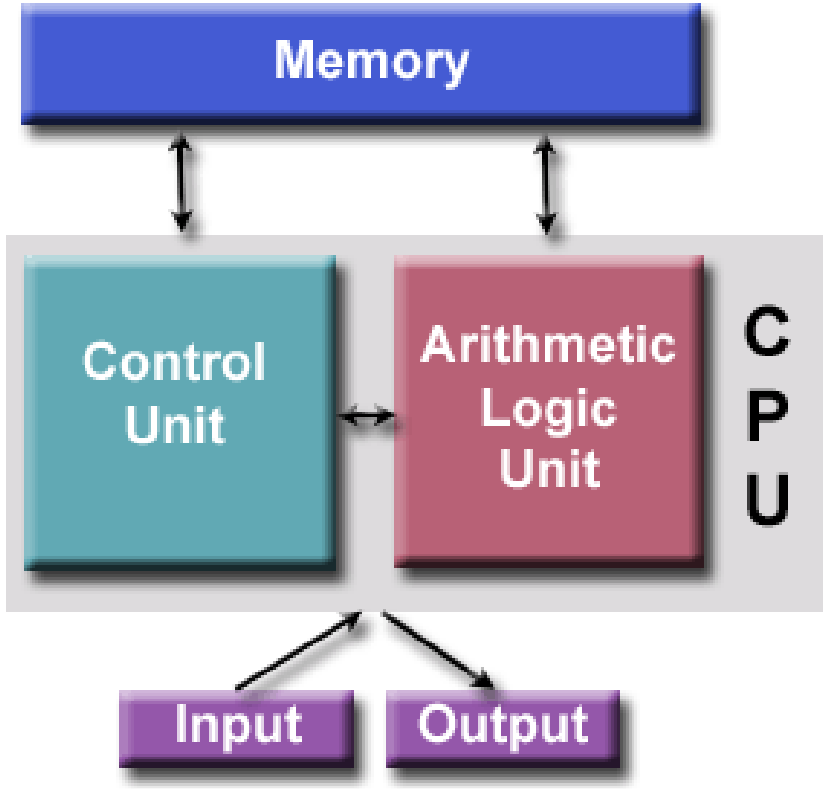
Chi e Come?



Fonte: Top500.org

Macchina di Von Neumann

- Matematico Ungherese, John von Neumann per primo indicò i requisiti generali (la struttura) di un calcolatore elettronico in un articolo del 1945
- Da allora, praticamente tutti i calcolatori hanno seguito lo stesso design che differiva dai primi calcolatori che erano programmati in maniera "hard-wired" (cablata)



Macchina di Von Neumann

- Quattro componenti
  - Memoria
  - Control Unit
  - Arithmetic Logic Unit
  - Input/Output
- Random access memory (RAM) utilizzata per memorizzare sia i programmi che i dati
- Istruzioni codificati come dati che specificano all'unità di controlla cosa fare
- I dati sono informazioni utilizzati dal programma
- L'unità di controllo recupera (fetch) le istruzioni da eseguire e i dati su cui eseguirle; decodifica le istruzioni; coordina l'esecuzione
- Aritmetic Unit esegue elementari operazioni aritmetiche
- Input/Output si interfaccia all'operatore umano

Classificazione di Flynn

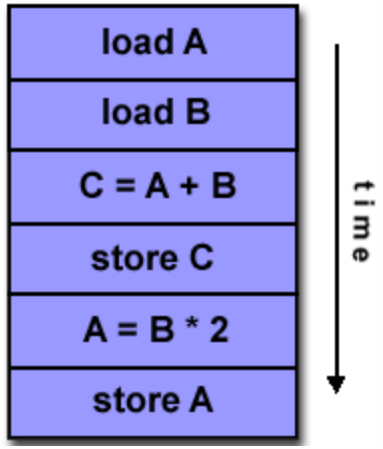
- I calcolatori paralleli possono essere classificati in modi differenti a seconda delle loro funzionalità
- La classificazione maggiormente usata è la classificazione di Flynn, utilizzata fino dal 1966
- Classifica le diverse architetture multiprocessore rispetto a due dimensioni: quella delle istruzioni e quella dei dati
- Ognuna di queste dimensioni può avere due possibili valori
  - Single
  - Multiple

Classificazione di Flynn

S I S D Single Instruction Single Data	S I M D Single Instruction Multiple Data
M I S D Multiple Instructions Single Data	M I M D Multiple Instructions Multiple Data

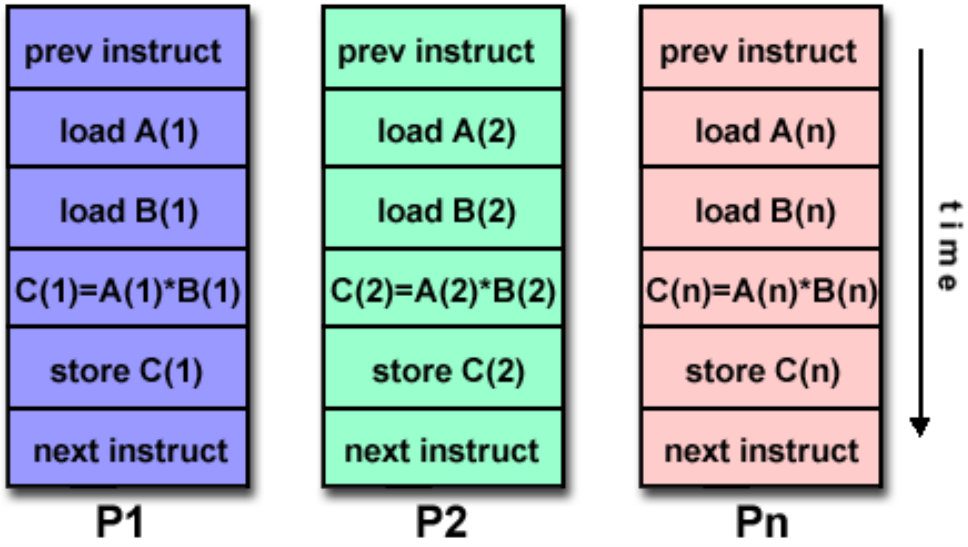
Single Instruction, Single Data (SISD)

- Singolo calcolatore sequenziale (non-parallelo)
- Single instruction
  - La CPU gestisce un solo stream di istruzioni
- Single data
  - Un solo data stream è utilizzato come input
- L'esecuzione è deterministica
- Individua la tipologia di calcolatori più comune e più vecchia
- Esempi: i primi mainframe, minicomputer, workstation; la maggioranza di tutti i PC moderni



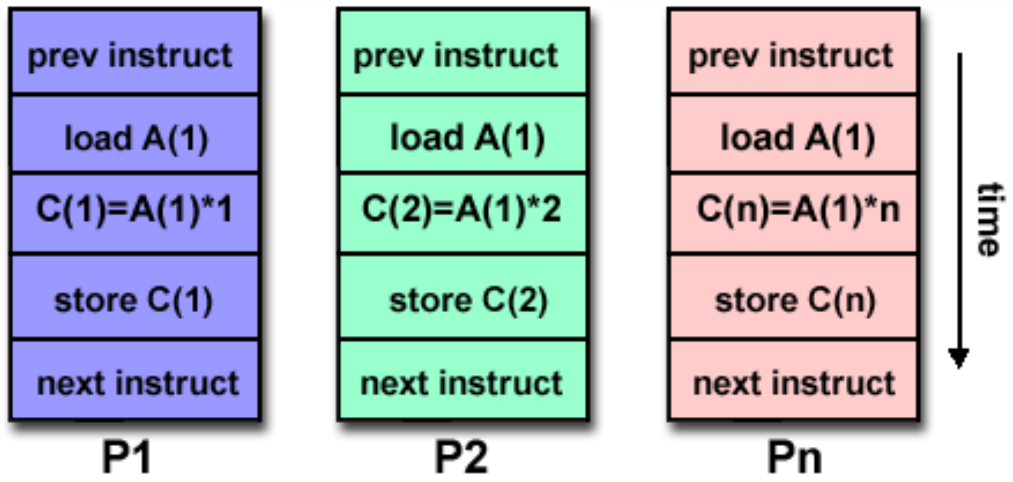
Single Instruction, Multiple Data (SIMD)

- Single instruction: tutte le unità eseguiscono la stessa istruzione allo stesso istante di tempo (ciclo di clock)
- Multiple data: ogni unità opera su un elemeno differente
- Esecuzione sincrona (lockstep) e deterministica
- Esempi
  - Processor Arrays
  - Vector Pipelines
- La maggior parte dei calcolatori moderni che utilizzano graphics processor units (GPUs) seguono un modello SIMD



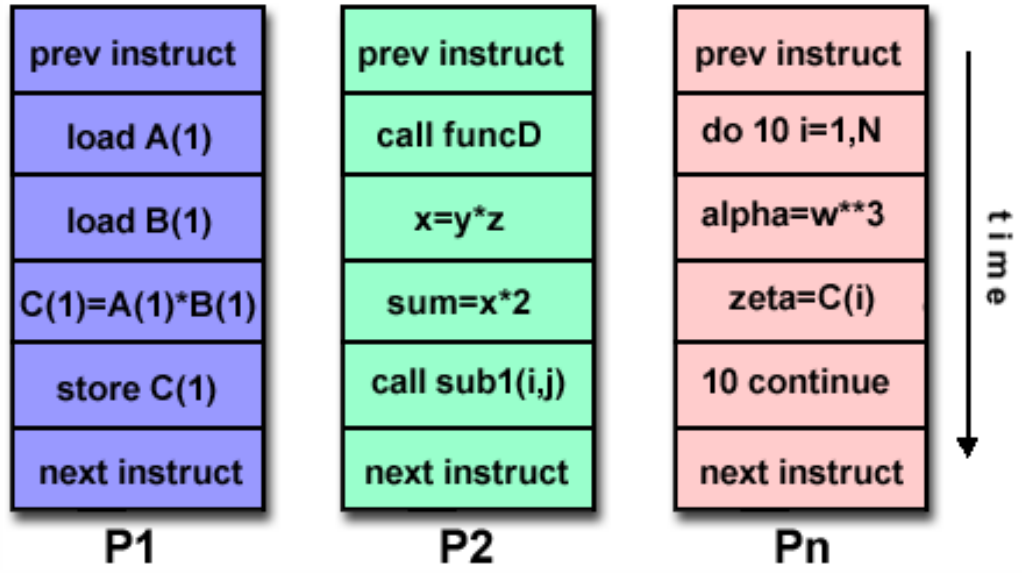
Multiple Instruction, Single Data (MISD)

- Singolo stream di dati che alimenta più unità di elaborazione
- Ogni unità opera sui dati indipendentemente con stream di istruzioni differenti
- Pochisssimi esempi di questo tipo di calcolatori
- Filtraggio multiplo su un singolo segnale
- Crack di un singolo segnale crittografato utilizzando più algoritmi



Multiple Instruction, Multiple Data (MIMD)

- Attualmente il modello più diffuso. La maggior parte dei calcolatori paralleli rientrano in questa categoria
- Multiple Instruction: ogni processore esegue un instruction stream differente
- Multiple Data: ogni processore lavora su un diverso data stream differente.
- L'esecuzione può essere sincrona o asincrona, deterministica o non-deterministica
- Esempi: maggior parte dei supercomputer, cluster, grid, ecc.

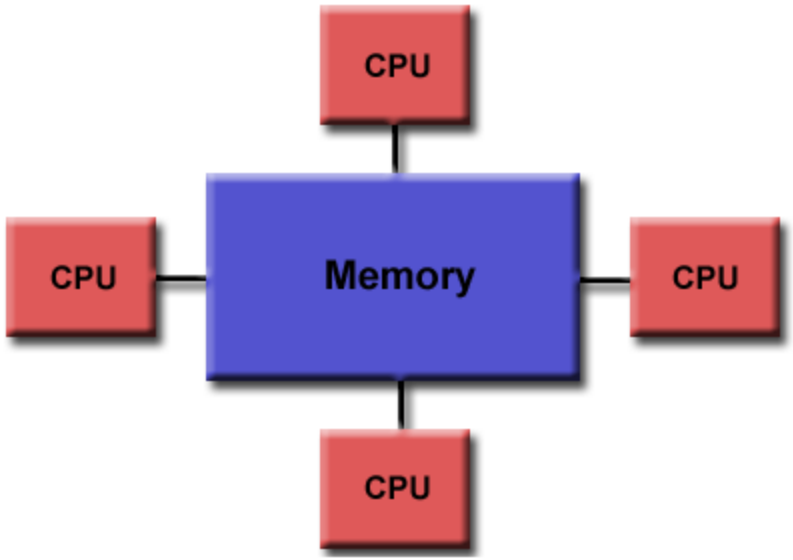


Organizzazione della Memoria: Shared Memory

- Nel modello a shared memory o memoria condivisa, tutti i processori hanno accesso a tutta la memoria vista come uno spazio di indirizzamento globale
- Più processori operano indipendentemente ma condividono la memoria
- Le modifiche di una locazione di memoria da parte di un processore è anche visibile da tutti gli altri processori
- Due tipologie di calcolatori
  - Uniform Memory Access (UMA)
  - Non-Uniform Memory Access (NUMA)

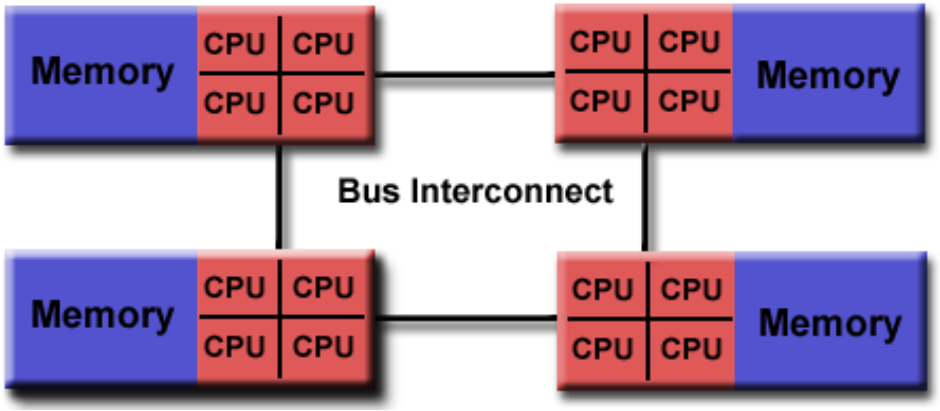
Shared Memory: Uniform Memory Access (UMA)

- Tipicamente rappresentato dai Symmetric Multiprocessor (SMP)
- Più processori identici, stesso tipo di accesso e di tempi di accesso alla memoria
- Anche noti come CC-UMA o Cache Coherent UMA
- Se un processore aggiorna una locazione di memoria condivisa, tutti gli altri processori sono a conoscenza dell'aggiornamento



Shared Memory: Non-Uniform Memory Access (NUMA)

- Costituiti da due o più SMP
- Ogni SMP ha accesso diretto alla memoria di altre SMP
- Non tutti hanno stesso tempo di accesso
- L'accesso alla memoria attraverso il bus è più lento
- Se c'è cache coherency sono anche chiamati CC-NUMA (Cache Coherent NUMA)



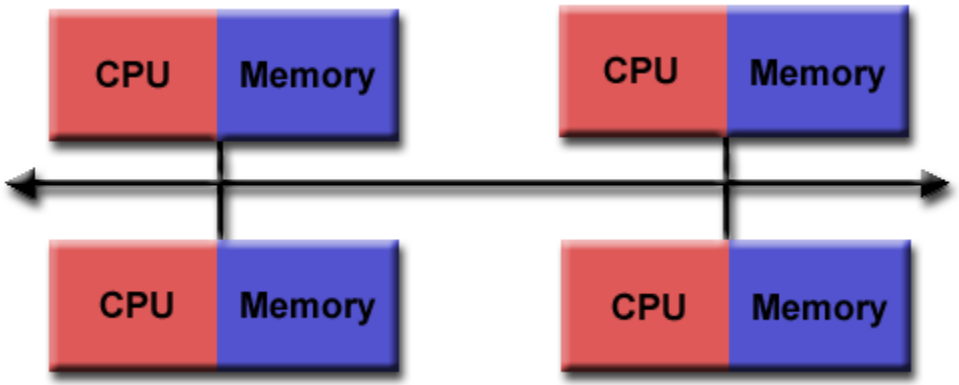
Shared Memory

- Vantaggi
  - Lo spazio di indirizzamento globale rende intuitiva la programmazione parallela
  - La condivisione dei dati fra processi è sia veloce e uniforme data la prossimità della memoria alla CPU

- Svantaggi
  - Scalabilità fra memoria e numero di CPU. Aumentare il numero di CPU può aumentare geometricamente il traffico fra memoria e CPU
  - Per i sistemi cache coherent aumenta il traffico associato alla gestione cache/memory
  - Il programmatore è responsabile per la sincronizzazione che garantisce l'accesso concorrente corretto alla memoria centrale
  - Il design e la produzione di calcolatori paralleli con modello a shared memory è sempre più difficile e costoso con l'aumento del numero di processori

Organizzazione della Memoria: Distributed Memory

- Richiede la presenza di una rete che connetta i processori
- I processori hanno la propria memoria locale che non è indirizzata o accessibile dalle altre unità
- Ogni processore opera independentemente. Le modifiche della memoria locale non hanno effetto sulla memoria di altri processori (non esiste la cache coherence)
- Se un processore deve aver accesso ai dati di un'altra unità, il programmatore deve definire esplicitamente quando e come i dati sono scambiati
- La tipologia di rete può cambiare molto ma può anche essere una semplice ethernet



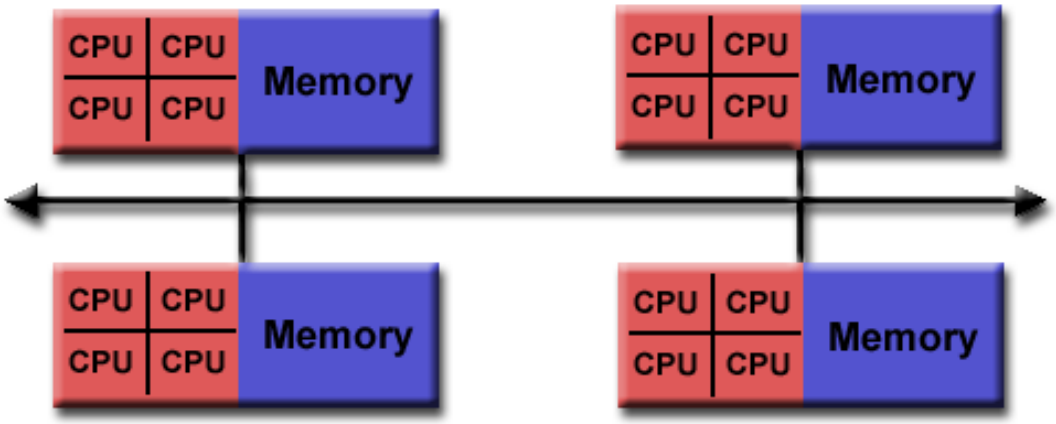
Organizzazione della Memoria: Distributed Memory

- Vantaggi
  - Scalabilità fra memoria e CPU. Incrementando il numero di processori, la grandezza della memoria aumenta proporzionalmente
  - Ogni processo accede rapidamente alla memoria locale (nessun overhead per la cache coherency)
  - Economicità: si possono utilizzare calcolatori e reti off-the-shelf

- Svantaggi
  - Il programmatore è responsabile per molti dei dettagli associati alla comunicazione fra processi
  - Difficile mappare le strutture dati basate su memoria condivisa a questo paradigma
  - Tempi per il non-uniform memory access (NUMA)

Organizzazione della Memoria: Hybrid Distributed-Shared Memory

- I calcolatori paralleli più grandi e veloci usano architetture ibride con memoria distribuita e condivisa
- La componente a memoria condivisa sono SMP con CC
- I processori di una SMP possono indirizzare la memoria della macchina come globale
- La componente distribuita della memoria è sulla rete che connette le diverse SMP
- Ogni SMP ha accesso solo alla propria memoria
- L'accesso alla memoria di un'altra SMP deve essere svolto attraverso la comunicazione sulla rete
- Il trend corrent indica un aumento di questo tipo di architettura
- I vantaggi e gli svantaggi sono quelli delle due architetture di base



Modelli di Programmazione Parallela

- Costituiscono un livello di astrazione sull'architettura hardware e il modello di memoria sottostante
- Diversi modelli di programmazione
  - Shared Memory
  - Threads
  - Message Passing
  - Data Parallel
  - Hybrid
- Questi modelli non sono specifici di un particolare tipo di macchina o di architettura di memoria
- Ogni modello di programmazione parallela (teoricamente) può essere implementato su qualsiasi architettura

Terminologia

- **Task:** un programma o un insieme di istruzioni che devono essere eseguite da un processore
- **Task Parallelo:** un task che può essere eseguito su più processori in modo safe (ottenendo risultati corretti)
- **Esecuzione Sequenziale**
  - L'esecuzione avviene un'istruzione alla volta (l'esecuzione tipica su singolo processore)
  - Virtualmente tutti i task paralleli hanno sezioni che devono essere eseguite in maniera sequenziale
- **Esecuzione Parallela**
  - Un programma viene eseguito come insieme di più di un task
  - Ogni task esegue la stessa istruzione o istruzioni differenti nel medesimo istante di tempo

Terminologia

- **Pipelining:** suddivisione di un task in passi eseguiti da unità differenti, su uno stream di input; simile a una catena di montaggio; è un modello di calcolo parallelo

- **Shared Memory:** architettura in cui tutti i processori hanno accesso diretto alla memoria condivisa. Nella programmazione, descrive un modello in cui i task paralleli hanno la stessa immagine della memoria e possono direttamente indirizzare e accedere alle locazioni di memoria

- **Symmetric Multi-Processor (SMP):** architectura hardware dove più processori condividono un unico spazio di indirizzamento

- **Distributed Memory:** nell'hardware, si riferisce a un'accesso alla memoria su rete. Nell'ambito dei modelli di programmazione si riferisce alla situazione in cui i task possono solo aver accesso alla memoria local del processo. L'accesso alla memoria di altre macchine viene implementato esplicitamente su rete

Terminologia

- **Comunicazione**
  - Si riferisce allo scambio di dati fra task paralleli
  - Esistino diverse tecniche possibili (con shared memory bus o su rete) network

- **Sincronizzazione**
  - Coordinamento dei task paralleli in tempo reale che è spesso associato con la comunicazione
  - Spesso implementato stabilendo un punto di sincronizzazione nell'applicazione dove un task non può procedere fino a quando un'altro task raggiunge lo stesso punto
  - Generalmente richiede l'attesa di uno o più task e quindi causa l'aumento del tempo di esecuzione

Terminologia

- **Granularità:** nel calcolo parallelo misura qualitativamente del rapporto fra calcolo e comunicazione
  - **Coarse:** quando tra eventi di comunicazione il tempo speso per l'elaborazione è piuttosto alto
  - **Fine:** il tempo speso per l'elaborazione tra gli eventi di comunicazione è relativamente piccolo

- **Speedup Osservato** di un codice parallelizzato è definito

  wall-clock time of serial exec/wall-clock time of parallel exec

  indicatore più semplice e maggiormente usato per misurare la performance di un programma parallelo

Terminologia

- **Parallel Overhead:** l'ammontare di tempo richiesto per coordinare I task paralleli, in contrapposizione al tempo utilizzato per l'effettiva elaborazione. Include il task start-up time, il tempo per la sincronizzazione, la comunicazione, ecc.

- **Scalabilità** si riferisce alla capacità di avere un incremento delle prestazioni (del parallel speedup) proporzionale all'aumento del numero dei processori. Fattori che contribuiscono alla scalabilità sono: l'hardware, l'algoritmo usato, l'overhead, ecc.

- **Multi-core Processors:** più processori (cores) su un unico chip

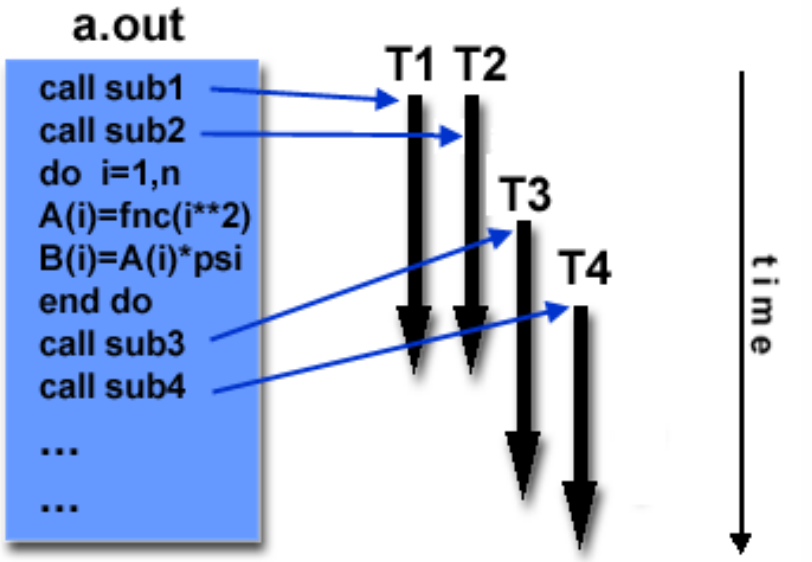
- **Cluster Computing:** utilizzo di più unità (processori, reti, SMP) per costruire un sistema parallelo

Modelli di Programmazione Parallela
Shared Memory Model

- I task condividono uno spazio di indirizzamento comune a cui possono accedere in lettura/scrittura in maniera asincrona
- Meccanismi per controllare l'accesso alla memoria condivisa (lock/semafori)
- Dal punto di vista della programmazione, non sono necessarie istruzioni per lo scambio di dati fra task.
- Difficile gestire dati localmente specialmente per programmatori mediamente esperti

Modelli di Programmazione Parallela Thread

- Un processo con più percorsi di esecuzione concorrenti
- Analogia: un programma a.out con diverse subroutine; a.out esegue una parte sequenziale e, successivamente, crea una serie di task (threads) che possono essere eseguiti parallelamente
- Ogni thread ha dati locali ma ha accesso alle risorse di a.out e al suo spazio di memoria
- Un thread può essere visto con una subrouting del main
- Ogni thread può eseguire una qualsiasi subrouting contemporaneamente ad altri thread



Modelli di Programmazione Parallela Thread

- I thread comunicano fra di loro attraverso la memoria globale. Questo richiede costrutti di sincronizzazione per garantire che più thread non aggiornino le stesse locazioni allo stesso tempo

- I thread possono essere creati e terminare, a.out però è sempre presente e garantisce le risorse condivise necessarie fino a quando l'applicazione non è terminata

- I thread sono tipicamente associati ad architetture shared memory (con memoria condivisa)

Modelli di Programmazione Parallela: Thread

- Dal punto di vista della programmazione, le implementazioni dei thread tipicamente includono:
  - Una libreria di subroutine che sono richiamate dal codice parallelo
  - Un insieme di direttive da includere nel codice sequenziale e parallelo
  - In entrambi i casi il programmatore è responsabile e determina tutto il parallelismo

- Diverse implementazioni, il tentativo di standardizzazione ha condotto a due implementazioni
  - POSIX Threads
  - OpenMP

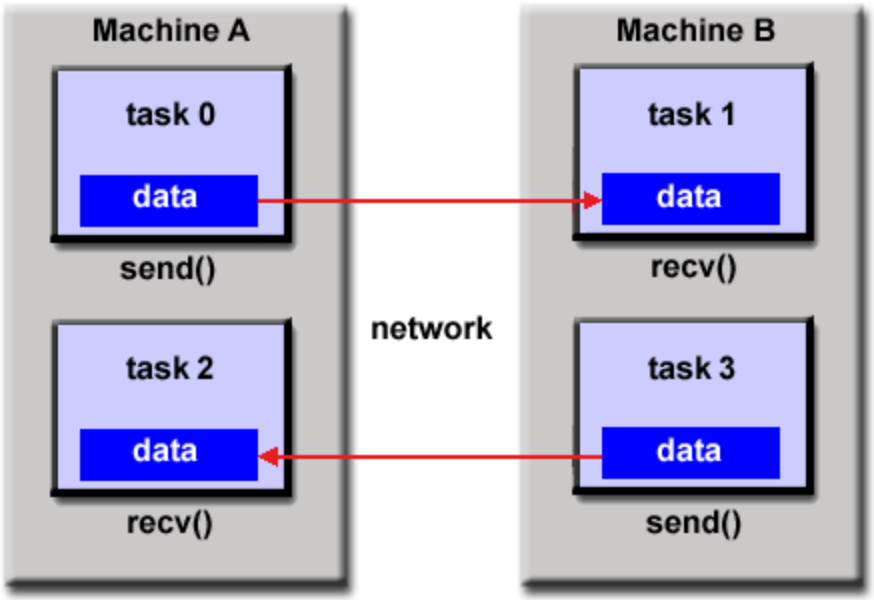
POSIX Thread vs OpenMP

- **Thread POSIX**
  - Basati su librerie, richiedono la gestione esplicita del parallelismo da parte del programmatore
  - Specified by the IEEE POSIX 1003.1c standard (1995).
  - Solo per il linguaggio C
  - Solitamente chiamati Pthreads

- **OpenMP**
  - Basato su direttive del compilatore
  - OpenMP Fortran API (28/10/1997)
  - C/C++ API (1998).
  - Portabile e multipiattaforma (sia Unix che Windows NT)
  - Molto semplice da utilizzare
  - Da la possibilità di avere un parallelismo incrementale

Modelli di Programmazione Parallela
Message Passing

- Un insieme di task, ognuno con la propria memoria locale
- Più task possono risiedere sulla stessa macchina fisica oppure su un numero arbitrario di macchine
- I task scambiano dati inviando o ricevendo messaggi
- Il trasferimento dei dati tipicamente richiede che I diversi processi eseguano operazioni per la cooperazione
- Ad esempio, a ogni operazione di invio (send) deve corrispondere a un'operazione di receive



Modelli di Programmazione Parallela
Message Passing

- Dal punto di vista del programmatore, le implementazioni del message passing tipicamente includono una libreria di subroutine che sono embedded nel codice sorgente
- Il programmatore è responsabile del parallelismo
- Diverse librerie per il message passing sono state disponibili fino dagli anni 80. Implementazioni differenti, bassa portabilità

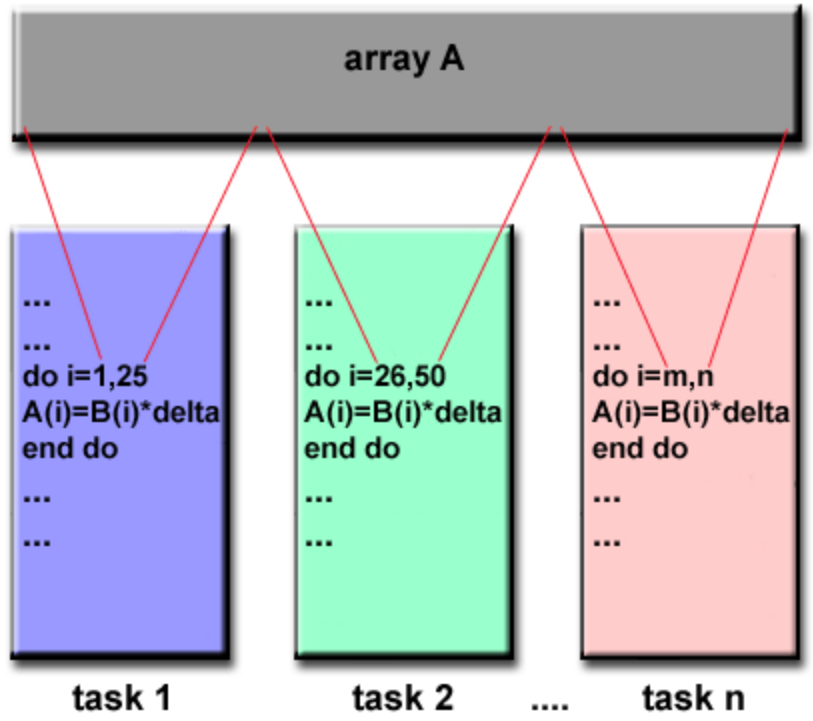
- Standardizzazione
  - Part 1 della Message Passing Interface (MPI) (1994)
  - Part 2 (MPI-2) (1996)

- MPI è lo standard di fatto per il message passing

- Nelle architetture a memoria condivisa, le implementazioni MPI non usano una rete di task ma usano la memoria condivisa per ragioni di performance

Modelli di Programmazione Parallela
Data Parallel

- Un insieme di task lavorano collettivamente sulla stessa struttura dati. Ogni task lavora su una diversa parte della struttura
- I task svolgono la stessa operazione sulla parte dei dati che gestiscono. Esempio, "somma 4 a tutti gli elementi di un array"
- Su un'architettura a memoria condivisa, tutti i task possono avere accesso ai dati attraverso la memoria globale
- Su architetture con memoria distribuita, la struttura dati è suddivisa e risiede sulla memoria locale di ogni task



Modelli di Programmazione Parallela
Data Parallel

- Implementato utilizzando costrutti
- Fortran 90 and 95 (F90, F95) ISO/ANSI (estensione del Fortran 77)
- High Performance Fortran (HPF): estensione del Fortran 90 per la programmazione (data parallel); direttive per specificare come i dati devono essere distribuiti
- L'implementazione con architetture di tipo distributed utilizzano il compilatore per convertire un programma aggiungendo chiamate alla libreria di message passing (tipicamente MPI) per distribuire i dati ai processori
- Il message passing avviene in maniera trasparente al programmatore

Modelli di Programmazione Parallela
Altri Approcci

- Ibridi: combinano due o più modelli di programmazione. Ad esempio, ol message passing model (MPI) può essere combinato con i thread POSIX oppure con un modello di shared memory (OpenMP)

- Single Program Multiple Data (SPMD): un solo programma eseguito da tutti i task simultaneamente

- Multiple Program Multiple Data (MPMD): più eseguibili che possono utilizzare dati differenti