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 sun 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)

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);
```

  e 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 devono scambiarsi dati, non è possibile mettere due istruzioni "Send" perché ognuno dei due processi resterà bloccato in attesa che il messaggio che ha inviato venga ricevuto; uno dei due processi deve ricevere prima il messaggio dell'altro, e poi spedire il proprio.
- il numero di chiamate a "Send" deve corrispondere esattamente al numero di "Recv"
- il numero e il tipo di dati inviati e ricevuti devono corrispondere.
- esistono comandi sincroni e asincroni.

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 coordnate di molte particelle sono calcolate nei processi slave, quindi passate al master che calcola la varianzai*

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

Compilare con MPI

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

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

```
mpirun -np 10 prog
```

  se vogliamo avviare il programma MPi prog con 10 processi

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

Calcoli con schede grafiche

- La scheda grafica agisce in modo indipendente dalla CPU per alleggerire il suo carico.

- Un forte impulso allo sviluppo delle schede grafiche è dato dallo sviluppo di videogiochi avidi di risorse.

- Solo da tre-quattro anni si sono sviluppate schede grafiche adatte a calcoli scientifici: sono un settore poco remunerativo.

- Esistono estensioni dei linguaggi C e FORTRAN per gestire le schede grafiche.

- NVIDIA ha sviluppato un proprio linguaggio che si chiama CUDA, la ATI uno diverso. OpenCL è un tentativo di unificarli, ma è più complicato. Nel seguito mi occuperò solo di CUDA.

Caratteristiche delle schede grafiche

- La scheda grafica (GPU) è in grado di fare meno operazioni della CPU.
- È fatto di un grande numero di processori ottimizzati.
- È molto adatta a processi che possano essere parallelizzati.
- Dispone di diverse memorie, alcune delle quali molto veloci.
- Il collo di bottiglia dei calcoli è spesso il trasferimento dati tra CPU e GPU.

Cosa si può e non si può fare

Cosa si può fare

- Si possono fare molti calcoli semplici in parallelo.
- Si può usare memoria ad accesso molto veloce.
- Si possono fare contemporaneamente calcoli sulla CPU e sulla GPU.

Cosa non si può fare

- Non si possono fare calcoli lunghi perché esiste un timeout.
- Non si possono trasferire velocemente dati tra la CPU e la GPU.
- Non si possono fare sulla GPU tutti i calcoli che si fanno sulla CPU.

Tipi di funzione

- Le funzioni si dividono in *host*, *global* e *device*
- *host* sono le normali funzioni che girano sulla CPU.
- *device* sono le funzioni chiamate dalla GPU che girano sulla GPU.
- la sintassi per chiamare funzioni *host* e *device* è la solita.
- *global* sono le funzioni chiamate dalla CPU ed eseguite dalla GPU.
- la sintassi è:
  funzione <<<griglia, blocco>>>(argomenti).
- se l'argomento è un vettore non viene copiato automaticamente in memoria.

Copiare per/da la memoria grafica

- Per copiare i vettori dalla CPU alla GPU bisogna allocare spazio su entrambe.

- Ci sono funzioni particolari (cudaMalloc) per allocare spazio sulla memoria grafica.

- ci sono funzioni dedicate alla copia di dati da/per la memoria grafica (cudaMemcpy)

- ci sono molto processi (thread) sulla GPU;

- ogni processo conosce il proprio numero tramite la variabile *threadIdx* e il proprio blocco tramite la variabile *blockIdx*. Queste variabili sono utilizzate per differenziare il comportamento di ogni thread