Progetto di High Performance Computing

Fabrizio Margotta (fabrizio.margotta@studio.unibo.it)
Matricola 789072

Alma Mater Studiorum-Università di Bologna

27/09/2019

Abstract

Lo scopo del progetto è di implementare versioni parallele di un semplice modello matematico di propagazione dei terremoti. Il modello che consideriamo è una estensione in due dimensioni dell'automa cellulare Burridge-Knopoff (BK).[2]

1 Introduzione

La realizzazione dell'intero progetto è avvenuta per fasi: inizialmente è stata implementata la versione OpenMP, solo in seguito la versione CUDA. Per ogni fase si è proceduti allo sviluppo della soluzione proposta per raffinamenti successivi e, contestualmente, alla scrittura della relazione.

L'intero progetto è stato gestito seguendo una versione semplificata della metodologia Kanban[4], utilizzando opportuni strumenti digitali per l'organizzazione delle singole attività di lavoro.

1.1 Analisi

L'analisi sul modello da implementare è stata particolarmente semplificata dalle indicazioni fornite dal docente.

Vengono tuttavia di seguito presentate alcune osservazioni sul modello dei dati, ad esempio si fa notare che non è presente alcuna particolare dipendenza sui dati da parte delle iterazioni dei cicli.

Ciò permette quindi di procedere con la parallelizzazione della versione seriale fornita dal docente senza particolari problemi.

2 Sviluppo

2.1 Ghost area

Entrambe le soluzioni proposte si avvalgono dell'utilizzo di una ghost area inizializzata a zero (e mai più modificata).

Ciò permette di ridurre notevolmente l'impatto sulle prestazioni dell'operazione di propagazione, poiché, avendo una nuova regione di celle intorno al dominio, non sarà più necessario controllare che le celle adiacenti a quella presa in esame siano ai bordi del dominio stesso. Il valore delle ghost cell (zero), inoltre, non influirà sui risultati delle operazioni dell'algoritmo.

In figura 1([3]) è possibile osservare il comportamento appena descritto.

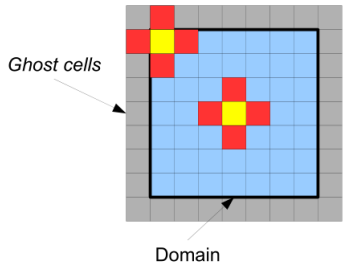

Figure 1: Rappresentazione della ghost area (celle grigie).

2.2 Accesso in memoria al dominio

È stato inizialmente ipotizzato l'accesso alla matrice considerandola come un array, in accordo con la rappresentazione in memoria delle matrici fornita dal linguaggio C. Ciò avrebbe permesso di scorrere in modo sequenziale la matrice stessa, risparmiando le chiamate alla funzione `IDX`.

Ad esempio, l'operazione di incremento di energia sarebbe stata implementata nel seguente modo:

```
for (int i = 0; i < n; i++) {
    grid[i] += delta;
}
```

Tuttavia, l'introduzione della *ghost area* ha reso preferibile (in termini di gestione dell'accesso alla memoria) l'utilizzo della funzione `IDX`, che contribuisce inoltre ad una buona comprensione e leggibilità del codice sorgente.

2.3 Versione OpenMP

In figura 2 è possibile osservare il risultato della simulazione in termini di energia media delle celle e numero di celle che hanno superato il valore soglia, misurate ad ogni passo di esecuzione.

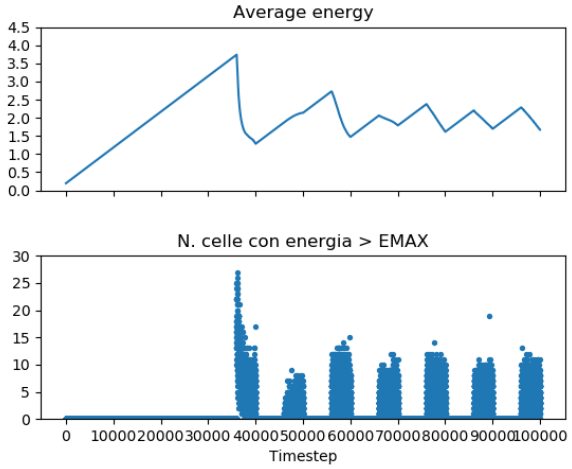

Figure 2: Risultato della simulazione.

2.3.1 Modifiche alla versione seriale

setup L'inizializzazione della ghost area a zero non è stata parallelizzata poiché i test effettuati e mostrati nella tabella 1 hanno evidenziato un degradamento, seppur minimo, delle prestazioni.

	Lato dominio	Numero passi	T_{setup} (s)
seriale	512	1000	0.00337177
parallelizzato			0.00454419

Table 1: Confronto dei tempi di esecuzione per la funzione `setup` nella versione seriale e nella versione parallelizzata.

Tale comportamento può essere dovuto ad un ulteriore carico di lavoro assegnato allo scheduler per un'attività che mal si presta (soprattutto per l'accesso ai lati sinistro e destro del dominio a causa dell'accesso in memoria) ad essere parallelizzata.

altre operazioni Tutte le principali operazioni (incremento, conteggio, propagazione e calcolo energia media) contengono dei cicli `for` che sono stati parallelizzati utilizzando la direttiva OpenMP `omp parallel for`. In tale modo l'intero carico di lavoro del ciclo a cui è stato applicato il costrutto viene suddiviso tra i thread che concorrono all'esecuzione del programma, realizzando la vera e propria parallelizzazione.

Nelle operazioni di conteggio e di calcolo dell'energia media è stata inoltre introdotta la clausola di riduzione con l'operatore somma, per poter parallelizzare anche le operazioni di somma all'interno delle direttive OpenMP.

2.3.2 Ulteriori ottimizzazioni

collapse È stato valutato l'utilizzo della clausola `collapse(2)` sui cicli `for` innestati per collassare le due iterazioni in una, allargando lo spazio di iterazione come da specifiche[1].

Tale clausola ha tuttavia causato un degrado delle prestazioni (si veda la tabella 2) e pertanto non è stata implementata nella soluzione proposta.

	Lato dom.	# passi	$\overline{T}_{\text{3-run}}(s)$	cache-references (M/sec)	cache-misses (%)*
senza `collapse(2)`	256	100000	13.33183	0.770	0.390
con `collapse(2)`			19.18336	0.560	0.402

Table 2: Dati raccolti sull'esecuzione della simulazione con e senza la direttiva `collapse(2)`.

*: of all cache refs.

Dai dati presenti in tabella è possibile osservare che, al contrario di come ci si aspetterebbe, nonostante la clausola `collapse` modifichi i *chunk* di memoria a cui si accede, ciò non comporta un degradamento significativo delle prestazioni nell'accesso alla memoria cache.

Di conseguenza si ipotizza che l'incremento delle tempistiche di esecuzione possa essere legato ad una cattiva gestione da parte dello scheduler nell'organizzare il lavoro dei thread.

schedule È stata inoltre valutata l'adozione della clausola `schedule`, utilizzando il parametro `type` nelle sue varianti `static`, `dynamic` e `guided`.

Le prove effettuate hanno mostrato un significativo aumento delle prestazioni impostando uno scheduling statico con blocchi di dimensione 8.

La tabella 3 mostra la differenza di tempistiche medie (su 3 esecuzioni) tra la versione in cui la clausola `schedule` non è specificata e quella in cui viene usata con parametri `static, 8`.

Lato dom.	# passi	$\overline{T}_{\text{3-run}}(s)$	$\overline{T}_{\text{3-run}}(s)$ con `schedule(static,8)`
256	100000	12.9581	10.3148
512		50.4409	40.5048
1024		250.4084	177.2195

Table 3: Comparazione delle tempistiche di esecuzione applicando uno scheduling statico.

Tale clausola ha quindi permesso una importante riduzione delle tempistiche, compresa tra il 20 e il 30% in meno rispetto alla precedente versione già parallelizzata.

Altri tipi di scheduling o diverse dimensioni dei *chunk* hanno pareggiato o addirittura peggiorato le precedenti tempistiche.

2.4 Versione CUDA

Il dominio è stato partizionato, come da suggerimenti, in blocchi 2D organizzati a loro volta in griglie 2D.

Per le operazioni effettuate nelle funzioni `count_cells` e `average_energy` il dominio è stato invece partizionato in blocchi monodimensionali, in accordo con la rappresentazione matriciale nel linguaggio di programmazione C.

A tal proposito, per le sopracitate operazioni, è stata opportunamente implementata l'operazione di riduzione mediante operatore somma.

In un primo momento, per semplicità, si è proceduto con la realizzazione della soluzione senza l'utilizzo della memoria condivisa del Device.

3 Valutazione prestazioni

AMD Opteron (tm) Processor 6376 8 core 16 thread.

Le tempistiche indicate sono state tutte ottenute mediante le funzioni di libreria messe a disposizione dal docente. Per ulteriori informazioni si consulti il codice sorgente.

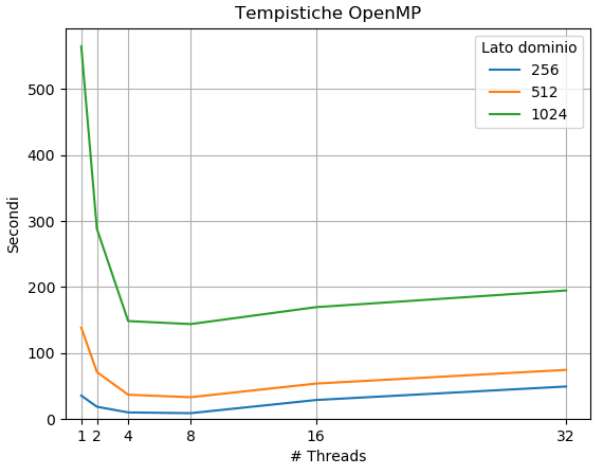

Figure 3: Comparazione tempistiche di esecuzione in OpenMP.

3.1 Speedup

Il calcolo dello speedup viene eseguito mediante la seguente formula:

$$S(p) = \frac{T_{serial}}{T_{parallel}(p)}$$

in cui:

p	:	# processori/core
T_{serial}	:	tempo di esecuzione della porzione seriale (T_{serial}=$T_{parallel}(1)$)
$T_{parallel}(p)$	:	tempo di esecuzione della porzione con p processori/core

Poiché l'implementazione della soluzione contiene porzioni di codice non parallelizzabili (si veda la funzione `setup`), dovremmo considerare $T_{parallel}(p)$ nel seguente modo:

$$T_{parallel}(p) = \alpha \cdot T_{serial} + \frac{(1-\alpha) \cdot T_{serial}}{p}$$

in cui α è il fattore relativo alla porzione di codice non parallelizzabile, calcolato come segue:

$$\alpha = \frac{T_{serial}}{T_{serial} + T_{parallel}}.$$

Tuttavia, come mostrato nella seguente tabella, la porzione di codice seriale (non parallelizzabile) impiega tempo trascurabile rispetto alla porzione di codice parallelizzata, pertanto anche α assume valore trascurabile.

Lato dominio	Numero passi	$T_{seriale}$ (s)	$T_{parallelo}$ (s)	α
256	100000	0.00140063	14.2519	0.00009827
1024		0.02356447	150.2037	0.00015686

Table 4: caption

Per le siffatte osservazioni, considereremo $T_{parallel}(p)$ in tal modo:

$$T_{parallel}(p) = \frac{T_{serial}}{p}.$$

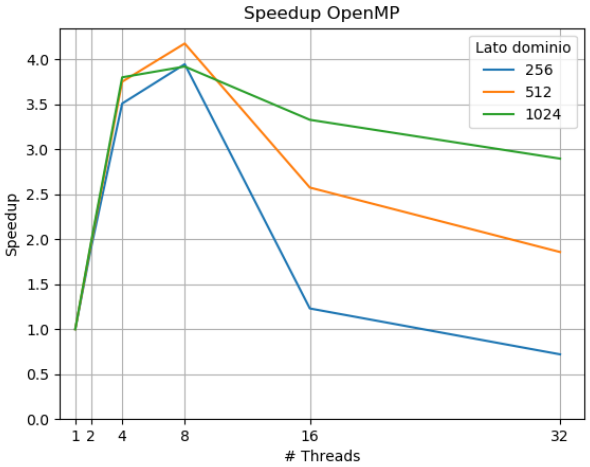

Figure 4: Grafico dello speedup.

3.2 Strong scaling

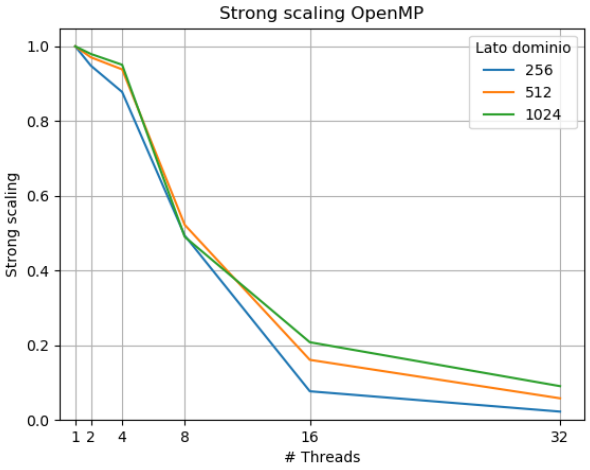

Figure 5: Grafico dello strong scaling.

3.3 Weak scaling

Dimensione del dominio:

$$side(p) = 256 \cdot \sqrt[3]{p}$$

come mostrato negli script di esempio del docente.

$$W(p) = \frac{T_1}{T_p}$$

in cui:

p	:	# processori/core
T_1	:	tempo di esecuzione di una unità di lavoro con un processore/core
T_p	:	tempo di esecuzione di p unità di lavoro con p processori/core

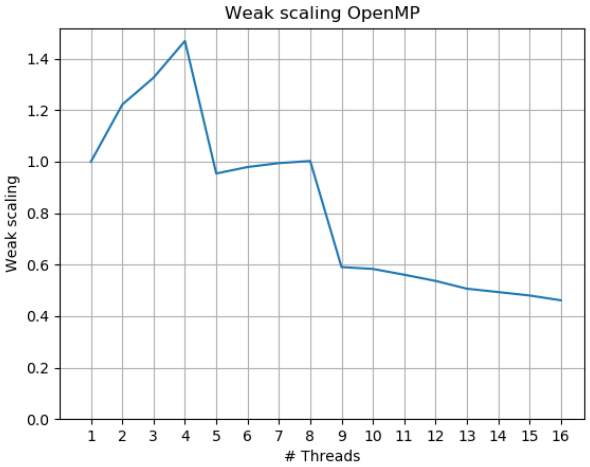

Figure 6: Grafico dello weak scaling.

4 Conclusioni

4.1 Note di sviluppo

Lo sviluppo dell'implementazione del modello è stato accompagnato dalla realizzazione di opportuni script orientati a rendere più rapido il testing sul server fornito dal docente.

Sono stati infatti realizzati meccanismi per il trasferimento, compilazione, esecuzione e ottenimento dei risultati da/verso il server utilizzando il programma `scp` e l'opzione `ProxyJump` di ssh. A tal proposito si consultino i file `.autoenv.zsh` e `README.md` del progetto ove è possibile trovare indicazioni e dettagli tecnici su come sfruttare il meccanismo ideato.

References

[1] OpenMP Architecture Review Board. *OpenMP Application Programming Interface*, 5.0 edition, November 2018.

[2] Robert Burridge and Leon Knopoff. Model and theoretical seismicity. *Bulletin of the seismological society of america*, 57(3):341–371, 1967.

[3] Moreno Marzolla. Lucidi delle lezioni per il corso di High Performance Computing: Parallel programming patterns, 38, September 2018.

[4] Wikipedia contributors. Kanban (development) — Wikipedia, the free encyclopedia, 2019. [Online; accessed 25-September-2019].