

BouncingBalls

1° Progetto – Programmazione Concorrente e Distribuita



Konrad Gomulka - matricola

Luca Salvigni - matricola

Manuele Pasini - matricola

Sommario

[1. Analisi del problema 3](#_Toc37422741)

[2. Descrizione soluzione 4](#_Toc37422742)

[3. Test Performance 5](#_Toc37422743)

# Analisi del problema

Il progetto consiste nella realizzazione di un’implementazione concorrente dell’algoritmo “bouncing balls”, dove n palline vengono create e fatte muovere all’interno di uno spazio limitato per un certo numero di volte k.

Ad ogni spostamento unitario possono verificarsi due tipi di collisione:

* Tra due palline
* Tra una pallina e un bordo dello spazio

Inizialmente i thread (il cui numero è uguale al numero di core della macchina su cui l’algoritmo è in esecuzione +1) si suddividono il numero di palline da gestire in modo equo.

Successivamente ogni thread calcola lo spostamento unitario delle palline a lui affidate e attende che tutti i thread abbiano finito di calcolare la nuova posizione delle proprie palline (tramite un algoritmo fornito).

A questo punto si entra nella fase di gestione delle collisioni che si possono suddividere in:

* Collisioni tra una pallina ed un bordo che possono essere risolte senza concorrenza, in quanto il bordo è un’entità statica che non interagisce con l’ambiente, tuttavia, dopo aver risolto una collisione con un bordo e spostata  nuovamente la pallina sarà necessario aggiornare la lista globale delle palline condivisa tra i vari thread.
* Collisioni tra due palline che devono essere risolte in modo atomico e concorrente senza causare situazioni di deadlock tra i vari thread, in quanto più thread nello stesso istante potrebbero individuare la stessa collisione.

# Descrizione soluzione

//METTIAMO GRAFICO E SCRIVIAMO UN Pò DIO LUPO

# Test Performance

Le misurazioni sono state svolte utilizzando un numero di palline N pari a 100, 1000 e 5000 ed un numero di iterazioni Nsteps pari a 500, 1000, 5000 su tre macchine differenti, valutando il comportamento della soluzione utilizzando sia il numero ottimale teorico di thread, sia considerando prove diverse con un numero variabile di thread per verificarne la scalabilità.

//METTIAMO I NOSTRI TEST