Relazione Progetto HPC 2023

Michele Montesi
Matricola: 0000974934
E-Mail: michele.montesi3@studio.unibo.it

12 aprile 2023

Introduzione

La presente ricerca presenta lo sviluppo di due versioni parallelizzate del software `sph.c`, implementate utilizzando la libreria *OpenMP* e la libreria *MPI* rispettivamente. L'obiettivo della ricerca è quello di valutare i vantaggi prodotti dall'utilizzo della programmazione multiprocessore.

Per valutare le prestazioni del software, è stato creato uno script in linguaggio Python che esegue il programma con un numero crescente di thread per *OpenMP* e di processi per *MPI*, partendo da 1 e arrivando a 12, e con un numero di particelle che varia da 1400 a 5700. Lo script registra i dati rilevanti in un foglio di lavoro Excel, dai quali verrà calcolata la media. Queste informazioni saranno poi utilizzate per valutare le prestazioni.

I software sono stati testati sul server `ISI-Raptor`

Versione OpenMP

Implementazione

In questa implementazione del software, effettuata con *OpenMP*, sono state parallelizzate le sezioni di codice che richiedono maggiore elaborazione utilizzando la direttiva `#pragma omp parallel for`.

Dove necessario sono state effettuate riduzioni al fine di ottimizzare le prestazioni del software parallelizzato.

Prestazioni

Dalla figura 1 si nota come all'aumentare del numero di thread, lo *speedup* aumenti in modo significativo, come si può notare dall'aumento della curva. Tuttavia, dopo aver raggiunto 8 thread, si può notare una fase di saturazione, in cui l'aumento del numero di essi non porta più a un miglioramento significativo delle prestazioni. In questo punto, l'utilizzo di thread aggiuntivi potrebbe peggiorare le prestazioni.

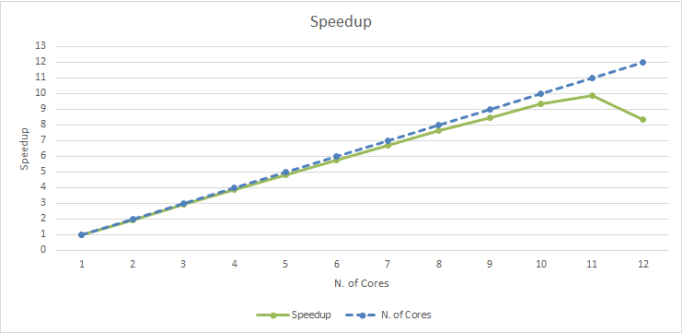

Figura 1: Speedup dell'implementazione con OpenMP

Dal grafico in figura 2 si può notare come la *Strong Scaling Efficiency* raggiunga il suo picco massimo per circa 6 thread, corrispondente a un'efficienza di circa il 90% rispetto all'efficienza ideale. Dopo il picco, questa inizia a diminuire, il che significa che l'aggiunta di thread aggiuntivi non porta a un miglioramento significativo delle prestazioni, e può addirittura causare un degrado delle prestazioni complessive.

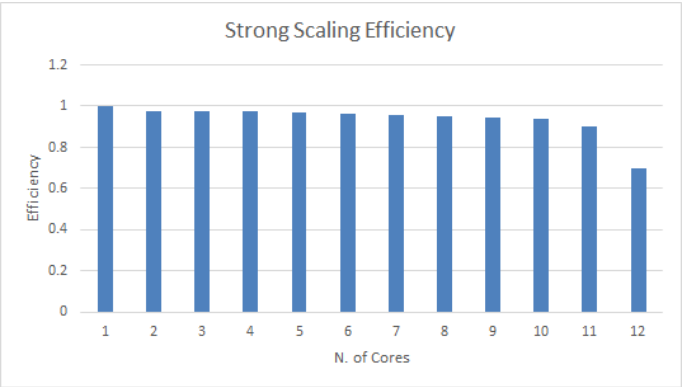

Figura 2: Strong scaling efficiency dell'implementazione con OpenMP

Nella figura 3 si può notare come la *Weak Scaling Efficiency* rimanga costante all'aumentare della dimensione del problema. Questo indica che il software è in grado di gestire in modo efficiente problemi di dimensioni diverse, distribuendo il carico di lavoro tra le CPU disponibili senza una diminuzione significativa delle prestazioni. Questa costante è un indicatore di una buona scalabilità del software. A 12 thread, l'efficienza debole cala, indicando che l'aggiunta di ulteriori non comporta migliori prestazioni.

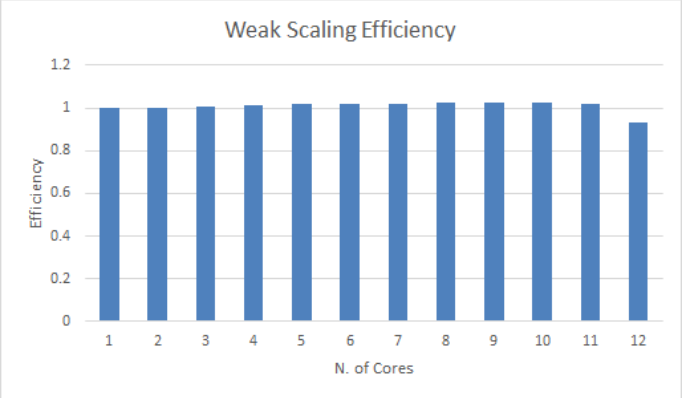

Figura 3: Weak scaling efficiency dell'implementazione con OpenMP

Versione MPI

Implementazione

Per l'implementazione del software con `MPI` è stato creato un `MPI_Datatype` contiguo per incapsulare la struttura dati durante lo scambio di messaggi. Per suddividere le particelle fra tutti i processori viene fatta una `MPI_Scatterv` dal processo 0. Dopo `compute_density_pressure` e `integrate` viene chiamata una `MPI_Allgatherv` per raccogliere i dati elaborati e ridistribuirli nuovamente a tutti i processori mentre per `compute_forces` non viene fatto in quanto questo agisce solamente in locale. Alla fine, dopo aver calcolato la velocità media delle particelle locali, viene eseguita una `MPI_Reduce` con operatore somma. Questo risultato viene diviso per tutti i processi ottenendo così la velocità media complessiva.

Prestazioni

Dalla figura 4 si può notare che le prestazioni del software aumentano in modo quasi lineare all'aumentare del numero di unità di elaborazione utilizzate. Tuttavia, il guadagno di prestazioni in termini di *speedup* diminuisce man mano che se ne aggiungono di nuove, a causa di fattori come l'aumento del tempo di comunicazione tra di esse, la necessità di sincronizzazione tra i processi e la presenza di parti del software che non possono essere parallelizzate. Inoltre, si può notare che lo *speedup* inizia a saturare intorno a 8 core.

Figura 4: Speedup dell'implementazione con MPI

Dal grafico in figura 5 emerge che la *Strong Scaling Efficiency* raggiunge il suo massimo picco intorno a 6 core, ottenendo un'efficienza di circa il 90% rispetto all'efficienza ideale. Tuttavia, dopo il picco, l'efficienza inizia a diminuire, indicando che l'aggiunta di ulteriori core non comporta un miglioramento significativo delle prestazioni, rischiando inoltre di peggiorarle in alcuni casi.

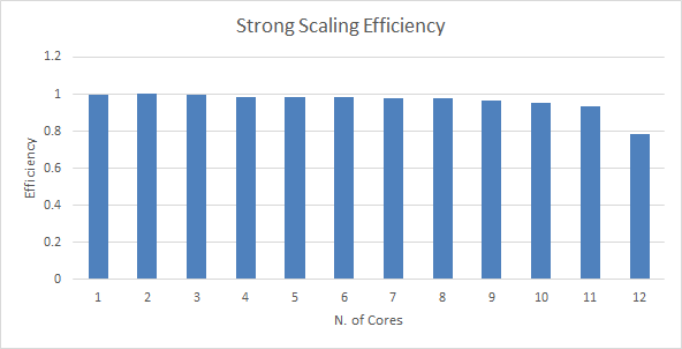

Figura 5: Strong scaling efficiency dell'implementazione con MPI

Il grafico della *Weak Scaling Efficiency* (Figura 6) mostra che l'efficienza del software rimane costante al variare della dimensione del problema. Ciò suggerisce che questo è in grado di gestire in modo efficiente problemi di dimensioni diverse, distribuendo il carico di lavoro tra le CPU disponibili senza una diminuzione significativa delle prestazioni. Questo è un indicatore della buona scalabilità del software. Tuttavia, l'aggiunta di ulteriori core oltre i 12 porta a una diminuzione dell'efficienza debole, indicando che l'uso di ulteriori risorse non si traduce in un miglioramento delle prestazioni.

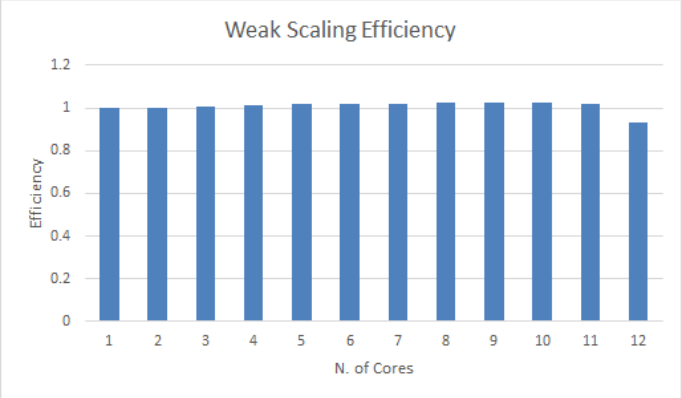

Figura 6: Weak scaling efficiency dell'implementazione con MPI

Conclusioni

L'implementazione del software con *OpenMP* ha portato a notevoli miglioramenti delle prestazioni, come indicato dal grafico di *speedup* in funzione del numero di *thread*. Si è osservato che l'aumento del numero di questi ha portato a un miglioramento significativo delle prestazioni fino a un certo punto, dopodiché l'effetto positivo si è saturato, indicando che l'utilizzo di ulteriori potrebbe persino peggiorare le prestazioni. La *strong scaling efficiency* è risultata piuttosto elevata per i casi con un numero ridotto di *thread*, ma è diminuita con l'aumento del numero di questi. La *weak scaling efficiency* è stata buona e costante al variare del numero di processori.

L'implementazione con *MPI* ha portato a una significativa riduzione dei tempi di esecuzione rispetto alla versione sequenziale del software, come indicato dal grafico di *speedup* in funzione del numero di processori. Si è osservato che la *strong scaling efficiency* è stata molto elevata per tutti i casi analizzati, indicando una buona scalabilità del software. La *weak scaling efficiency* è stata buona e costante al variare del numero di processori.

In conclusione, l'implementazione del software con *OpenMP* e *MPI* ha portato a significativi miglioramenti delle prestazioni rispetto alla versione sequenziale. L'approccio OpenMP è particolarmente efficace per problemi di dimensioni limitate e con un numero ridotto di thread, mentre l'approccio *MPI* è più efficiente per problemi di dimensioni maggiori e con un numero elevato di processori.