Progetto di High Performance Computing 2021/2022

Daniel Guariglia, matr. 0000916433

28 settembre 2022

Indice

1 Introduzione

Questa relazione contiene una rapida descrizione ed analisi delle strategie di parallelismo adottate per il calcolo di un modello HPP, la prima a memoria condivisa tramite OpenMP, la seconda a memoria distribuita tramite MPI. Tutti i grafici riportati sono stati ottenuti tramite esecuzioni su server (isi-raptor) usando un massimo di 12 core.

2 Versione OpenMP

2.1 Descrizione

Analizzando il codice della soluzione seriale proposta è stata individuata all'interno della funzione "step", responsabile della computazione delle fasi pari e dispari, la porzione di codice da parallelizzare; nello specifico all'interno di questa funzione sono presenti due `for` annidati che eseguono N/2 iterazioni l'uno.

Entrambi i cicli incrementano il proprio indice a multipli di 2 fino al raggiungimento di N, l'indice del ciclo esterno è usato per scorrere le righe del dominio, mentre quello più interno le colonne; le operazioni effettuate all'interno dei cicli non presentano dipendenze tra le iterazioni.

Parallelizzando tramite la direttiva `#pragma omp parallel for` il ciclo esterno ogni thread riceve una porzione del dominio indipendente.

2.2 Valutazione delle prestazioni

Tramite l'inserimento della direttiva descritta sopra è stato ottenuto un speedup lineare 2.1. Di seguito è riportato il grafico della Strong scaling efficiency 2.2.

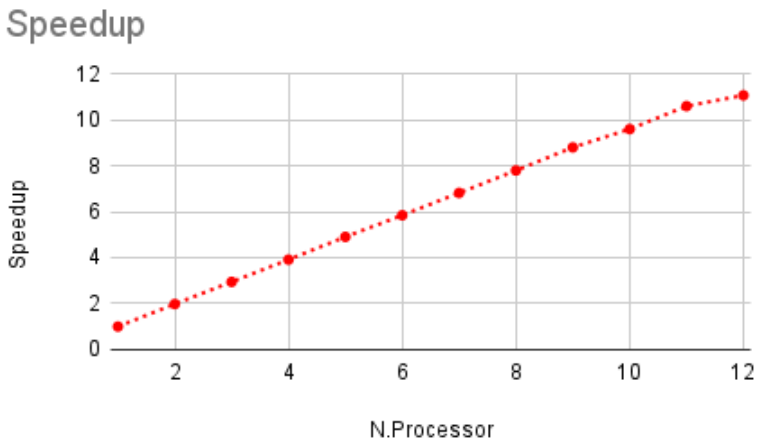

Figura 2.1: Grafico speedup lineare ottenuto con test su server isi-raptor

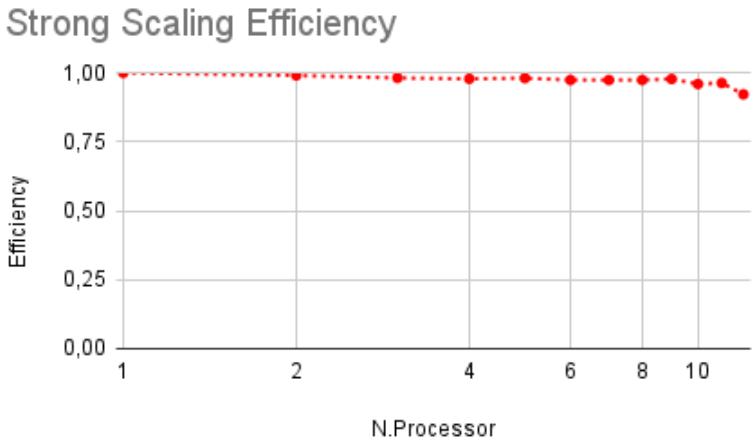

Figura 2.2: Grafico Strong scaling efficiency ottenuto con test su server isi-raptor

3 Versione MPI

3.1 Descrizione

Il dominio iniziale viene caricato dal processo 0 che si occupa della lettura da file e successivamente della divisione e distribuzione tra i processi tramite l'uso della funzione `MPI_Scatterv` avvalendosi di un `MPI_Datatype` di tipo `contiguous` di dimensione uguale a due righe. Inizialmente ogni processo riceve una porzione sempre di righe pari del dominio con il quale può operare la fase pari 3.1. Ogni processo dichiara un puntatore ad un'area di memoria di dimensione sufficiente per contenere la porzione di dominio che dovrà elaborare più una riga inizialmente vuota che verrà usata nella fase dispari, quest'ultima riga viene inizialmente esclusa nell'elaborazione della fase pari.

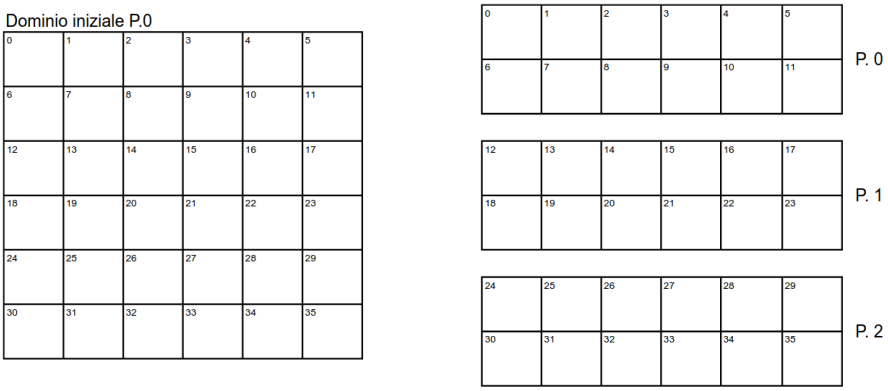

Figura 3.1: Esempio di divisione di un dominio N=6 in 3 processi

Viene dunque eseguita la fase pari sulle porzioni di dominio ricevute, al termine della quale ogni processo invia la prima riga del risultato al processo di indice inferiore e ricevere l'ultima dal processo di indice superiore come riportato in figura 3.2.

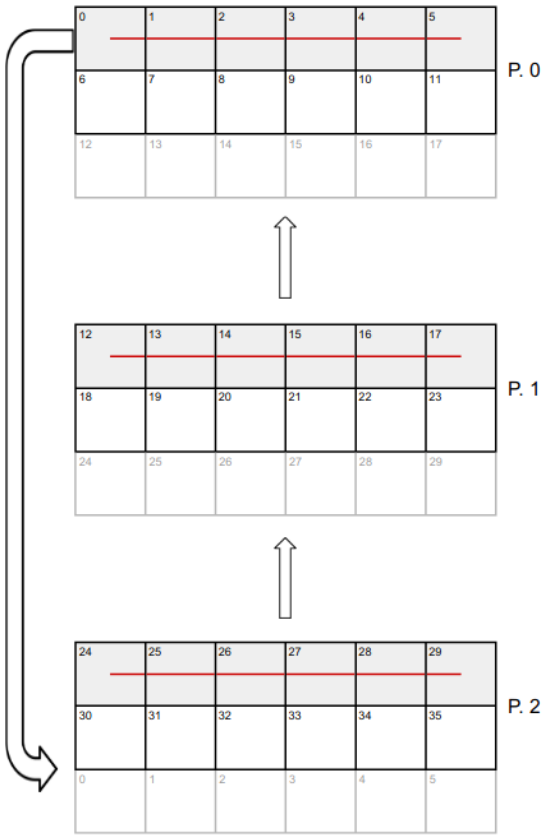

Figura 3.2: Ogni processo invia e riceve una riga

Dopo aver ricevuto la nuova riga ogni processo esegue la fase dispari sul proprio dominio escludendo la porzione di dati prima inviata.

In fine il dominio completo viene ricostruito nel processo 0 tramite l'uso della funzione `MPI_Gatherv`.

Il programma termina dopo l'esecuzione ripetuta di questo processo per N volte.

3.2 Valutazione delle prestazioni

Con questa soluzione a memoria distribuita i processi coinvolti richiedono necessariamente lo scambio di dati tra di essi, rallentando la computazione rispetto alla soluzione a memori condivisa sopra proposta. Di seguito viene riportato il grafico dello speedup 3.3 e della strong scaling efficiency 3.4

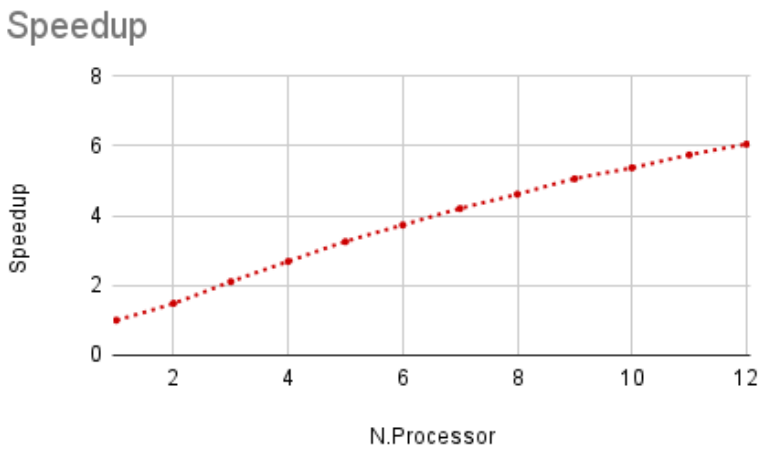

Figura 3.3: Grafico speedup ottenuto su server isi-raptor

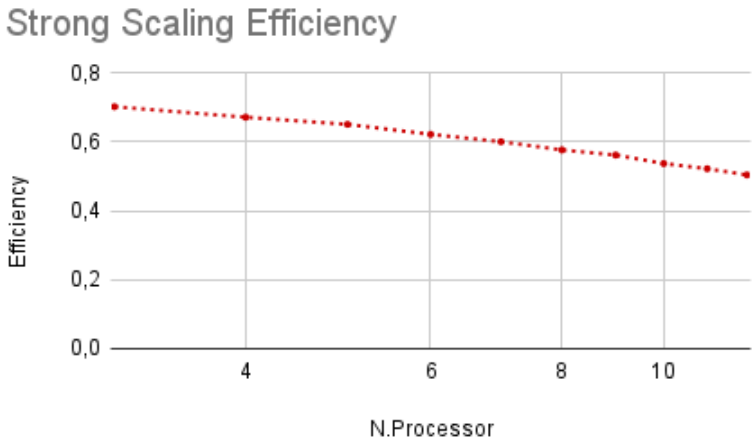

Figura 3.4: Grafico strong scaling efficiency ottenuto su server isi-raptor

4 Conclusioni

La versione a memoria distribuita risulta essere notevolmente più lenta a causa della necessità dello scambio di dati, il problema proposto sembra più indicato ad una risoluzione tramite memoria condivisa.