# Progetto di High Performance Computing 2016/2017

XiangXiang Ma, matr. 0000718971

05/11/2017

## Introduzione

Il progetto è stato realizzato usando OpenMP e CUDA, cercando in entrambi i casi di ottimizzare le performance. In particolare, si è adottata una metodologia di lavoro incrementale caratterizzata da più versioni: una prima versione "semplice", ma funzionante; e una seconda versione più complessa, allo scopo di migliorare i tempi di esecuzione.

Per la valutazione delle performance è stato utilizzato il seguente hardware:

- Per OpenMP:
    - AMD Ryzen 7 1700 8C/16T @ 3.0 Ghz, DDR4 @ 2800Mhz
    - AMD Opteron 6376 16C/16T (lab)
    - Intel i7 4700HQ 4C/8T @ 2.4 Ghz, DDR3 @ 1600Mhz
- Per CUDA:
    - GTX 1080 con 2560 cuda cores
    - GTX 1070 con 1920 cuda cores
    - GTX 850M con 640 cuda cores
    - Tesla C870 con 128 cuda cores (lab)

## Implementazione del modello BML con OpenMP

La parallelizzazione con OpenMP è stato ottenuto assegnando l'aggiornamento delle celle della matrice(sotto forma di array) del modello a un pool di thread creato da OpenMP, invece di far eseguire il tutto da un singolo processore.

Una prima versione prevedeva la creazione del pool ad ogni step tramite la direttiva *omp parallel for*, creando di conseguenza un possibile overhead. Successivamente si è pensato di riciclare i thread del pool tramite il nesting delle direttive *omp for* all'interno di un *omp parallel.*

In termini di scalabilità si è notato subito uno speedup (figura 1 e 2) molto lineare all'aumentare dei thread se mappati ad un core fisico, inoltre grazie all'uso di Hyper-Threading (HT) e Simultaneous multithreading (SMT) si ottiene un ulteriore aumento delle performance intorno al 20%.

Di seguito i grafici di speedup, wall-clock time e di efficienza:

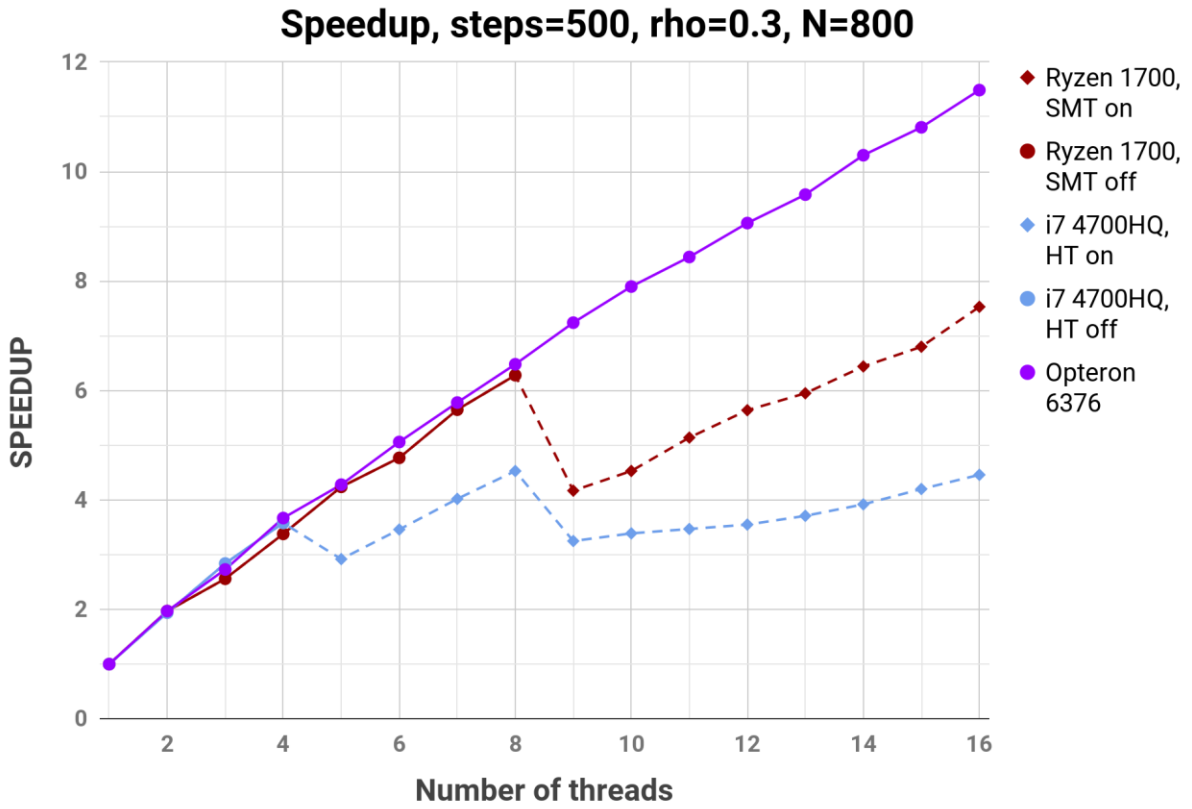


*Figure 1: Speedup della versione OpenMP con N_STEPS=500, rho=0.3 e N=800. Le linee tratteggiate indicano che il numero di thread OpenMP supera il numero di core della CPU, attivando di conseguenza HT/SMT.*

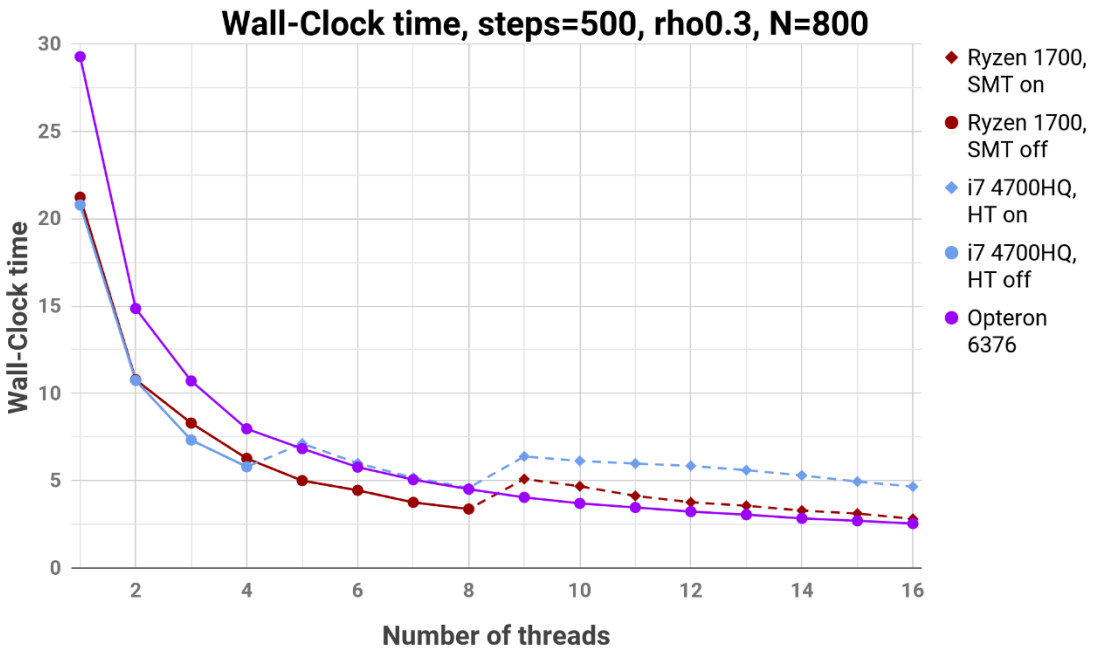


*Figure 2: Wall-clock time della versione OpenMP con N_STEPS=500, rho=0.3 e N=800. Le linee tratteggiate indicano che il numero di thread OpenMP supera il numero di core della CPU, attivando di conseguenza HT/SMT.*

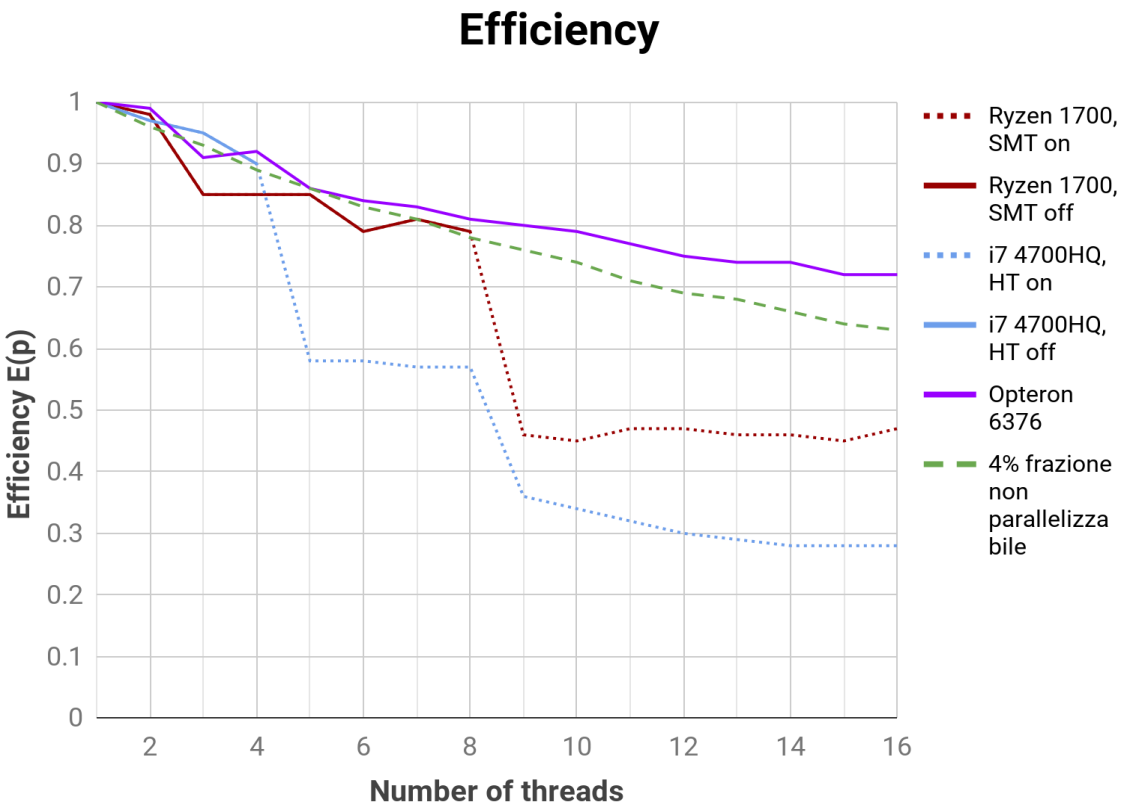


*Figure 3: Efficienza della versione OpenMP con N_STEPS=500, rho=0.3 e N=800. Le linee tratteggiate indicano che il numero di thread OpenMP supera il numero di core della CPU, attivando di conseguenza HT/SMT.*

Data la natura del problema, ovvero la necessità di sincronizzare i due step (horizontal_step e vertical_step), non è possibile avere una totale parallelizzazione, inoltre i thread che finisco prima l'esecuzione di uno step devono aspettare che tutti gli altri abbiano completato. Dalla figura 3 riguardante l'efficienza, possiamo dedurre che la frazione di codice non parallelizzata sia inferiore al 4%.

## Implementazione del modello BML con CUDA

L'implementazione con CUDA è caratterizzata da 2 kernel, horizontal_step(...) e vertical_step(...), i quali eseguono lo stesso lavoro delle rispettive funzioni della versione seriale. Entrambi i kernel hanno la stessa modalità di funzionamento: viene creata una grid con abbastanza blocks in modo da associare ogni cella del modello ad un solo cuda-thread, il quale provvederà all'aggiornamento della cella.

Siccome le chiamate ai kernel avvengono in maniera asincrona, per una esecuzione corretta è stato necessario sincronizzarli tramite la funzione cudaDeviceSynchronize(). Inoltre, si è scelto di utilizzare un blocksize basso (=16) per ragioni di compatibilità, in quanto i hardware più vecchi non sono dotati di abbastanza memoria shared.

Durante lo sviluppo si è notato che la maggiore fonte di overhead è dovuta alla latency dell'accesso alla memoria globale della GPU. Di conseguenza si sono proposte due soluzioni: la prima salva le variabili globali nella memoria locale, così da non dover accedere due alla stessa locazione di memoria globale; la seconda usa la shared memory dei block.

Le prestazioni dipendono, naturalmente, dall'hardware usato e dal numero di cuda cores presenti nella GPU. In particolare, dalla figura 4, si è osservato che per schede di vecchia generazione come la Tesla C870, la versione con shared memory migliora in modo significativo le performance. Si ottengono, invece, risultati opposti con quelle più recenti, peggiorando sensibilmente i tempi di esecuzione.

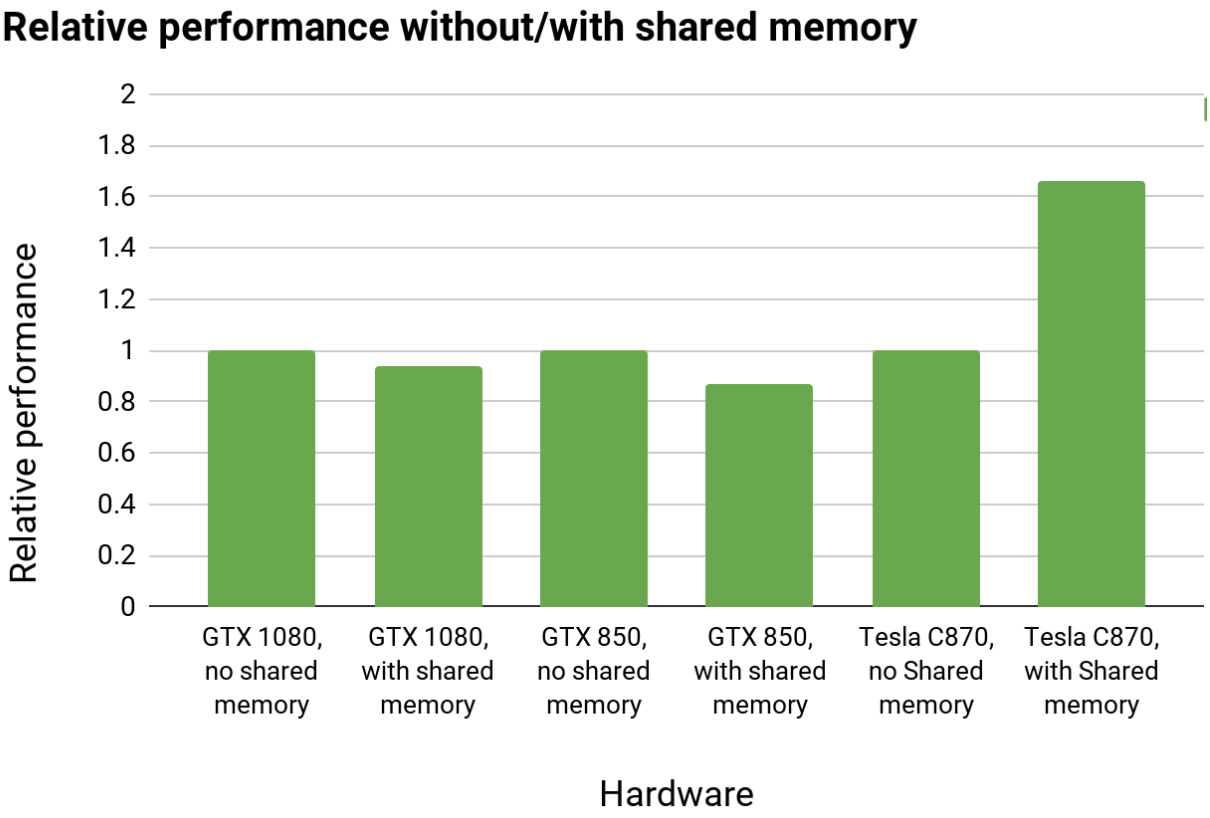


*Figure 4: Prestazioni della versione con shared memory in rapporto alla versione senza shared memory della medesima GPU.*

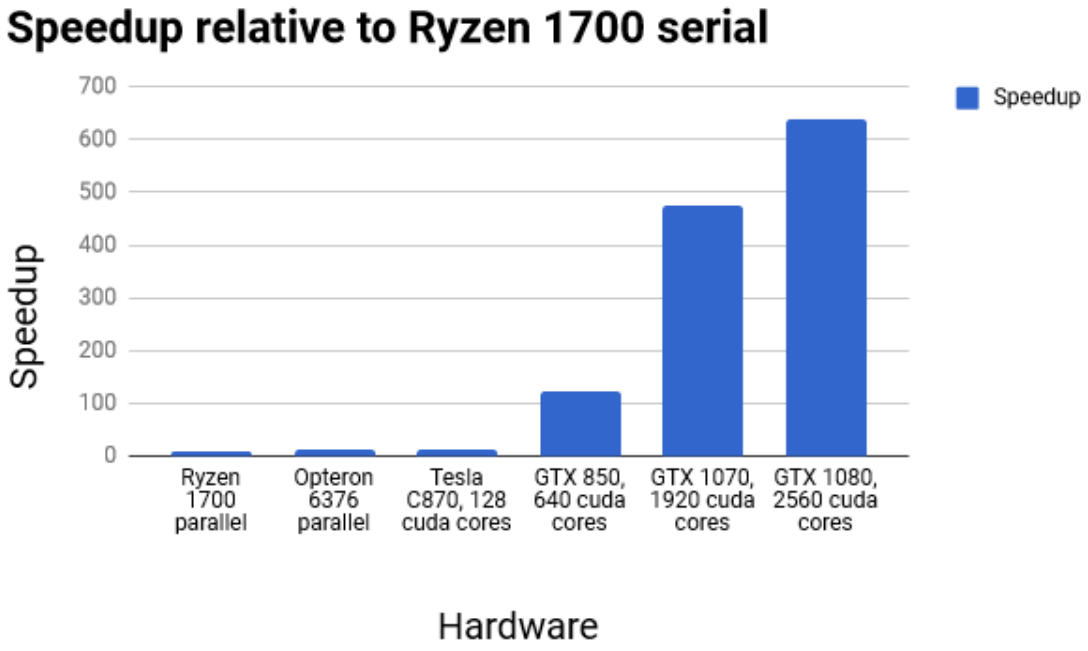


*Figure 5: Speedup della versione CUDA (senza shared memory) rispetto alla versione seriale eseguito su Ryzen 1700. Configurazione: N_STEPS=500, rho=0.3 e N=800.*

Dalla figura 5 possiamo notare che si è ottenuto un'ottima scalabilità della versione CUDA. In particolare, se teniamo esclusivamente conto del numero di cuda cores, osserviamo uno speedup superscalare all'aumentare dei cuda cores. Ovviamente questo **non** sarà sicuramente vero se andiamo ad aggiungere altri fattori, come memorie GDDR più veloci e frequenze maggiori delle diverse GPU.

## Conclusioni

Il modello BML si appresta meglio alla implementazione CUDA, in quanto in grado di sfruttare la parallelizzazione massiva offerta dalle GPU.

In generale, dai dati ricavati, si può affermare che le performance di una moderna GPU sono di due ordini di grandezza superiori rispetto al programma seriale eseguito su una CPU.