Specifica del progetto

Lo scopo di questo progetto è l'implementazione mediante MPI di un algoritmo parallelo per architetture a memoria distribuita per calcolare la distribuzione della distanza angolare di n stelle nel numero di intervalli definiti dall'utente.Ogni coppia di punti ha una una sola distanza angolare, la distanza angolare tra i punti i e j è uguale alla distanza angolare tra i punti j e i, dunque le distanze da calcolare sono $N(N-1)/2$, dove N rappresenta il numero di stelle.
L'algoritmo deve soddisfare i seguenti passi:

- Determinare le distanze angolari tra i punti, utilizzando più processori.
- determinare la distribuzione di θ_{ij} distanze angolari per $0 \leq i < j < N$,
- la distribuzione deve avvenire nei K sottointervalli indicati dall'utente. Tale che ogni sotto intervallo k contenga le distanze comprese tra $\pi k/K$, e $\pi(k+1)/K$ quindi, il k-esimo intervallo contiene: $I_k = [\pi k/K, \pi(k+1)/K)$.

Analisi del problema

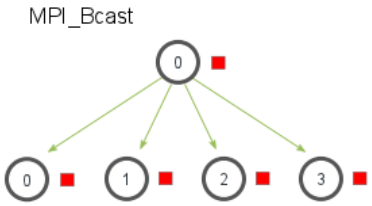
Il problema può essere dunque suddiviso in più fasi per favorire la parallelizzazione delle operazioni:

1. Invio delle stelle acquisite dal processo principale verso gli altri, ognuno calcolerà una parte delle distanze angolari, utilizzando una array per immagazzinare le somme delle distanze che ricadono negli intervalli
2. Si sommano tutti i contributi dei processori per ottenere l'istogramma finale, contenendo a questo punto tutte le distanze angolari distribuite tra i K intervalli

Il programma è strutturato in modo tale da poter parallelizzare il calcolo utilizzando qualsiasi numero di processi e qualsiasi sia il numero degli intervalli indicati.

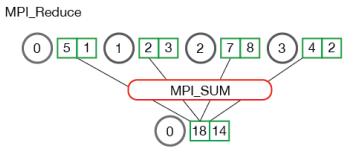
Scelte progettuali

Fase I



In una prima fase, vengono inviate le stelle a tutti i processi *MPI_Bcast*. Si è scelto di rappresentare i dati come una matrice *MxM* triangolare, escludendo la diagonale (poichè $\theta_i = \theta j$) rappresentando così tutte le distanze tra le coppie di stelle. Ogni processo dunque, percorrerà solo una parte della matrice in base alla porzione di distanze assegnatogli. Ogni processo conserverà poi i risultati in un'array che rappresenta l'istrogramma finale.

Fase II



Nella seconda fase invece, si utilizza la *MPI_Reduce* per sommare tutti gli array locali elemento per elemento. Alla fine dell'operazione otteniamo l'istogramma finale, contenente tutte le distanze angolari.

Implementazione

Sono state create dapprima le funzioni di acquisizione dei dati che riempiono apposite strutture invocate dal processo principale.

Strutture

È stata creata la struttura (e il tipo) *star* per contenere i valori delle dimensioni delle stelle costituita da 3 double (`x,y,z`).

Fase iniziale

Nella fase iniziale vengono calcolati tutte le distanze da suddividere e quelle assegnate ad ogni processo:

```
    unsigned long  total_distances = (stars_n * (stars_n - 1)) / 2;
    unsigned long  distances_to_compute = (total_distances / numtasks)
                                        + (taskid < total_distances % numtasks);
```

Viene confrontato il resto della divisione per il numero di processi, e aggiunta 1 stella (il risultato del confronto) se il resto è maggiore del numero del task, così ripartendo equamente le stelle rimanenti.
In questa maniera ci assicuriamo che non ci siano distanze rimanenti da calcolare, escludendo la possibilità di limitare l'input del programma.
Vengono poi calcolate quindi la distanza iniziale e finale per ogni processo (`localstart,localstop`) .
Si allocano le strutture dati necessarie:

- `stars`, riempita con le stelle, solo dal processo principale tramite `load_tree`
- `histogram_local`, che conterrà le distanze calcolate dai singoli processi
- `histogram`, allocato solo nel processo principale, che raccoglierà i contributi dai processori

Fase I

	$star_1$	$star_2$	$star_3$	$star_4$
$star_1$		θ_{12}	θ_{13}	θ_{14}
$star_2$			θ_{23}	θ_{24}
$star_3$				θ_{34}
$star_4$				

Ogni processo cicla nel proprio intervallo di distanze calcolato. Per evitare un doppio ciclo `for` durante il calcolo e quindi risparmiare potenza computazionale è stato favorito un approccio algebrico al problema:
Si possono immaginare le distanze disposte in una matrice MxM triangolare dove M rappresenta il numero delle stelle, e le celle contengono le distanze, inoltre la diagonale non viene presa in considerazione, quindi nel codice per convenienza viene istanziato come $m = stars_n - 1$.
Vengono suddivise le porzioni della matrice tra i processi, leggendo i valori dal basso verso l'alto per ottimizzarne la velocità.

```
    long double gt_i = M * (M + 1) / 2 - 1 - i;
    long double K = floor((sqrt(8 * gt_i + 1) - 1) / 2);

    double distance = star_angular_distance(
                stars[(unsigned long) (M - 1 - K)],// Resolves the star in the row
                stars[
                          (unsigned long)
                          (
                               i - M * (M + 1) / 2 + (K + 1) * (K + 2) / 2
                          )
                ] // Resolves the star in the column
                );
```

Ci saranno $M-1$ elementi da calcolare per la prima riga, $M-2$ per la seconda e così via, per un totale di $M-1+(M-2)+...+1 = M(M-1)/2$ elementi.
È più facile iniziare la computazione dall'ultima cella della matrice (che rappresenta la distanza), poichè non c'è nessuna distanza da computare (la diagonale è da escludere) e a salire il numero di celle da calcolare aumenta di 1 elemento alla volta (nella 4° riga, ci sono 4 elementi da escludere, nella 3° riga, ci sono 3 elementi da escludere, e così via), le ultime K righe prendono $K(K-1)/2$ posizioni.
Supponendo ora di avere un indice i (rappresentante la distanza angolare i-esima), vogliamo ottenere le stelle corrispondenti di cui calcolare la distanza angolare.
Ci saranno $M(M-1)/2-1-i$ elementi con posizioni più grandi di i, chiamiamolo i', si vorrà trovare il più grande valore tale che $k(k-1)/2 \leq i'$. $k(k-1)/2 \leq i'$ si esplicita in $k \leq (\sqrt{(8i'+1)}-1)/2$.
Chiamiamo ora K come il più grande tra i k, ci saranno quindi K righe dopo (su $M-1$ righe), per la quale `row_index(i,M)` è $M-2-K$.
Per le colonne: dato che la riga inizia a $(K+1)(K+2)/2$ dalla fine, sottraiamo la posizione della riga da i: $i - m*(m+1)/2 + (K+1)*(K+2)/2$ dove m è il numero delle stelle
Per ottenere poi l'intervallo corrispondente, si divide la distanza ottenuta per la dimensione dell'intervallo, la parte intera corrisponderà al numero dell'intervallo e quindi si incrementerà il contatore corrispondente.

Fase II

Nella seconda fase, vengono raccolti i contributi di ogni processo, inseriti in `histogram_local` , riducendoli tramite la `MPI_Reduce` in `histogram` nel processo principale:

```
MPI_Reduce(histogram_local,
           histogram,
           intervals,
           MPI_UNSIGNED,
           MPI_SUM,
           mpi_root,
           MPI_COMM_WORLD);
```

Inoltre, nel progetto viene aggiunta la funzione `draw_histogram` per generare i dati risultanti e il plot utilizzando `gnuplot` alla fine del calcolo. Al termine dell'esecuzione venogno generati:

- `star_angular_distance_distribution.dat` contenente i risultati dell'istogramma
- `star_angular_distance_distribution.png` il grafico dell'elaborato con gnuplot (necessario avrlo installato nella macchina

Risultati

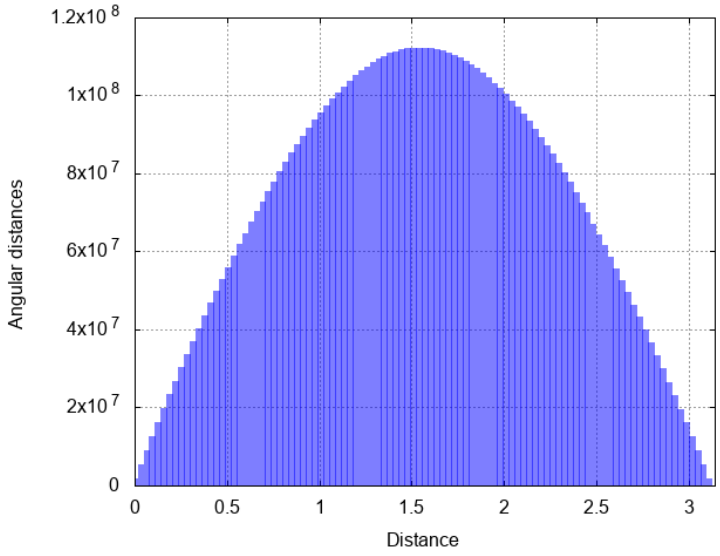


Figure 1: Istogramma finale: nell'asse delle x sono indicati i valori delle distanze, da 0 a π, nell'asse delle y quante distanze ricadono nell'intervallo

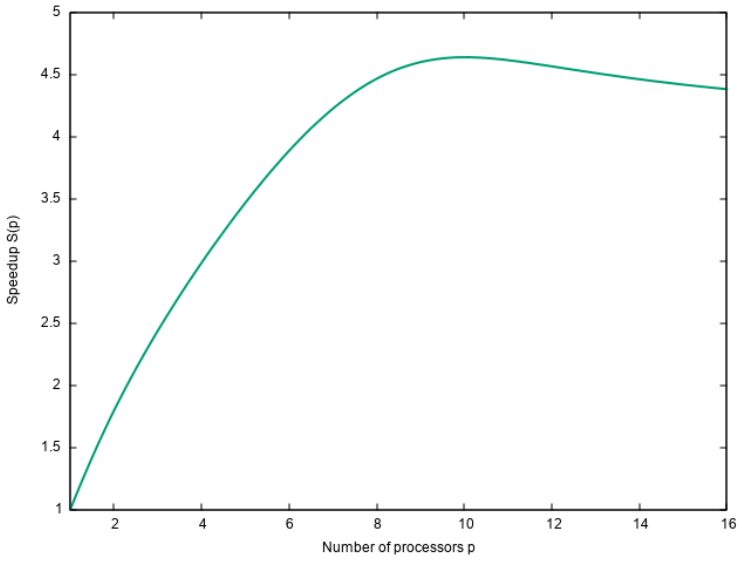


Figure 2: Grafico Speedup: i test sono stati effettuati su un octa-core con tecnologia Hyper-Thread, si può notare infatti che tra i 6 e gli 8 core l'incremento non è stato sensibile come lo sarebbe stato in un vero octa-core

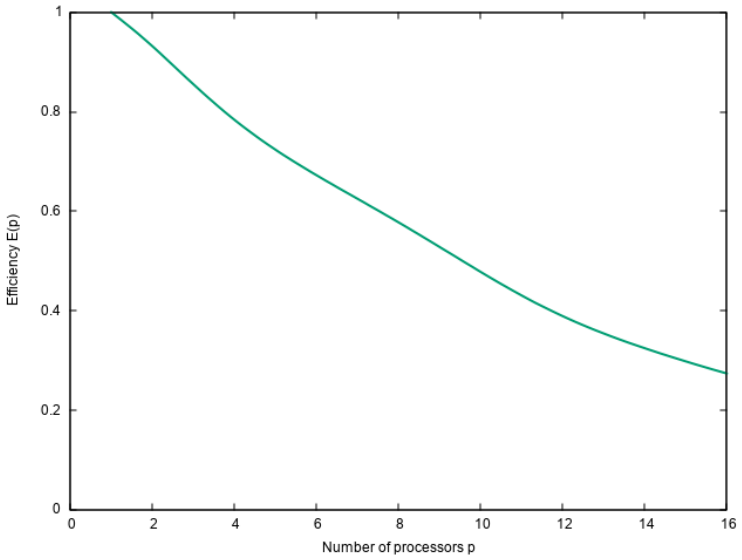


Figure 3: Grafico dell'efficienza