Progetto di High Performance Computing 2018/2019

Lorenzo Casini, matr. 0000800947
28/09/2019

Introduzione

Lo scopo di questo progetto è realizzare due versioni che sfruttano la parallelizzazione sia a memoria condivisa che a memoria distribuita di un programma seriale *erathquake.c* come base di partenza.

Ho scelto di implementare la versione a memoria condivisa con OpenMP metre ho realizzato la versione a memoria distribuita con MPI.

Per comodità ho realizzato un repository github per gestire lo sviluppo del progetto sopratutto per passare in maniera agevole le modifiche sul server *isi-raptor03* ed effetturare i vari test dei programmi realizzati.

Link repo: https://www.github.com/k4s0/ProgettoHPC

Per il progetto sono state usate le slide e gli esempi presentati nel corso.

Versione OpenMP

Per realizzare la versione OpenMP del programma seriale mi sono focalizzato principalmente sui vari step che quest'ultimo eseguiva in quanto ho riscontrato le maggiori criticità all'interno di esse, e se parallelizzate potevano dara il maggior beneficio in termini di performance. Altra questione da tenere in considerazione il ciclo principale di esecuzione del programma seriale non è stato parallelizzato in quanto conteneva diverse dipendenze, per esempio la funzione di inizzalizzazione è eseguita da un solo processo.

(**iterazione**)

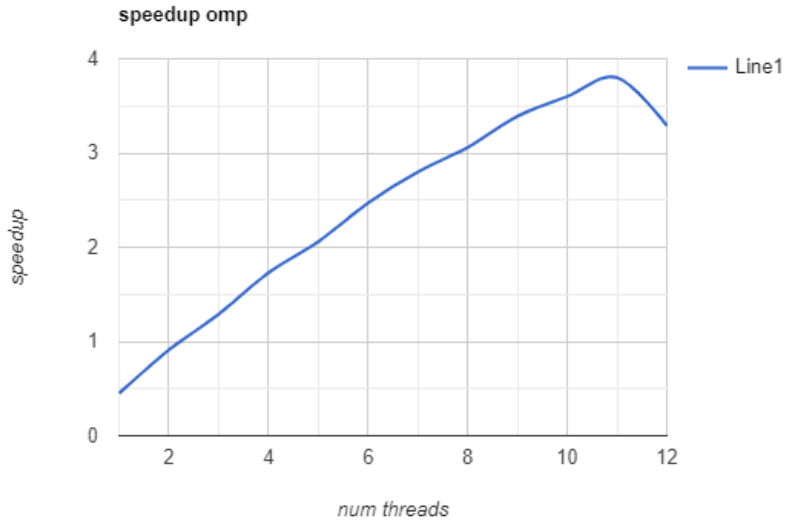
Come primo passo il programma esegue la funzione *increment_energy*, che presentava al suo interno due cicli che valorizzavano tutte le celle del dominio causando un *embarassingly parallel problem*, perché ogni singola area può essere calcolata indipendetemente dalle altre.

*(i)*commenti nel codice**

(**propagazione**)

Il secondo passo esegue la funzione *propagate_energy*, la prima cosa è stata l'implemetazione di una ghost area, questo ci permette di evitare un numero considerevole di controlli durante l'esecuzione. Per rompere le dipendenze fra i cicli ogni cella controlla se le celle affianco hanno soglia maggiore di Emax.

(ii)*commenti nel codice*

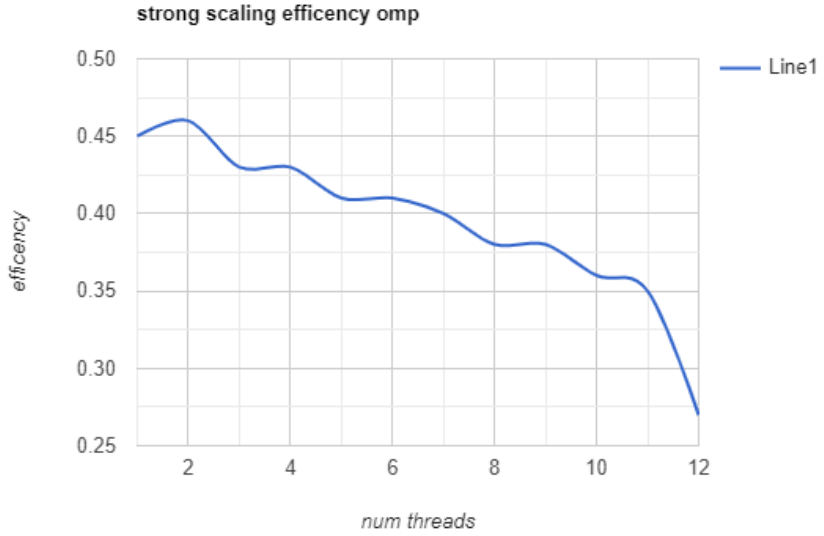


Il grafico illustra lo speedup del programma realizzato in OpenMP, come è possibile notare lo speedup cresce all'aumentare del numero dei threads. Sia il grafico che l'output della console ci mostrano che la versione OpenMP eseguita con 1 core subisce un considerevole rallentamento (S:42.1138 sec. VS OMP: 94.2977 sec.), però successivmanete incremnetando il numero di core il programma aumenta notevolmente la sua velocità di esecuzione fino ad arrivare a 11.8 sec contro i 42.1 sec che impiega la versione seriale. Altro crollo di prestazioni lo notiamo quando il programma inizia ad usare più di 11 core, si denota il fatto che fatica evidentemente a parallelizzare le operazioni causando un overhead.

```
Testing the serial version...
./earthquake : 6553.6000 Mupdates in 42.1138 seconds (155.616555 Mupd/sec)

Testing the OpenMP version with 1 threads...
./omp-earthquake : 6553.6000 Mupdates in 94.2976 seconds (69.499121 Mupd/sec)
Testing the OpenMP version with 2 threads...
./omp-earthquake : 6553.6000 Mupdates in 46.5954 seconds (140.648986 Mupd/sec)
Testing the OpenMP version with 3 threads...
./omp-earthquake : 6553.6000 Mupdates in 31.8582 seconds (205.711492 Mupd/sec)
Testing the OpenMP version with 4 threads...
./omp-earthquake : 6553.6000 Mupdates in 24.8199 seconds (264.046072 Mupd/sec)
Testing the OpenMP version with 5 threads...
./omp-earthquake : 6553.6000 Mupdates in 20.3509 seconds (322.030189 Mupd/sec)
Testing the OpenMP version with 6 threads...
./omp-earthquake : 6553.6000 Mupdates in 17.4603 seconds (375.341905 Mupd/sec)
Testing the OpenMP version with 7 threads...
./omp-earthquake : 6553.6000 Mupdates in 15.4514 seconds (424.141939 Mupd/sec)
Testing the OpenMP version with 8 threads...
./omp-earthquake : 6553.6000 Mupdates in 13.7529 seconds (476.525380 Mupd/sec)
Testing the OpenMP version with 9 threads...
./omp-earthquake : 6553.6000 Mupdates in 12.7568 seconds (513.735120 Mupd/sec)
Testing the OpenMP version with 10 threads...
./omp-earthquake : 6553.6000 Mupdates in 11.8192 seconds (554.489092 Mupd/sec)
Testing the OpenMP version with 11 threads...
./omp-earthquake : 6553.6000 Mupdates in 11.8448 seconds (553.288312 Mupd/sec)
Testing the OpenMP version with 12 threads...
./omp-earthquake : 6553.6000 Mupdates in 12.8069 seconds (511.725235 Mupd/sec)
All test are done.
STUDENTI\lorenzo.casini@isi-raptor03:~/hpc/ProgettoHPC1819/src$
```

Nel secondo grafico è rappresentata l'efficenza del programma in confronto al programma seriale.

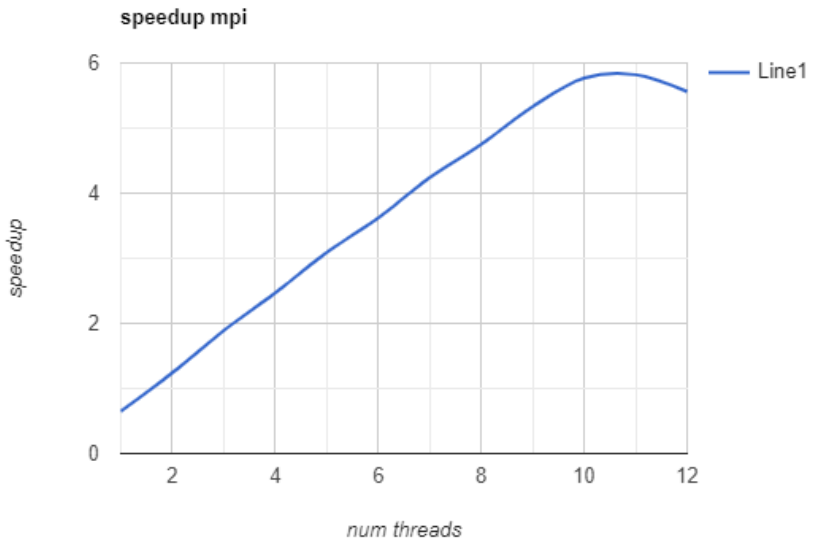


Versione MPI

Per la versione MPI come già realizzato in quella OperMP ho creato una ghost area, tenendo in considerazione il numero di core ho suddiviso la matrice in tante righe quanti core presenti, ad ogni iterazione aggiorno le informazioni tramite l'istruzione *mpi_sendrecv.*

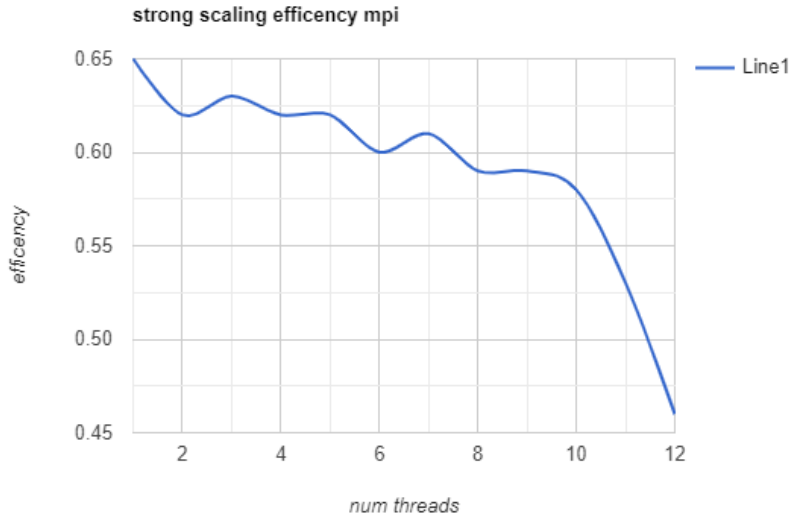
Per trovare la media dell'energia e calcolare il numero di celle che presentano energia maggiore di Emax, come nel caso di OpenMP, sono tutte operazioni che possone essere eseguite dalla chiamata della funzione *mpi_reduce.*

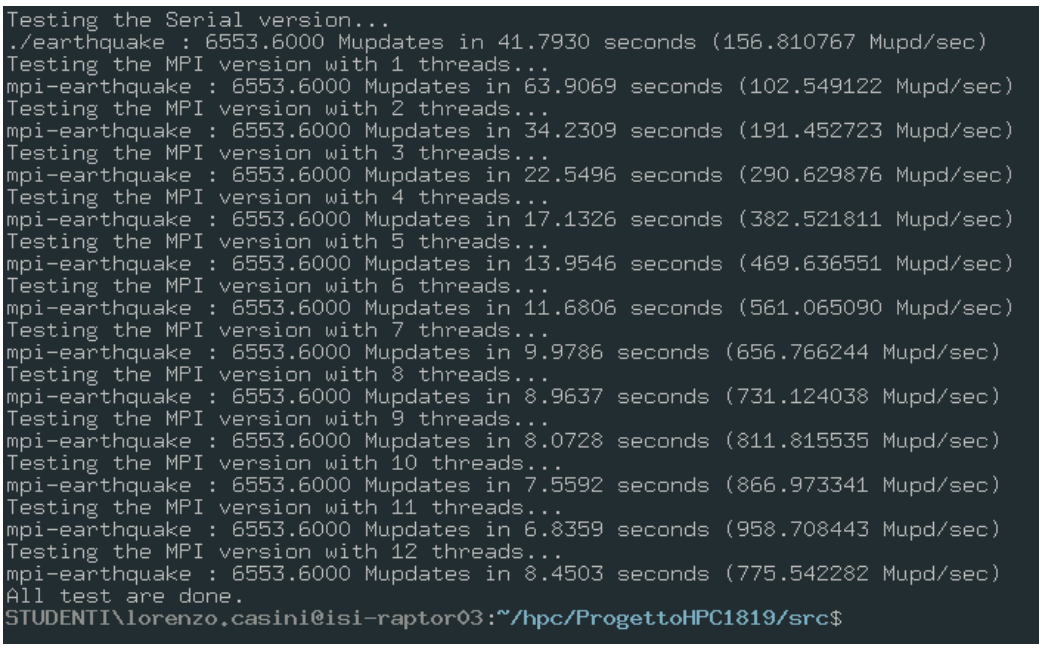
Come già specificato nel programma OpenMP anche qui il ciclo principale non è stato possibile parallelizzarlo, sono stati tenuti in considerazione anche i commenti presenti nell funzione *setup().*



Nel grafico precedente è possibile osservare lo speedup del programma in versione MPI.

Sotto è presente invece il grafico che rappresenta l'efficenza del programma MPI





Sopra è possibile consultare l'output a console del programma MPI con i tempi di esecuzione a confronto con la versione seriale.

Conclusioni

Osservando i grafici sopra riportati, i calcoli eseguiti nel file *graph-calc.ods* che gli output della console dopo aver eseguito tutte e tre le versioni del programma, le versioni parallelizzate hanno le massime performance quando vengono eseguite con *10 threads OpenMP* e *11 threads MPI.*

Mettendo a confronto il tempo di esecuzione del programma seriale (41 sec.) e i due risultati migliori a parità di core delle versioni parallelizzate si evidenzia che:

OpenMP viene eseguito in 11.8 sec.

MPI viene eseguito in 6.8 sec.

Quindi concludendo la versione più efficiente per risolvere il problema sopra indicato è quella a memoria distribuita.

Riferimenti bibliografici

Per la realizzazione dei grafici mi sono basato sulla slide:

- **L02-performance-evaluation.pdf**

Per la risoluzione di alcune problematice legate al programma omp mi sono basato su queste slide:

- **L03-patterns.pdf**
- **L04-OpenMP.pdf**