Progetto di High Performance Computing 2016/2017

Luka Micheletti, matr. 0000723450
1/8/2017

Introduzione

Il progetto realizzato consiste in un'implementazione del modello BML in versione parallela tramite i framework di calcolo parallelo OpenMP, che sfrutta il parallelismo multiprocessore a memoria condivisa, e CUDA, che sfrutta il parallelismo massivo delle GPU nVidia.

L'automa BML consiste in due passi di computazione per ogni iterazione, uno orizzontale e uno verticale, che muovono le diverse celle della matrice, rispettivamente da sinistra verso destra e dall'alto verso il basso. Il programma seriale si limita a leggere i valori in una matrice e a copiarli nell'altra, trasformati, uno per uno.

I due programmi prodotti implementano l'automa in modo differente, essendo i metodi di parallelizzazione dei due framework molto diversi tra loro; inoltre, vista la necessità di hardware specifico per entrambe le versioni, sono stati testati su macchine diverse. La principale differenza tra le due versioni parallele del programma sta proprio nell'implementazione dei due step dell'algoritmo.

Implementazione del modello BML con OpenMP

I due step di computazione (orizzontale e verticale) sono composti da due cicli annidati, che scorrono la matrice corrente e, per ogni elemento calcolano lo step successivo, scrivendolo nella seconda matrice.

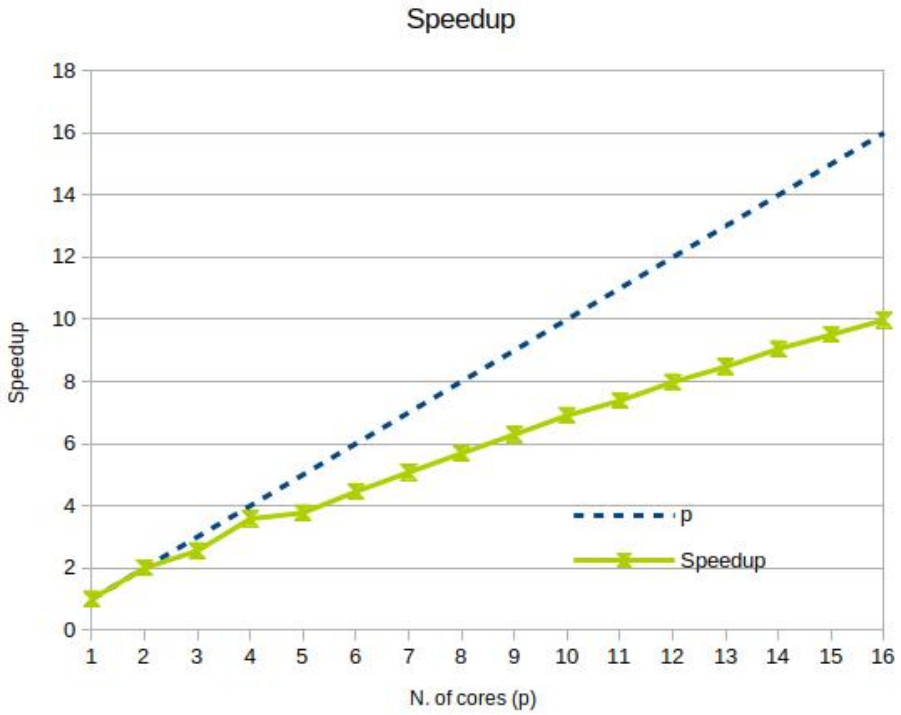
Per parallelizzare la computazione è stata sfruttata la direttiva #pragma omp parallel for, che suddivide i cicli tra i thread del pool in uso.

È stata sfruttata anche la direttiva collapse insieme a parallel for per comporre i due cicli annidati e dividerli entrambi tra i thread del pool a disposizione.

Il programma è stato testato sulla macchina di laboratorio disi-hpc.csr.unibo.it, che monta una CPU AMD Opteron 6376, con 16 core fisici e senza hyperthreading.

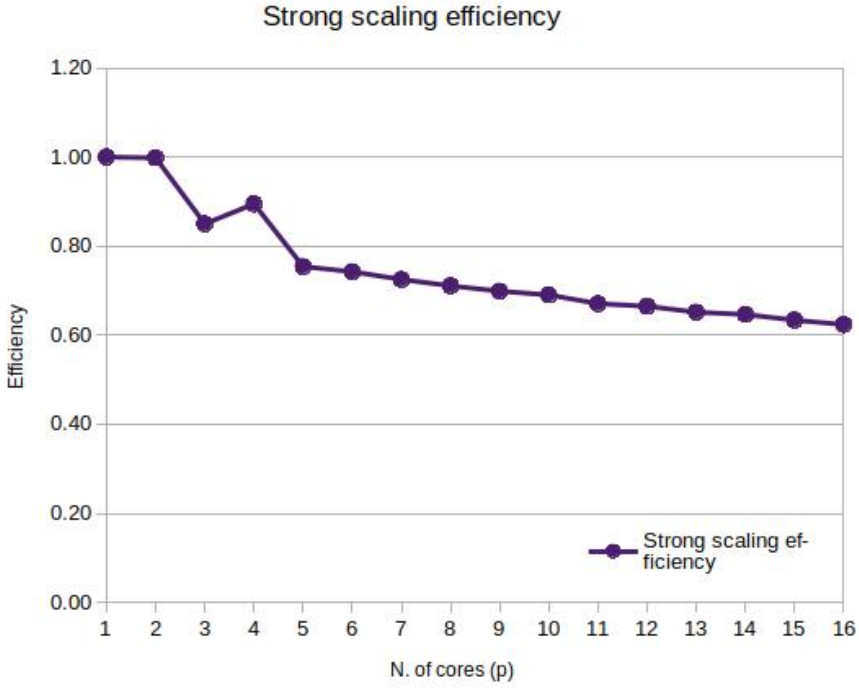
Sono state effettuate misurazioni sul tempo di esecuzione per misurare lo speedup e la strong scaling efficiency al variare del numero di core utilizzati.

L'andamento teorico dello speedup è pari al numero di core utilizzati per ogni test; tale andamento non è fisicamente ottenibile, ma l'obiettivo dell'ottimizzazione è avvicinare la curva reale a quella teorica fruttando le direttive messe a disposizione dall'ambiente di sviluppo. OpenMP, in particolare offre metodi di parallelizzazione limitati ad architetture a memoria condivisa, adatti alla macchina usata per le misurazioni, ma non applicabili ad architetture per parallelizzazione più elevata, che sono, solitamente basate su memoria distribuita.



I valori visualizzati nel grafico sono risultati dalla media di cinque misurazioni per ogni numero di core utilizzati, sulla quale viene calcolato lo speedup rispetto all'esecuzione su un solo core. Il calcolo corretto dello speedup andrebbe effettuato rispetto all'algoritmo seriale, tuttavia, per semplicità si è scelto di calcolarlo rispetto all'algoritmo parallelo eseguito su un solo core (che, tra l'altro è meno performante di quello seriale ad hoc).

Dal grafico si nota che lo speedup reale, come prevedibile, non si avvicina a quello teorico (pari al numero di core usati), soprattutto nei test finali. Si nota, inoltre che lo speedup subisce un rallentamento nel passaggio da due a tre e da quattro a cinque core e, in generale nel passaggio da un numero di core pari a uno dispari, anche se il fenomeno è meno accentuato quando il numero di core usati si avvicina a quello disponibile nel processore.



Il grafico sopra mostra l'andamento della strong scaling efficiency al variare del numero di core utilizzati. Si nota che, come per lo speedup, il passaggio da due a tre e da quattro a cinque core è critico in termini di efficienza, in quanto si misurano cali notevoli; il fenomeno è presente ogni volta che si passa da un numero di core pari a uno dispari, ma è particolarmente visibile nei casi già citati. Un dato difficilmente apprezzabile dal grafico dello speedup è il miglioramento nel passaggio da tre a quattro core, unico step che produce un aumento dell'efficienza.

I dati misurati indicano che, probabilmente il sistema operativo e, in particolare lo scheduler sono ottimizzati per gestire un numero di thread pari contemporaneamente (più probabilmente potenze di due), il che penalizza un algoritmo che ne sfrutta un numero dispari.

L'algoritmo usato, che sfrutta due matrici per svolgere i passi di computazione, permette di mantenere condivise tra i thread le variabili usate, in quanto ogni passo consiste in una lettura da una matrice e una scrittura nell'altra, evitando eventuali corse critiche. Inoltre la direttiva omp parallel for suddivide il ciclo (collassato dai due cicli annidati) tra i thread a disposizione, che hanno, quindi accesso solo alla propria parte delle due matrici.

Una misurazione interessante sarebbe quella del tempo di esecuzione su una CPU con hyperthreading, che mostrerebbe sicuramente un calo nell'aumento di prestazioni in corrispondenza del passaggio dall'uso dei soli core fisici a quelli virtuali.

Implementazione del modello BML con CUDA

Per la versione CUDA si è scelto di ottimizzare il lavoro della gpu attraverso l'uso di uno strato di ghost cell intorno alle matrici: le due matrici hanno dimensione (N + 2) * (N + 2), invece che N * N e, prima di ogni passo (orizzontale e verticale), le ghost cell di ogni matrice vengono inizializzate con i valori dell'altra estremità. Essendo il dominio ciclico, ad ogni passo andrebbero letti i valori all'estremità opposta della matrice; l'uso delle ghost cell facilita questa operazione copiando preventivamente i valori delle celle opposte in celle adiacenti.

Per copiare i valori nelle ghost cell sono usati dei kernel, poiché le matrici su cui la GPU lavora sono salvate nella memoria del device.

Il programma può essere ulteriormente ottimizzato sfruttando la memoria condivisa messa a disposizione dal device, il che, probabilmente garantirebbe un aumento di prestazioni notevole, vista la ridotta latenza nell'accesso a tale memoria. Nonostante questa ottimizzazione sia valida, la memoria shared del device è condivisa solo tra i thread di uno stesso blocco, il che richiede cautela nella sua implementazione.

Durante la stesura del programma sono sorti due problemi principali: la gestione di matrici di input di dimensione non multipla della dimensione dei blocchi di thread nel device (impostata a 16 per non superare il numero massimo di thread per blocco consentito dalla GPU in uso e favorire l'esecuzione contemporanea dei thread, organizzati in warp di 32) e il riempimento delle ghost cell in entrambe le matrici.

Il primo problema è stato risolto semplicemente arrotondando per eccesso le dimensioni della griglia di blocchi necessaria per l'esecuzione dell'algoritmo: stabilita la dimensione dei blocchi (BLKSIZE) al massimo possibile per la GPU in uso, la dimensione della griglia è stata impostata a N / BLKSIZE, arrotondando il calcolo per eccesso. A questo punto il numero effettivo di thread a disposizione è superiore alla dimensione della matrice, perciò è necessario che ogni thread controlli che la sua posizione sia interna alla matrice e, in tal caso computare. Questo metodo, visto a lezione, implica uno spreco di risorse, ma è estremamente rapido nell'implementazione, il che lo rende molto comodo.

Il secondo problema è stato facilmente risolto eseguendo la copia dei valori delle celle prima di ogni

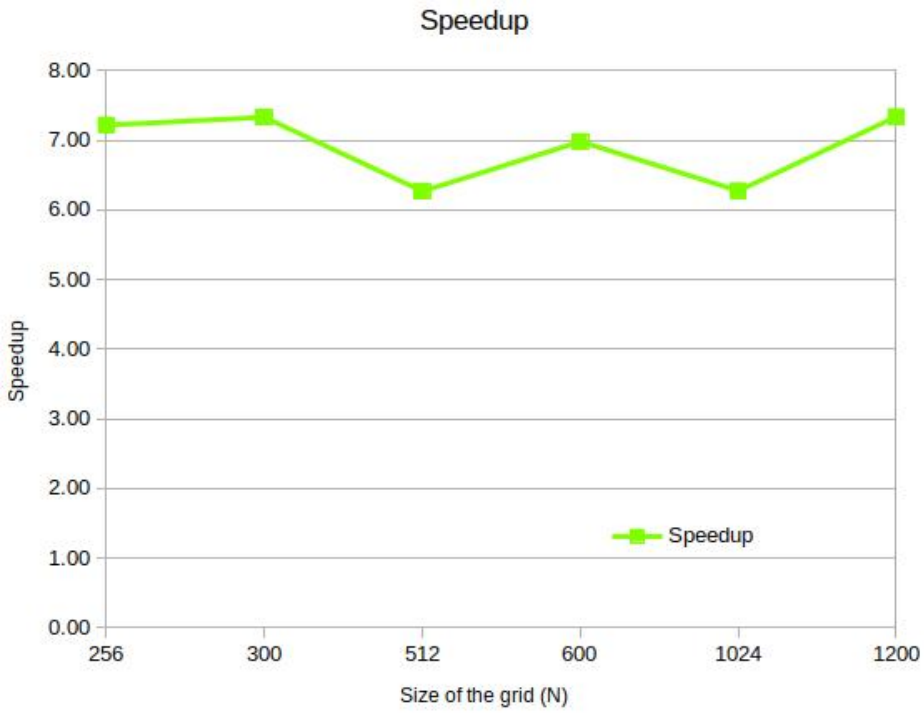
passo di computazione: prima del passo orizzontale vengono inizializzate le ghost cell della prima matrice e, dopo tale passo vengono inizializzate quelle della seconda, così che il passo verticale possa essere eseguito senza problemi.

Sono state effettuate misurazioni del tempo di esecuzione per diverse configurazioni dell'input, in particolare è stato fatto un calcolo del tempo medio di esecuzione al variare della dimensione della matrice da trasformare, mantenendo costanti il numero di iterazioni a 256 e la densità di veicoli a 0.2: i valori presi in considerazione sono 256, 300, 512, 600, 1024 e 1200.

La scelta dei valori è pensata per testare le performance dell'algoritmo anche con dimensioni di input non multiple della dimensione dei blocchi.

Lo speedup è calcolato rispetto a una versione seriale dell'algoritmo, anch'essa testata con le stesse dimensioni di input.

I tempi sono stati misurati eseguendo il programma sulla macchina di laboratorio disi-hpc-cuda.csr.unibo.it, che monta una CPU Intel Core 2 Duo e una scheda video nVidia Tesla c870, con 128 core e compute capability 1.0.



Dal grafico si nota come lo speedup rimanga sempre relativamente alto, ma con oscillazioni inaspettate: raggiunge valori più alti in corrispondenza dei test relativi a input di dimensioni non multiple del BLKSIZE. Una possibile spiegazione è che la versione seriale soffra più di quella parallela nei casi in cui la dimensione delle matrici non è una potenza di due.

Il fenomeno non è stato approfondito, ma potrebbe anche essere dovuto a una casualità, per esempio al tipo di scheduling o a inconsistenze nelle condizioni dell'ambiente di esecuzione tra le varie misurazioni.

Il calcolo della strong scaling efficiency nella versione CUDA risulta impreciso: nonostante il numero di core fisici della GPU rimanga fisso, la decomposizione in thread risulta solo virtuale e il numero di thread messi in gioco da CUDA varia a seconda della dimensione dell'input ed è, comunque talmente elevato che l'efficienza rasenta lo zero. Calcolando l'efficienza sul numero di core fisici, il calcolo perde di significato, in quanto, ovviamente l'efficienza è inversamente proporzionale alla dimensione dell'input.

Conclusioni

Rispetto al programma seriale (a quello parallelo su un solo core, nel caso di OpenMP), lo speedup di entrambe le versioni è risultato notevole; l'elevato numero di core della macchina disi-hpc.csr.unibo.it ha garantito uno speedup di circa 10 (10 volte più veloce) con il programma OpenMP, usando tutti i core a disposizione, mentre la versione CUDA ha prodotto uno speedup di circa 7. Mentre la versione OpenMP ha ricalcato le previsioni sull'andamento dello speedup al variare del numero di core utilizzati, la versione CUDA ha prodotto un'andamento imprevisto: dimensioni dell'input non multiple del BLKSIZE o, comunque di una potenza di due, hanno generato uno speedup maggiore rispetto a dimensioni "standard". Dimensioni di input non multiple del BLKSIZE causano una generazione di thread sovrabbondante, perciò i thread in eccesso, semplicemente non computano; questo fenomeno risulta in un effettivo spreco di risorse (seppur minimo), che rende inatteso il risultato ottenuto. In ogni caso i test effettuati non possono essere perfettamente accurati, in quanto le condizioni dell'ambiente non sono direttamente controllabili, né conoscibili.

Il modello BML presenta una particolarità legata alla densità di veicoli nella matrice: superata una certa densità di soglia, il simulatore tende a creare ingorghi indistricabili, generando una situazione statica (tutti i veicoli rimangono immobili). Questa particolarità, tuttavia non incide sul tempo di esecuzione, in quanto, a prescindere dal movimento dei veicoli, la divisione in due step di computazione rende necessaria la scrittura di ogni cella sulla seconda matrice; in altre parole non è prevista un'ottimizzazione nel caso di veicoli immobili. Ciò permette di variare la densità di veicoli senza alterare il tempo di esecuzione del programma.