Corso di Algoritmi Avanzati—modulo 2

Quarto progetto di programmazione—Anno Accademico 2014/2015

Moreno Marzolla

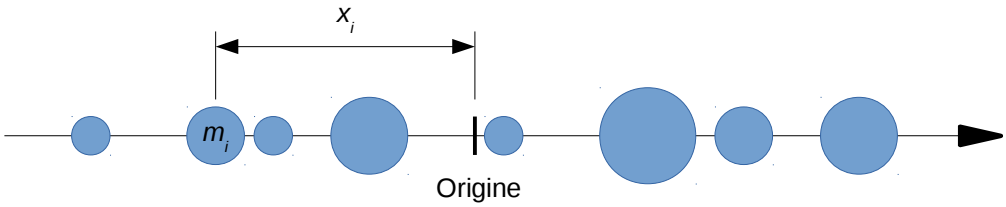
Versione 1.0 del 12/11/2015

Prima versione

Descrizione del progetto

Scopo del progetto è simulare la dimanica di n particelle in una dimensione (n-body problem) utilizzando MPI.

Consideriamo n punti di massa rispettivamente m_0, … m_{n-1} chilogrammi, disposti lungo una retta, aventi velocità iniziali note. Le masse si considerano puntiformi (quindi non hanno dimensione, a differenza di quanto mostrato nella figura che segue), e le posizioni sono misurate a partire da una origine fissata sulla retta.



Le masse si attraggono per effetto della forza di gravità; poiché sono puntiformi, non urtano mai l'una con l'altra, e possono quindi "passarsi attraverso" muovendosi lungo la retta.

La dinamica del sistema può essere calcolata in modo iterativo, integrando le equazioni del moto con la regola di Eulero del primo ordine. Indichiamo con x^k_j e v^k_j la posizione e la velocità del punto j-esimo al passo k. Le posizioni e le velocità sono scalari; una velocità positiva indica che il punto si muove verso destra, mentre una velocità negativa indica che il punto si muove verso sinistra.

1. Si calcola la forza F^k_j che agisce sulla particella j al passo k come segue:

$$F^k_j := \sum_{i \neq j} \frac{G m_i m_j}{d^2_{ij}} n_{ij}$$

   ove G è la costante di gravitazione universale ($G = 6{,}674 \times 10^{-11}$ N m^2 kg^{-2}), d_{ij} è la distanza, al passo k, tra le particelle i e j ($d_{ij} := |x^k_i - x^k_j|$) e n_{ij} è il vettore unitario normale che, sempre al passo k, punta dalla particella j verso la particella i ($n_{ij} := (x^k_i - x^k_j) / d_{ij}$)

2. Si calcola l'accelerazione a^k_j che agisce sulla particella j al passo k come segue:

$$a^k_j := F^k_j / m_j$$

3. Si calcola la velocità v^{k+1}_j della particella j al passo $k+1$ come:

$$v^{k+1}_j := v^k_j + a^k_j \Delta t$$

4. Si calcola la posizione x^{k+1}_j della particella j al passo $k+1$ come:

$$x^{k+1}_j := x^k_j + v^{k+1}_j \Delta t$$

Pertanto, partendo da posizioni x^0_j e velocità v^0_j iniziali note, possiamo iterare i passi 1—4 per determinare la posizione e velocità delle particelle ai passi 1, 2, … T. Osserviamo che il passo 1 è

l'unico che effettua una computazione globale (in cui cioè ogni calcolo della forza richiede le masse e le posizioni di tutti i punti).

Il numero di punti, le masse, posizioni e velocità iniziali devono essere lette da un file di testo di nome `in.txt` avente la struttura seguente:

```
n
m0      x0      v0
m1      x1      v1
…
mn-1    xn-1    vn-1
```

dove n (intero positivo) rappresenta il numero di punti, m_j, x_j e v_j sono rispettivamente la massa, posizione iniziale e velocità iniziale del punto j-esimo (sono tutti reali di tipo `double`; le masse sono strettamente positive, mentre posizioni e velocità possono anche essere negative).

Il programma MPI deve accettare i seguenti parametri sulla riga di comando:

- il numero T di passi dell'algoritmo di Eulero da eseguire;
- il valore dt

Il programma, dopo aver letto i parametri delle masse dal file `in.txt`, eseguirà T passi dello schema di Eulero con intervallo di integrazione dt, e salverà su un nuovo file `out.txt` le masse, posizioni e velocità degli n punti dopo l'ultimo passo, con lo stesso formato del file di input (deve essere possibile dare in input al programma il file di output, dopo avergli cambiato nome).

Suggerimenti

Sulla pagina web (http://www.moreno.marzolla.name/teaching/2014-2015/algoritmi-avanzati/) è disponibile una versione sequenziale dell'algoritmo (siete pregati di segnalarmi eventuali errori). Chi lo desidera può basare la versione MPI sulla versione sequenziale.

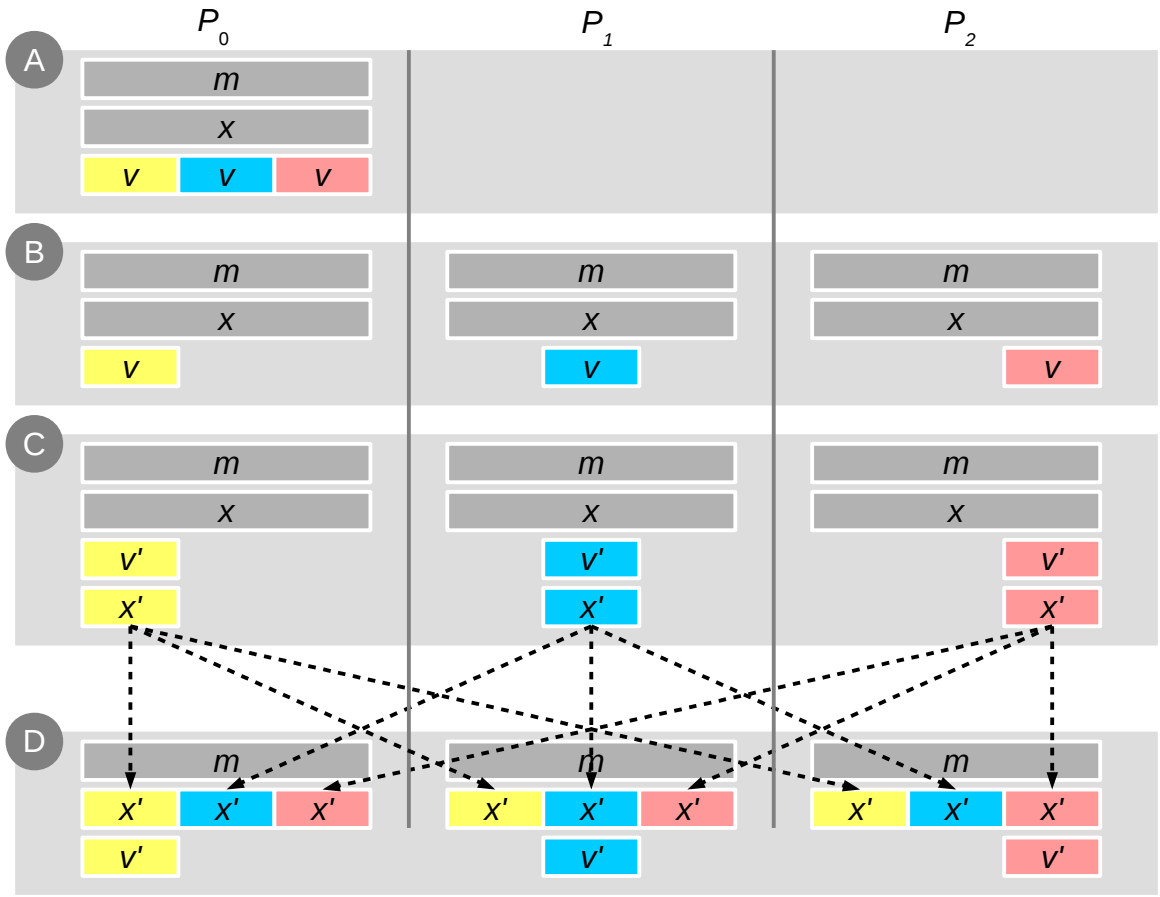
Suggerisco di adottare la strategia di parallelizzazione che consiste nel distribuire tra i processi MPI il calcolo delle forze, suddividendo i punti tra i processi MPI. Lo schema di funzionamento dell'algoritmo parallelo è mostrato nella figura alla pagina seguente.

A) Il processo master legge i dati di input

B) Il master distribuisce una copia dell'intero array m (masse) e x (posizioni) a tutti i processi; invece, l'array v (velocità) viene suddiviso equamente tra tutti i processi

C) Ciascun processo utilizza le equazioni 1—4 illustrate in precedenza per calcolare le nuove posizioni (x') e velocità (v') dei punti a lui assegnati.

D) Nell'ultima fase, le nuove posizioni locali x' sono distribuite a tutti i processi, utilizzando la funzione `MPI_Allgather`, in modo che ciascun processo possegga localmente una copia dell'intero array delle nuove posizioni x'. A questo punto i valori di x' e v' prendono il posto di x e v, e l'algoritmo riprende dal punto B.



Per semplificare l'implementazione **è consentito imporre che il numero di punti *n* sia un multiplo esatto del numero di processi MPI**. Questo semplifica la distribuzione dell'array *v* e il successivo assemblaggio dei frammenti di *x*'.

Modalità di svolgimento del progetto

- Il progetto deve essere svolto individualmente. Non è consentito condividere codice o relazione con altri studenti, né utilizzare software disponibile in rete (fa eccezione il codice sequenziale di esempio, e il codice dei programmi mostrati a lezione, che sono riutilizzabili a piacimento). Non è consentito discutere il progetto con altri.
- Il progetto deve essere realizzato in un singolo file sorgente chiamato `nbody-1d.c`. Il sorgente deve essere adeguatamente commentato. All'inizio del file sorgente deve essere incluso un commento che riporta cognome, nome e numero di matricola dell'autore/autrice, nonché il comando da usare per la compilazione (vedi punto successivo).
- Il progetto deve essere implementato in linguaggio C come applicazione a riga di comando, facendo uso delle librerie MPI. Il progetto deve essere compilabile ed eseguibile senza errori su una delle macchine Linux dei laboratori studenti (sistema operativo Debian GNU/Linux 7.8), oppure sull'immagine dei laboratori virtuali (http://www.virtlab.unibo.it/index.html) mediante il comando

  ```
  mpicc -Wall nbody-1d.c -o nbody-1d [eventuali flag di compilazione]
  ```

  Indicare nel commento iniziale del sorgente e/o nella relazione la riga di comando corretta per la compilazione nel caso siano richiesti altri flag, ad esempio nel caso in cui sia necessario includere librerie di sistema. Nota: sulle macchine del laboratorio è installato

OpenMPI (pacchetti `openmpi-bin`, `libopenmpi-dev` ed `openmpi-doc` per la documentazione).

- Per verificarne la correttezza, il programma verrà eseguito mediante il comando

  `mpirun -n` *`proc`* `./nbody-1d` *`nsteps dt`*

  dove *proc* indica il numero di processi MPI da creare. Tutte le istanze saranno lanciate localmente, in modo da evitare ogni problema con la rete. Sarà presente un file di input corretto (il programma sequenziale di esempio è in grado di generarne uno con valori pseudocasuali; controllare il sorgente per informazioni).
- Si può assumere che il numero *n* di masse sia sempre (molto) maggiore del numero di processi MPI. **Si può assumere che il numero *n* di punti sia un multiplo esatto del numero di processi MPI**. Si può infine assumere che sia sempre possibile mantenere nella memoria locale di ciascun processo MPI tutti i dati necessari.
- Assieme al sorgente deve essere consegnare una **relazione** in formato PDF in un file chiamato `relazione.pdf`. La relazione non deve superare la lunghezza di **cinque facciate** in formato A4 (font non inferiore a 10 punti). **Nel computo delle cinque facciate rientrano TUTTE le facciate del documento consegnato** (pagina del titolo, eventuale indice o altro); suggerisco di non sprecare spazio dedicando una facciata al titolo o all'indice, che in un documento così corto sarebbero comunque superflui. Indicare il proprio cognome, nome, e numero di matricola all'inizio della relazione.
- La relazione deve descrivere in modo sintetico l'implementazione con particolare riguardo alla strategia di parallelizzazione adottata e gli eventuali limiti e/o potenzialità di tale strategia.
- La relazione **deve obbligatoriamente includere una sintetica analisi sperimentale della scalabilità dell'algoritmo realizzato, mostrando i grafici di speedup ed efficienza**. Non è richiesto che l'analisi di scalabilità venga effettuata sulle macchine del laboratorio. È possibile effettuare i test lanciando tutte le istanze localmente (anche in numero superiore al numero di core disponibili); ovviamente non terrò conto dei tempi di esecuzione in termini assoluti, ma solo del fatto che l'analisi di speedup ed efficienza sia stata svolta correttamente.

Scadenza e modalità di consegna

E' necessario consegnare:

1. Il file sorgente del programma realizzato;
2. La relazione in formato PDF.

Il progetto e la relazione devono essere consegnati entro le ore **12:00 (mezzogiorno)** di lunedì **11 gennaio 2016**.

Ricordo che è necessario ottenere una valutazione positiva del progetto per poter partecipare alla prova scritta del modulo 1. La valutazione verrà comunicata via mail, e se positiva resterà valida esclusivamente per la sessione d'esami di gennaio/febbraio 2016.

La consegna deve avvenire inviando una mail dal proprio indirizzo istituzionale @studio.unibo.it a moreno.marzolla@unibo.it con subject

Consegna Progetto Algoritmi Avanzati 2014/2015

Nella mail vanno indicati cognome, nome, numero di matricola del mittente. Alla mail deve essere allegato un archivio in formato `.zip` oppure `.tar.gz` contenente il sorgente e la relazione in formato PDF; chi lo desidera può includere altri files, ad esempio un `Makefile` per la compilazione

(non obbligatorio). L'allegato sarà denominato con il cognome dell'autore (es., `Rossi.zip` oppure `Rossi.tar.gz`), e conterrà una directory con lo stesso nome (es., `Rossi/`) contenente a sua volta il sorgente e la relazione. Riceverete una mail di conferma entro pochi giorni.

Nota per gli utenti Apple: gli archivi .zip prodotti sotto MacOSX spesso includono una directory `.DS_Store/`. L'archivio che viene consegnato **deve essere privo** di tale directory.

Valutazione dei progetti

Per ottenere una valutazione positiva è necessario che il programma compili sulle macchine Linux del laboratorio studenti e funzioni correttamente secondo le specifiche indicate in questo documento. Ulteriori elementi di cui si terrà conto:

- Correttezza della strategia di parallelizzazione adottata;
- Generalità della soluzione implementata;
- Qualità della relazione, in termini di chiarezza, correttezza e presentazione del contenuto;
- Qualità del codice; verrà penalizzato il ricorso a micro-ottimizzazioni inutili, nonché codice ridondante o eccessivamente complesso. Verranno altresì penalizzati errori di programmazione (come ad esempio, accessi out-of-bound, memory leak, uso errato di variabili non inizializzate, eccetera).