

BouncingBalls

1° Progetto – Programmazione Concorrente e Distribuita



Konrad Gómułka - 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”, il cui task principale consiste nella gestione delle collisioni tra n palline generate casualmente su un piano e messe in movimento per un certo numero di iterazioni k.

Ad ogni spostamento unitario possono verificarsi due tipi di collisione:

* Tra due palline
* Tra una pallina e un bordo del piano

Inizialmente i thread (il cui numero è uguale al numero di core della CPU 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

La soluzione implementata consiste nella suddivisione delle varie palline in un numero di thread pari ai core della CPU + 1, nel caso vi sia un resto alla divisione le palline in eccesso vengono affidate al primo thread.

Una volta fatto partire ogni thread avvia un ciclo nel quale vengono controllate tutte le palline assegnate in cerca di collisioni. Per ogni pallina assegnata al thread vengono controllate tutte le successive:

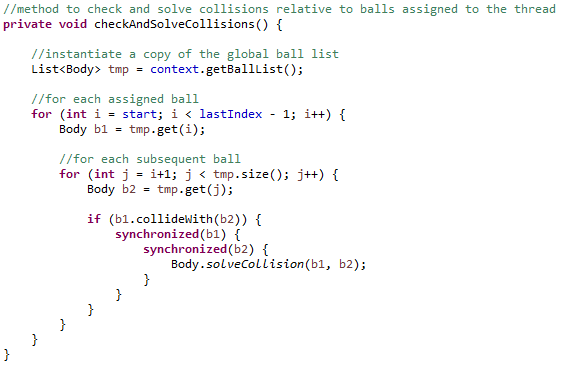


Figura 1 Metodo di individuazione e risoluzione delle collisioni

L’accesso alla lista globale dato che è in sola lettura non è gestito mediante semafori o monitor, mentre una volta rilevata una collisione dato che è necessaria un’operazione di scrittura il thread esegue la lock dell’oggetto tramite il metodo synchronized (che implementa un monitor). Dato che ogni thread controlla solo le palline successive non vi è deadlock in quanto l’ultimo thread potrà accedere unicamente alle proprie palline senza interferire con gli altri.

La sincronizzazione tra i thread viene gestita mediante una CyclicBarrier pertanto ogni thread una volta finite le proprie operazioni si metta in attesa degli altri.

Una volta risolte le collisioni tra palline viene fatto il controllo delle collisioni con i boundary per poi procedere con l’aggiornamento delle posizioni delle palline (ogni thread aggiornerà le proprie).

# 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.

I test sono stati effettuati 4 volte per ogni valore, ricavandone quindi la media espressa in ms.

**TEST AMD Ryzen 2600 Six-Core processor 4GHz**

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 100 palline | |  |  | 1000 palline | |  |  | 5000 palline | |  |  |
| n. Thread | 500 step | 1000 step | 5000 step | n. Thread | 500 step | 1000 step | 5000 step | n. Thread | 500 step | 1000 step | 5000 step |
| seq | 63 | 70 | 121 | seq | 636 | 1875 | 8664 | seq | 13809 | 27775 | 136454 |
| 1 | 48 | 57 | 162 | 1 | 794 | 1531 | 7255 | 1 | 18859 | 37111 | 186774 |
| 2 | 53 | 73 | 210 | 2 | 631 | 1194 | 5725 | 2 | 14272 | 28726 | 138679 |
| 4 | 86 | 120 | 424 | 4 | 475 | 838 | 3928 | 4 | 9168 | 18087 | 88290 |
| 8 | 120 | 204 | 827 | 8 | 452 | 811 | 3409 | 8 | 7111 | 13935 | 65275 |
| 12 | 154 | 270 | 1242 | 12 | 426 | 739 | 3404 | 12 | 5530 | 10739 | 52657 |
| 13 | 173 | 303 | 1353 | 13 | 399 | 758 | 3516 | 13 | 5149 | 10109 | 49238 |
| 16 | 203 | 362 | 1638 | 16 | 422 | 784 | 3634 | 16 | 4695 | 9180 | 44893 |
| 20 | 241 | 462 | 2094 | 20 | 469 | 834 | 3915 | 20 | 4958 | 9567 | 46905 |

Tabella 1 Risultati dei test con processore AMD Ryzen 2600

**Con i relativi SpeedUp calcolati:**

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 100 palline | |  |  | 1000 palline | |  |  | 5000 palline | |  |  |
| n. Thread | 500 step | 1000 step | 5000 step | n. Thread | 500 step | 1000 step | 5000 step | n. Thread | 500 step | 1000 step | 5000 step |
| 1 | 1,3125 | 1,22807 | 0,746914 | 1 | 0,801008 | 1,22469 | 1,194211 | 1 | 0,732223 | 0,74843 | 0,730583 |
| 2 | 1,188679 | 0,958904 | 0,57619 | 2 | 1,007924 | 1,570352 | 1,513362 | 2 | 0,967559 | 0,966894 | 0,983956 |
| 4 | 0,732558 | 0,583333 | 0,285377 | 4 | 1,338947 | 2,23747 | 2,205703 | 4 | 1,506217 | 1,535633 | 1,54552 |
| 8 | 0,525 | 0,343137 | 0,146312 | 8 | 1,40708 | 2,311961 | 2,541508 | 8 | 1,941921 | 1,993183 | 2,090448 |
| 12 | 0,409091 | 0,259259 | 0,097424 | 12 | 1,492958 | 2,537212 | 2,545241 | 12 | 2,497107 | 2,586367 | 2,591374 |
| 13 | 0,364162 | 0,231023 | 0,089431 | 13 | 1,593985 | 2,473615 | 2,464164 | 13 | 2,68188 | 2,747552 | 2,771315 |
| 16 | 0,310345 | 0,19337 | 0,073871 | 16 | 1,507109 | 2,391582 | 2,38415 | 16 | 2,941214 | 3,025599 | 3,039538 |
| 20 | 0,261411 | 0,151515 | 0,057784 | 20 | 1,356077 | 2,248201 | 2,213027 | 20 | 2,785196 | 2,903209 | 2,909157 |

Tabella 2 Tabella degli SpeedUp calcolati mediante la formula (Vecchio tempo di esecuzione / Nuovo tempo di esecuzione)

**Come si può notare anche dai seguenti grafici, il picco degli SpeedUp è stato raggiunto con 5000 palline e un numero indifferente di step:**

Figura 2 Rappresentazione grafica degli SpeedUp relativi a 100 palline

Con un numero basso di palline possiamo notare che lo SpeedUp tende a diminuire incrementando il numero di Worker.

Figura 3 Rappresentazione grafica degli SpeedUp relativi a 1000 palline

Figura 4 Rappresentazione grafica degli SpeedUp relativi a 5000 palline