Calcolo parallelo

- Ci sono problemi di fisica, come le previsioni meteorologiche o alcune applicazioni grafiche, che richiedono un'enorme potenza di calcolo

- Una sola CPU (o un solo core) , per quanto potenti, non sono sufficienti o richiederebbero tempi lunghissimi

- È necessario trovare un modo per fare eseguire un certo compito a più CPU

Grid

- Il mio problema si può dividere in tanti programmi, che posso fare girare in modo indipendente l'uno dall'altro

- Ogni programma può girare su di un computer diverso

- La struttura "Grid" dell'INFN permette di fare questa operazione facendo girare i programmi che risolvono il mio problema su computer sparsi in tutto il mondo

- È usato per analizzare l'enorme mole di dati degli esperimenti di oggi (LHC al CERN, per esempio)

Computer vettoriali

- Suppongo di dover fare la somma di due numeri: a me pare una singola operazione, ma il computer deve
  1. *Prendere il primo numero dalla memoria e copiarlo in un registro*
  2. *Fare lo stesso col secondo numero*
  3. *Effettuare la somma*
  4. *Copiare la somma in memoria*

- Una singola operazione corrisponde a molti passi diversi

- Per sommare diverse coppie di numeri posso fare il passo (1) della seconda somma mentre faccio il passo (2) per la prima coppia

- Questo accelera il calcolo, ma richiede un software dedicato per il particolare computer che stiamo usando

- Ci sono linguaggi di programmazione adatti a questo tipo di procedure (HPC)

MPI

- Faccio eseguire un singolo programma su più CPU (o più core di un processore multicore)
- Ogni programma è diviso in un certo numero di processi
- I processi devono poter comunicare tra di loro e devono potersi scambiare informazioni (Message Passing Interface)
- Ogni processo deve essere consapevole di quale sia il suo numero
- Ogni processo deve essere consapevole di quanti sono i processi in totale

Specifiche

- Ci sono funzioni e librerie studiate per svolgere questo compito

- Le specifiche sono standard, l'implementazione è a discrezione del produttore

- Le chiamate a funzione sono parte delle specifiche

- Esistono diverse versioni di MPI: attualmente c'è la 2, ma esistono anche 1.0 e 1.2, che contengono un numero più ristretto di funzioni

MPI con sei funzioni

- Occorrono funzioni per iniziare e terminare MPI

```
MPI_Init(int argc, char **argv);
MPI_Finalize();
```

- Mi serve conoscere il numero totale dei processi

```
MPI_Comm_size(MPI_COMMON_WORLD, &Nprocessi);
```

- Mi serve sapere qual è il numero di un particolare processo (da 0 e Nprocessi-1)

```
MPI_Comm_rank(MPI_COMMON_WORLD, &rank);
```

- Ora devo spedire/ricevere informazioni a/da gli altri processi; per spedire uso

```
MPI_Send(&buffer, Ndati, tipo_dato,
   destinazione, tag, MPI_COMMON_WORLD);
```

- Per ricevere

```
MPI_Recv(&buffer, Ndati, tipo_dato,
   sorgente, tag, MPI_COMMON_WORLD, status);
```

- Nel ricevere si può anche evitare di specificare da quale processo, purché si usi la macro `MPI_ANY_SOURCE` al posto di "sorgente"; "status" contiene, in questo caso informazioni sul processo sorgente

- Il tipo di dato è definito da macro `MPI`: esempi sono `MPI_INT` e `MPI_DOUBLE`

Deadlock

- È la bestia nera dei programmatori `MPI`
- il numero di "Send" deve corrispondere esattamente al numero di "Recv";
- l'ordine in cui le informazioni arrivano è importante: se due processi si scambiano messaggi, non è possibile mettere due istruzioni "Send" prima di due "Recv" perché ognuno dei due processi aspetterà conferma della ricezione del messaggio per procedere e ricevere il messaggio dell'altro;
- il numero e il tipo di dati inviati e ricevuti devono corrispondere;
- esistono comandi sincroni e asincroni;
- esistono anche comandi che consentono di inviare e ricevere messaggi allo stesso tempo.

Master e Slave

- Spesso conviene che un processo (di solito il numero zero) abbia un ruolo privilegiato. Esempi sono

  1. *calcolo di un integrale: i valori della funzione sono calcolati nei processi slave e poi passati al master che fa la somma*

  2. *calcolo di un coefficiente di diffusione: le coordinate di molte particelle sono calcolate nei processi slave, quindi passate al master che calcola la varianza*

- Il master conterrà molte più istruzioni di ricezione, mentre gli slave molte più di invio

Compilare ed eseguire con MPI

- Il compilatore richiede che siano definite alcune macro: per questo motivo si usa una istruzione apposita: `mpic++`

- Per avviare il programma c'è lo stesso problema, in più bisogna dire al computer quanti processi vogliamo usare. Per esempio, per eseguire il programma con 10 processi si da' il comando

```
mpirun -np 10 prog
```

- Attenzione! Se np=1 e c'è un processo master, questo potrebbe aspettare eternamente che un inesistente slave gli mandi dei dati!

Problemi paralleli e non

- Non tutti i problemi sono adatti al calcolo parallelo

- Se il problema si può spezzare in tanti problemi indipendenti, allora si può parallelizzare

- Esempio: calcolare il valore medio di N numeri pseudocasuali: posso creare k processi, ognuno dei quali fa la media su N/k numeri; alla fine si fa la media delle medie

- Se uno dei processi deve conoscere il risultato di un altro processo per essere eseguito, non si può rendere parallelo

- Esempio: relazioni di ricorrenza; però anche in questo caso si può fare qualcosa

$$x_{n+1} = a_n x_n + b_n = a_n(a_{n-1}x_{n-1} + b_{n-1}) + b_n$$
$$x_{n+1} = a_n a_{n-1} x_{n-1} + a_n b_{n-1} + b_n$$

- x_{n+1} può essere calcolato a partire da x_{n-1}, quindi posso calcolare indipendentemente due sequenze, corrispondenti agli indici pari e dispari

CUDA

- Acronimo per:
  Computer Unified Device Architecture

- è un'estensione del linguaggio C (C++) per programmare schede grafiche;

- queste sono oggi dotate di centinaia o migliaia di processori indipendenti, in grado di operare calcoli in parallelo

- alcune sono in grado di operare su numeri in doppia precisione

- la performance è ottimale quando eseguono operazioni semplici (non chiamate di funzioni come cos(x) e sin(x))

- i processi svolti parallelamente sono divisi in blocchi e/o in thread;

- i blocchi consentono un maggior numero di processi, ma non comunicano tra loro

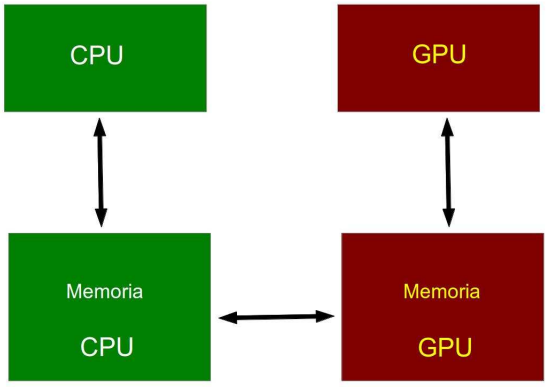
- thread possono essere alcune centinaia, con possibilità di condividere un'area di memoria;

- i processi sulla GPU (Graphic Processing Unit) hanno un timeout su molte schede grafiche

Passaggio di informazioni

- Le istruzioni che girano sulla CPU non hanno accesso alla memoria della GPU, e viceversa;

- l'unico modo di comunicare è trasferire oggetti dalla memoria della CPU a quella della GPU e viceversa



Istruzioni fondamentali

- Allocazione memoria sulla GPU

```
cudaMalloc((void**)& GPU_array, n.o bytes);
```

- copia da memoria CPU a GPU e viceversa

```
cudaMemcpy(GPU_Array *, CPU_array *, n.o bytes,
           cudaMemcpyHostToDevice);
cudaMemcpy(CPU_Array *, GPU_array *, n.o bytes,
           cudaMemcpyDeviceToHost);
```

- chiamata di una funzione f(x,y,z)

```
f<<<n.o blocchi, n.o thread>>>(x,y,z);
```

- una funzione eseguita sulla CPU si chiama *host*, una eseguita sulla GPU ma chiamata dalla CPU si chiama *global* (deve essere void) ed una chiamata ed eseguita sulla GPU si chiama *device*

Esercizi

1. *Calcolare un integrale con la regola del trapezio.*

2. *Calcolare il valore medio di* x^2 *con* $0 \leq x \leq 1$ *per una distribuzione uniforme, dividendo il calcolo tra vari processi*

3. *Risolvere un sistema di equazioni parallelizzando l'eliminazione gaussiana.*