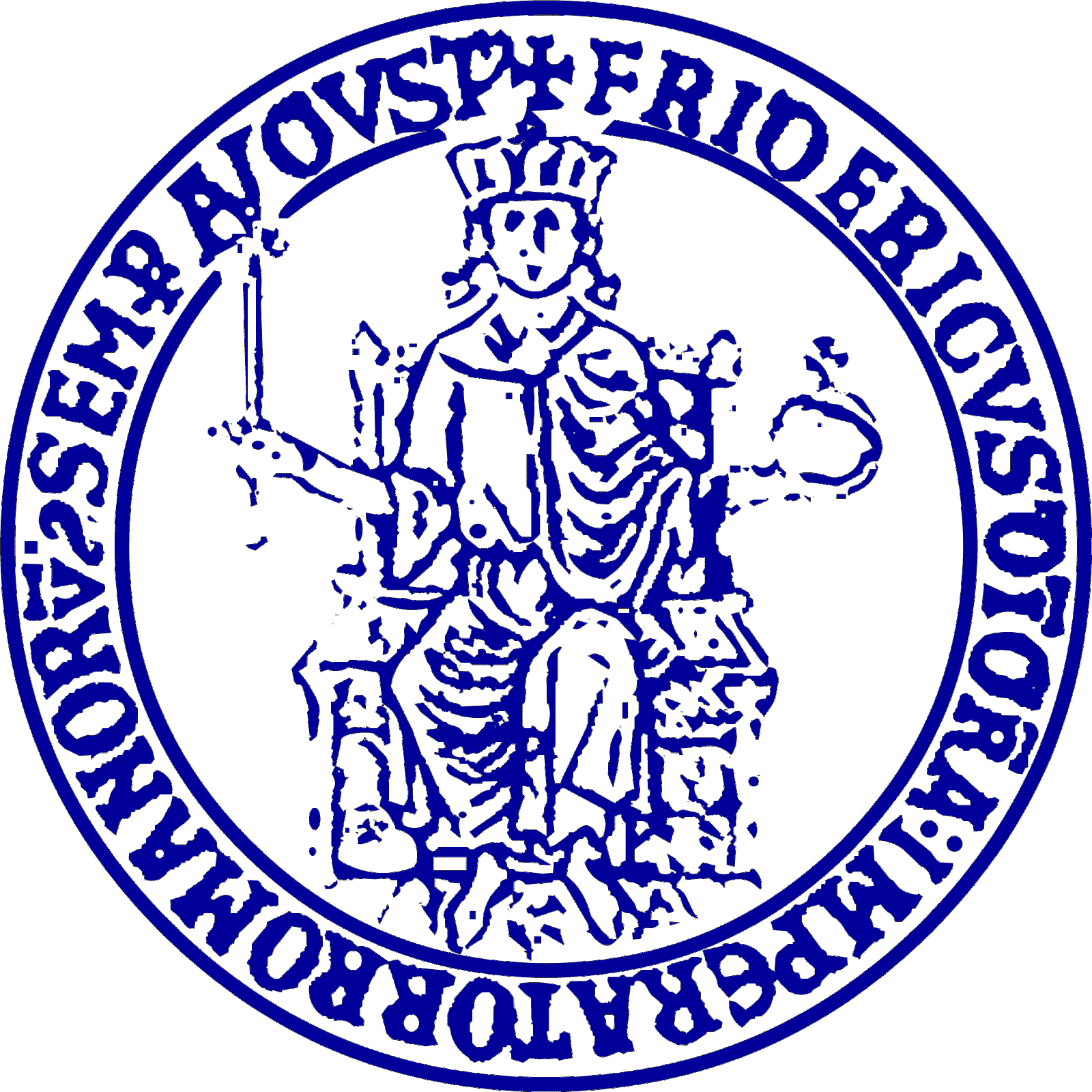
**UNIVERSITÀ DEGLI STUDI DI NAPOLI FEDERICO II**

SCUOLA POLITECNICA E DELLE SCIENZE DI BASE

DIPARTIMENTO DI INGEGNERIA ELETTRICA E TECNOLOGIE DELL’INFORMAZIONE



CORSO DI LAUREA IN MAGISTRALE INFORMATICA

Documentazione esterna Parallel Distributed Computing per il calcolo del prodotto matrice per matrice in parallelo

Studente: **Giuseppe Piccolo** Studente: **Giuliano Taglialatela**

Matricola: **N97000415** Matricola: **N97000404**

Anno accademico 2022/2023

Sommario

[1. Definizione e analisi del problema 4](#_Toc122442721)

[1.1 Traccia dell’elaborato 4](#_Toc122442722)

[1.2 Analisi del problema 4](#_Toc122442723)

[2. Descrizione dell’algoritmo 4](#_Toc122442724)

[2.1 Introduzione 4](#_Toc122442725)

[2.2 Ottenimento dati forniti in ingresso 5](#_Toc122442726)

[2.3 Inizializzazione matrici e griglia di processori 7](#_Toc122442727)

[2.4 Distribuzione sottomatrici ai processori 7](#_Toc122442728)

[2.5 Esecuzione algoritmo **Broadcast Multiply Rolling** 9](#_Toc122442729)

[2.6 Unione sotto-blocchi prodotti dall’algoritmo 10](#_Toc122442730)

[3. Input e Output 11](#_Toc122442731)

[4. Indicatori di errore 12](#_Toc122442732)

[5. Subroutine 13](#_Toc122442733)

[5.1 Subroutine di MPI 13](#_Toc122442734)

[5.2 Subroutine personali 15](#_Toc122442735)

[6. Analisi dei tempi 20](#_Toc122442736)

[7. Esempi d’uso 25](#_Toc122442737)

# 1. Definizione e analisi del problema

## 1.1 Traccia dell’elaborato

Sviluppare un algoritmo per il calcolo del prodotto matrice-matrice, in ambiente di calcolo parallelo su architettura MIMD a memoria distribuita, che utilizzi la libreria MPI.

## 1.2 Analisi del problema

Il problema che deve essere affrontato, nello specifico, è il prodotto matrice x matrice, dove entrambe le matrici sono quadrate e hanno la stessa dimensione. Formalmente, siano AÎÂmxm e BÎÂmxm matrici, ciò che vogliamo eseguire è l’operazione A x B. Quindi bisogna eseguire il prodotto riga per colonna tra A e B per ottenere la matrice prodotto. In un sistema di calcolo non parallelo, l’implementazione di quest’algoritmo è abbastanza semplice e banale. Nel nostro caso, l’algoritmo deve essere implementato in un sistema di calcolo parallelo a memoria distribuita su architettura MIMD. La prima difficoltà nell’implementare il programma, è la suddivisione del problema in *p* sotto-problemi, dove *p* indica il numero di processori disponibili nel nostro sistema di calcolo parallelo. La seconda difficoltà è quella di eseguire l’operazione matrice x matrice dei vari sotto-problemi distribuiti ai *p* processori in parallelo e gestire tutti le comunicazioni necessarie per procedere con il calcolo. Come operazione finale, una volta risolto tutti i sotto-problemi dei *p* processori e ricavato tutti i prodotti parziali, il tutto deve essere messo assieme per ricavare il prodotto finale. Quindi, da come si è potuto notare, l’implementazione della versione parallela di quest’algoritmo non è banale. Specificatamente, l’implementazione di quest’algoritmo utilizzerà la strategia del **Broadcast Multiply Rolling**, di conseguenza anche la suddivisione del problema in sotto-problemi sarà adattata a questa strategia. Tutto ciò verrà approfondito nel capitolo successivo.

# 2. Descrizione dell’algoritmo

## 2.1 Introduzione

Come già accennato, la strategia utilizzata per eseguire il prodotto tra matrici è quella del **Broadcast Multiply Rolling**. Ci sarà una prima fase in cui il programma si occuperà di ottenere i dati in ingresso forniti dall’utente, tra cui i dati necessari per la creazione di una griglia di processori. Questa struttura dati permette di distribuire i processori del nostro sistema di calcolo parallelo in una griglia logica bidimensionale, assegnandoli delle coordinate. La griglia ci servirà per facilitare la suddivisione del problema e per l’esecuzione dell’operazione in parallelo. Per sfruttare la strategia **Broadcast Multiply Rolling**, anche la griglia dovrà essere quadrata. Fatto ciò, date le due matrici AÎÂmxm e BÎÂmxm da moltiplicare, queste verranno suddivise in blocchi quadrati. Il numero di blocchi dipenderà dal numero di righe e dal numero di colonne che avrà la griglia create precedentemente. A ogni processore verrà assegnato un blocco della matrice A e un blocco della matrice B, disposti nella stessa posizione del processore *p-esimo* posizionato nella griglia. Dopodiché, verrà applicato l’algoritmo del **Broadcast Multiply Rolling**, che si occuperà di ricavare i sotto-blocchi della matrice prodotta da A e B. L’operazione finale da eseguire è quella di unire tutti i sotto-blocchi ricavati dall’algoritmo, così da ottenere la matrice prodotta dall’operazione A x B. Ricapitolando, il programma è strutturato in cinque fasi, specificatamente:

* Ottenimento dati forniti in ingresso;
* Inizializzazione matrici e griglia di processori;
* Distribuzione sottomatrici ai processori;
* Esecuzione algoritmo **Broadcast Multiply Rolling**;
* Unione sotto-blocchi prodotti dall’algoritmo;

Nelle seguenti sezioni verrà approfondito ogni fase dell’operazione.

## 2.2 Ottenimento dati forniti in ingresso

Per una corretta esecuzione dell’algoritmo, c’è bisogno di fornire al programma una serie di parametri a riga di comando. I parametri necessari per il funzionamento dell’algoritmo sono cinque:

* Parametro **righe griglia processori**. Questo parametro deve essere un intero positivo diverso da zero e moltiplicato con il parametro colonne griglia processori il risultato deve essere uguale al numero dei processori. Indica il numero di righe che dovrà avere la griglia di processori. Se non verranno rispettate queste condizioni sarà lanciata un’eccezione e l’esecuzione del programma verrà terminata prima del calcolo del prodotto tra matrici.
* Parametro **colonne griglia processori.** Questo parametro deve essere un intero positivo diverso da zero e moltiplicato con il parametro righe griglia processori il risultato deve essere uguale al numero dei processori. Indica il numero di colonne che dovrà avere la griglia di processori. Se non verranno rispettate queste condizioni sarà lanciata un’eccezione e l’esecuzione del programma verrà terminata prima del calcolo del prodotto tra matrici.
* Parametro **grandezza matrice**. Questo parametro deve essere un intero positivo maggiore di uno. Indica la dimensione delle matrici che verranno create e inizializzate. Siccome le matrici sono quadrate, il valore rappresenterà il numero di righe e il numero di colonne di entrambe le matrici. Se non verrà rispettata la condizione precedente sarà lanciata un’eccezione e l’esecuzione del programma verrà terminata prima del calcolo del prodotto tra matrici.
* Parametro **nascondi matrici di input**. Questo parametro deve essere un valore booleano uguale a zero o uno. Indica se si vuole stampare le matrici generate dal programma prima del calcolo del loro prodotto. Ciò è utile nel caso in cui le matrici sono molto grandi e si vuole solamente prendere i tempi dell’algoritmo. Se non verrà rispettata la condizione precedente sarà lanciata un’eccezione e l’esecuzione del programma verrà terminata prima del calcolo del prodotto tra matrici.
* Parametro **nascondi matrice di output**. Questo parametro deve essere un valore booleano uguale a zero o uno. Indica se si vuole stampare la matrice ottenuta dall’algoritmo **Broadcast Multiply Rolling**. Ciò è utile nel caso in cui la matrice è molto grande e si vuole solamente prendere i tempi dell’algoritmo. Se non verrà rispettata la condizione precedente sarà lanciata un’eccezione e l’esecuzione del programma verrà terminata prima del calcolo del prodotto tra matrici.

Nel programma sono state create tutta una serie di funzioni che si occupano di ottenere i parametri elencati precedentemente e di controllare se le condizioni sono rispettate. In caso negativo, le funzioni si occuperanno di terminare l’esecuzione del programma. Di seguito è mostrato lo pseudocodice di questa prima parte del programma:

if (argc < 4)

print(“Error! Not enough argv arguments”);

exit(-1);

int grid\_rows = argv[1];

if(grid\_rows <= 0)

print(“Error! Grid rows parameter has incorrect value”);

exit(-1);

int grid\_columns = argv[2];

if(grid\_columns <= 0)

print(“Error! Grid columns parameter has incorrect value”);

exit(-1);

if(grid\_rows \* grid\_columns != processors)

print(“Error! grid\_rows \* grid\_columns != processors”);

exit(-1);

int size\_matrices = argv[3];

if(size\_matrices <= 1)

print(“Error! Size matrices parameter has incorrect value”);

exit(-1);

int hide\_input\_matrices = argv[3];

if(hide\_input\_matrices!= 0 and hide\_input\_matrices != 1)

print(“Error! Hide input matrices parameter has incorrect value”);

exit(-1);

int hide\_output\_matrix = argv[4];

if(hide\_output\_matrix!= 0 and hide\_output\_matrix!= 1)

print(“Error! Hide output matrix parameter has incorrect value”);

exit(-1);

## 2.3 Inizializzazione matrici e griglia di processori

Una volta ottenuti i parametri e verificato che il loro valore sia corretto, si può passare alla fase due del programma. Ovvero quella dell’inizializzazione delle matrici, su cui eseguire la moltiplicazione matrice x matrice, e dell’inizializzazione della griglia di processori. Le matrici verranno allocate dinamicamente e inizializzate con dei valori casuali nell’intervallo [-100, 100] con dimensione specificata nell’argomento passato a riga di comando. In seguito, sarà creata la griglia dei processori con il numero righe e colonne specificato nei parametri. È stata utilizzata la funzione fornita a lezione per creare la griglia dei processori. Di seguito è mostrato lo pseudo codice di questa fase:

double \*matrix\_a = alloc(rows \* columns);

double \*matrix\_b = alloc(rows \* columns);

int row, column;

for(row = 0; row != rows; ++row)

for(column = 0; column != columns; ++column)

matrix\_a[row \* columns + column] = get\_random\_number();

matrix\_b[row \* columns + column] = get\_random\_number();

MPI\_Comm grid\_comm;

MPI\_Comm grid\_rows\_comm;

MPI\_Comm grid\_columns\_comm;

int coordinates[2];

create\_grid(&grid\_comm, &grid\_rows\_comm, &grid\_columns\_comm,

rank\_processor, grid\_rows, grid\_columns, coordinates);

## 2.4 Distribuzione sottomatrici ai processori

Inizializzate le matrici e le griglia dei processori, il passo successivo è quello di distribuire le matrici ai processori. Specificatamente, le matrici saranno suddivise in *d* x *q* sotto-blocchi, dove *d* indica il numero di righe della griglia di processori e *q* indica il numero di colonne della griglia di processori. Ai processori saranno assegnati i sotto-blocchi delle matrici in questo modo: sia *p* un processore posizionato alla riga i-esima e colonna j-esima. Gli verranno assegnati il sotto-blocco della matrice A posizionato alla riga i-esima e colonna j-esima e il sotto-blocco della matrice B posizionato alla riga i-esima e colonna j-esima. Quindi ogni processore, inizialmente avrà un sotto-blocco della matrice A e un sotto-blocco della matrice B. Nella figura di seguito è mostrato un esempio di suddivisione nel caso di una griglia 3 x 3:

Matrice B

Matrice A

B0

B1

B2

B3

B4

B5

B6

B7

B8

A8

A7

A0

A1

A2

A5

A4

A3

A6

Processo P0

A0 B0

Processo P1

A1 B1

Processo P2

A2 B2

Distribuzione in sotto-blocchi

Processo P5

A5 B5

Processo P4

A4 B4

Processo P3

A3 B3

Processo P8

A8 B8

Processo P7

A7 B7

Processo P6

A6 B6

Di seguito è mostrato lo pseudo codice della distribuzione delle matrici:

if(rank\_processor == 0)

{

      for(int i = 1; i != processors; ++i)

      {

double \*submatrix\_a = get\_submatrix(i, matrix\_a);

double \*submatrix\_b = get\_submatrix(i, matrix\_b);

send\_submatrix(i, submatrix\_a);

send\_submatrix(i, submatrix\_b);

}

}

else

{

double \*submatrix\_a = alloc(size\_submatrix \* size\_submatrix);

double \*submatrix\_b = alloc(size\_submatrix \* size\_submatrix);

receive\_submatrix(0, submatrix\_a);

receive\_submatrix(0, submatrix\_b);

}

## 2.5 Esecuzione algoritmo **Broadcast Multiply Rolling**

Una volta eseguita la distribuzione i sotto-blocchi delle matrici, questi possono essere usati per eseguire il calcolo in parallelo del prodotto matrice x matrice. Inizialmente, ci troveremo nella situazione in cui ogni processore *p* alla riga i-esima e colonna j-esima della griglia, ha un blocco della matrice A, posizionato alla riga i-esima e colonna j-esima della matrice A, e un blocco della matrice B, sempre posizionato alla riga i-esima e colonna j-esima della matrice B. Per eseguire il prodotto parziale, ogni processore *p* dovrebbe moltiplicare l’intera riga i-esima di sotto-blocchi della matrice A, con l’intera colonna di sotto-blocchi della matrice B. Il problema è che ogni processore, alla prima iterazione, possederà solamente un blocco della matrice A e un solo blocco della matrice B. Per risolvere questo inconveniente, viene utilizzata la strategia del **Broadcast Multiply Rolling**. Questa strategia consiste nell’eseguire dei broadcast sulle righe della griglia di processori, per condividere i sotto-blocchi della matrice A, e di inviare il sotto-blocco della matrice B al processore della colonna corrente che si trova sopra di esso, per ogni processore *p* della griglia. Specificatamente. Alla prima iterazione, i processori che appartengono alla diagonale della griglia, eseguiranno un broadcast, sulle righe correnti, per condividere il proprio sotto-blocco della matrice A con gli altri processori che appartengono alla stessa riga. Fatto ciò, ogni processore moltiplica, il sotto-blocco della matrice A, appartenente alla diagonale della matrice, con il sotto-blocco B che già possedeva. Alla seconda iterazione, la diagonale della matrice A trasla di una posizione, ovviamente per fare ciò, bisogna considerare che il lato destro della matrice sia collegato con il lato sinistro, così da considerarla contigua. Successivamente, i processori della diagonale traslata, eseguiranno un nuovo broadcast sulle righe della griglia per condividere il sotto-blocco della matrice A con i processori appartenenti alle stesse righe. Dopodiché, ogni processore invierà il sotto-blocco della matrice B che possiedono al processore posizionato sopra di esso sulla stessa colonna e riceverà il sotto-blocco della matrice B posseduto dal processore posizionato sotto di esso. Ovviamente per fare ciò, bisogna considerare che il lato inferiore della matrice sia collegato con il lato superiore, così da considerarla contigua. Successivamente, si eseguirà il prodotto tra il sotto-blocco della matrice A, appartenente alla diagonale della matrice e il sotto-blocco della matrice B, ricevuto dal processore inferiore, sommato con la sottomatrice calcolata al passo 1. Il processo re-itera per *m* volte, dove *m* indica la dimensione della matrice quadrata. Di seguito è mostrato lo pseudo-codice dell’algoritmo:

for(int i = 0; i != m; ++i)

{

update\_sender\_coordinates(sender\_coordinates, i)

if(i == 0)

{

if(coordinates[0] == coordinates[1])

{

sender\_coordinates[1] = coordinates[1];

copy(matrix\_to\_broadcast, submatrix\_a);

}

broadcast(matrix\_to\_broadcast, sender\_id);

block = execute\_matrix\_multiplication(matrix\_to\_broadcast,

submatrix\_b);

}

else

{

if(sender\_coordinates[1] == coordinates[1])

copy(matrix\_to\_broadcast, submatrix\_a);

broadcast(matrix\_to\_broadcast, sender\_id);

send(submatrix\_b, destination\_submatrix\_b\_id);

receive(submatrix\_b, sender\_submatrix\_b\_id);

}

}

## 2.6 Unione sotto-blocchi prodotti dall’algoritmo

Dopo l’esecuzione dell’algoritmo, come risultato finale, avremo che ogni processore possederà un blocco prodotto dalle varie moltiplicazioni e somme tra sotto-blocchi delle matrici. Il blocco in questione è una “parte” della matrice C che dovrebbe essere prodotta moltiplicando la matrice A con la matrice B. Nello specifico, ogni processore della griglia posizionato alla riga i-esima e alla colonna j-esima avrà il blocco alla riga i-esima e alla colonna j-esima della matrice C. Quindi, l’operazione finale da eseguire, è quella di unire tutti questi blocchi, in modo tale da formare la matrice C. Di seguito, è mostrato lo pseudo-codice di questa operazione finale:

if(rank\_processor == 0)

{

        copy\_block(product\_matrix, submatrix\_block, 0);

        float \*received\_block = alloc(size\_submatrix \* size\_submatrix);

        for(int i = 1; i != processors; ++i)

        {

            receive\_block(received\_block, i);

            copy\_block(product\_matrix, received\_block, i);

        }

}

    else

        send\_block(submatrix\_block, 0);

# 3. Input e Output

Per eseguire il programma bisogna lanciare l’eseguibile prodotto dalla compilazione, ma non è sufficiente per l’esecuzione corretta di quest’ultimo. Ci sono alcuni parametri che devono essere passati in input al momento del lancio del file eseguibile. Qui è mostrato una rappresentazione generica su come eseguirlo con i parametri da passare in ingresso:

mpiexec -n NUMERO\_PROCESSORI {PERCORSO\_CARTELLA\_DI\_LAVORO}/{NOME\_ESEGUIBILE} NUMERO\_RIGHE\_GRIGLIA NUMERO\_COLONNE\_GRIGLIA DIMENSIONE\_MATRICI NASCONDI\_MATRICI\_GENERATE NASCONDI\_PRODOTTO\_MATRICI

Di seguito sono elencate le descrizioni dei parametri:

* NUMERO\_RIGHE\_GRIGLIA: questo parametro rappresenta il numero di righe che deve avere la griglia di processori, che verrà usata per il calcolo del prodotto matrice x matrice, il valore deve essere un intero positivo maggiore di zero;
* NUMERO\_COLONNE\_GRIGLIA: questo parametro rappresenta il numero di colonne che deve avere la griglia di processori, che verrà usata per il calcolo del prodotto matrice x matrice. Il valore deve essere un intero positivo maggiore di zero;
* DIMENSIONE\_MATRICI: questo parametro rappresenta il numero di righe e di colonne che le matrici A e B, da generare all’inizio del programma, devono avere. Il valore deve essere un intero positivo maggiore di uno;
* NASCONDI\_MATRICI\_GENERATE: questo parametro è di tipo binario, ovvero, può assumere come valore o 0 oppure 1. Se il suo valore sarà uguale a 0, prima che il software avvii l’esecuzione della moltiplicazione, stamperà le matrici A e B. Se il suo valore sarà uguale a 1, tutto ciò non verrà stampato. Questo può essere comodo nel caso in cui si hanno due matrici di grande dimensione, di conseguenza la stampa di questi due oggetti risulterebbe inconveniente. Quindi si può decidere di visionarli o meno prima dell’esecuzione;
* NASCONDI\_PRODOTTO\_MATRICI: questo parametro è di tipo binario, ovvero, può assumere come valore o 0 oppure 1. Se il suo valore sarà uguale a 0, prima che il software avvii l’esecuzione della moltiplicazione, stamperà il prodotto C, dell’operazione tra le matrici A e B. Se il suo valore sarà uguale a 1, tutto ciò non verrà stampato. Questo può essere comodo nel caso in cui il prodotto ha una grande dimensione e si vuole solo visionare il tempo impiegato dall’algoritmo per eseguire la somma. Quindi si può decidere di visionarli o meno prima dell’esecuzione.

Nel caso in cui, uno o più parametri vengano omessi, oppure il loro valore non è corretto, il programma stamperà un messaggio di errore e l’esecuzione verrà terminata restituendo il valore -1.

Per quanto riguarda l’output del programma dipenderà dagli ultimi due parametri. Nello specifico, nel caso in cui i parametri NASCONDI\_MATRICI\_GENERATE e NASCONDI\_PRODOTTO\_MATRICI siano uguale a zero, le matrici generate casualmente verranno stampati prima del calcolo del prodotto e, dopo aver eseguito il prodotto tra questi due oggetti, verrà stampato anche il risultato finale. Mentre nel caso in cui la variabile NASCONDI\_MATRICI\_GENERATE sarà impostato a uno, la stampa iniziale delle matrici generate casualmente verrà nascosta. Nel caso in cui la variabile NASCONDI\_PRODOTTO\_MATRICI è uguale a uno, la stampa del prodotto finale verrà eseguita. Se entrambi i parametri sono uguali a uno, l’unica stampa che verrà effettuata e quella che riguarda il tempo di esecuzione. Quest’ultimo verrà stampato indipendentemente dai valori dati ai parametri in ingresso. Ovviamente il tempo di esecuzione, verrà stampato se non ci sono stati errori.

# 4. Indicatori di errore

Come già accennato, il software deve gestire alcune situazioni di errore. Questi casi capitano quando vengono passati dei valori errati ai parametri che devono essere passati in ingresso, prima dell’esecuzione del programma oppure quando non si passa un numero sufficiente di argomenti per eseguire il corretto funzionamento dell’algoritmo. Ricordiamo che il numero di argomenti da passare a riga di comando sono cinque e ognuno di essi è gestito differentemente in caso di incorrettezza. Nello specifico:

* Gestione errore **numero insufficiente di argomenti**: in questo caso l’eccezione viene lanciata quando il numero di argomenti passati è inferiore a cinque. Il programma stamperà nel file **.err** il messaggio di errore: “Error! Not enough argv arguments” e successivamente l’esecuzione del programma verrà terminata bruscamente;
* Gestione errore parametro **NUMERO\_RIGHE\_GRIGLIA**: in questo caso l’eccezione viene lanciata quando il valore di questo parametro sarà uguale a un valore minore o uguale a zero. Il programma, una volta intercettato l’errore, stamperà nel file **.err** il messaggio di errore: “Error! Grid rows incorrect value” e successivamente l’esecuzione del programma verrà terminata senza completare il calcolo del prodotto;
* Gestione errore parametro **NUMERO\_COLONNE\_GRIGLIA**: in questo caso l’eccezione viene lanciata quando il valore di questo parametro sarà uguale a un valore minore o uguale a zero. Il programma, una volta intercettato l’errore, stamperà nel file **.err** il messaggio di errore: “Error! Grid columns incorrect value” e successivamente l’esecuzione del programma verrà terminata senza completare il calcolo del prodotto;
* Gestione errore **NUMERO\_RIGHE\_GRIGLIA \* NUMERO\_COLONNE\_GRIGLIA != NUMERO\_PROCESSORI:** in questo caso l’eccezione viene lanciata quando il numero di righe della griglia moltiplicato con il numero delle colonne della griglia da un valore diverso dal numero di processori disponibili. Il programma, una volta intercettato l’errore, stamperà nel file **.err** il messaggio di errore: “Error! grid\_rows \* grid\_columns != processors” e successivamente l’esecuzione del programma verrà terminata senza completare il calcolo del prodotto;
* Gestione errore parametro **DIMENSIONE\_MATRICI**: in questo caso l’eccezione viene lanciata quando il valore di questo parametro sarà uguale a un valore minore o uguale a uno. Il programma, una volta intercettato l’errore, stamperà nel file **.err** il messaggio di errore: “Error! Size matrices incorrect value” e successivamente l’esecuzione del programma verrà terminata senza completare il calcolo del prodotto;
* Gestione errore parametro **NASCONDI\_MATRICI\_GENERATE**: in questo caso l’eccezione viene lanciata quando il valore di questo parametro sarà uguale a un valore diverso da 0 e da 1, siccome è un parametro di tipo binario. Il programma, una volta intercettato l’errore, stamperà nel file **.err** il messaggio di errore: “Error! hide\_input\_matrices must have as value 0 or 1” e successivamente l’esecuzione del programma verrà terminata senza completare il calcolo del prodotto;
* Gestione errore parametro **NASCONDI\_PRODOTTO\_MATRICI**: in questo caso l’eccezione viene lanciata quando il valore di questo parametro sarà uguale a un valore diverso da 0 e da 1, siccome è un parametro di tipo binario. Il programma, una volta intercettato l’errore, stamperà nel file **.err** il messaggio di errore: “Error! hide\_output\_matrix must have as value 0 or 1” e successivamente l’esecuzione del programma verrà terminata senza completare il calcolo del prodotto;

Un’altra situazione che può non portare a una situazione di “stallo” è quando le dimensioni delle matrici sono troppo grandi. Siccome, quando si inviano dei dati il buffer della libreria MPI ha una grandezza limitata, questo porta a bloccare l’esecuzione del programma appena incontra la funzione di MPI per inviare i sotto-blocchi delle matrici agli altri processori.

# 5. Subroutine

Oltre alle funzioni della libreria standard del C, sono state utilizzate alcune funzioni della libreria MPI, per eseguire le operazioni in parallelo. Sono anche state scritte delle “utility” per permettere l’implementazione dell’algoritmo e per migliorare la leggibilità del codice.

## 5.1 Subroutine di MPI

* **int MPI\_Init(int \*argc, char \*\*\*argv)**

**input:**

* + **argc**: puntatore al numero di argomenti;
  + **argv:** puntatore al vettore parametri;

**output:**

* + **IERROR**: solo per fortran;
* **int MPI\_Comm\_rank(MPI\_Comm comm, int \*rank)**

**input:**

* + **comm:** Comunicatore (gestore).

**output:**

* **rank**: rango del gruppo di processi di comm
* **IERROR:** solo per fortran
* **int MPI\_Comm\_size(MPI\_Comm comm, int \*size)**

**input:**

* **comm:** Comunicatore (gestore).

**output:**

* **size:** numero di processori nel gruppo di comm
* **IERROR**: solo per fortran
* **int MPI\_Bcast** **(void \*buffer, int count, MPI\_Datatype datatype,  int root, MPI\_Comm comm)**

**input:**

* + **buffer:** Indirizzo iniziale del buffer
  + **count:** Numero di istanze nel buffer
  + **datatype:** Tipo di dati del buffer
  + **root:** Rango della radice per la trasmissione
  + **comm:** Comunicatore (gestore).
  + **errorcode:** Codice di errore da restituire all’ambiente di invocazione

**output:**

* + **request:** richiesta (gestore)
  + **IERROR**: solo per fortran
* **double MPI\_Wtime()**

**output:**

* **time:** tempo in secondi del tempo trascorso dall’inizio dell’esecuzione
* **int MPI\_Send (const void \*buf, int count, MPI\_Datatype datatype, int dest, int tag, MPI\_Comm comm)**

**input:**

* **buffer:** Indirizzo iniziale del buffer
* **count:** Numero di istanze nel buffer
* **datatype:** Tipo di dati del buffer
* **dest:** rango di destinazione
* **tag:** etichetta del messaggio
* **comm:** Comunicatore (gestore).

**output:**

* **IERROR**: solo per fortran
* **int MPI\_Recv (void \*buf, int count, MPI\_Datatype datatype, int source, int tag, MPI\_Comm comm, MPI\_Status \*status)**

**input:**

* **count:** Numero di istanze nel buffer
* **datatype:** Tipo di dati del buffer
* **source:** Rango del sorgente
* **tag:** etichetta del messaggio
* **comm:** Comunicatore (gestore).

**output:**

* **buf:** Indirizzo iniziale del buffer ricevitore
* **status:** Oggetto status
* **IERROR**: solo per fortran
* **int MPI\_Cart\_create(MPI\_Comm comm\_old, int ndims, const int dims[], const int periods[], int reorder, MPI\_Comm \*comm\_cart)**

**input:**

* **comm\_old**: Gestore comunicatore di input
* **ndims**: Numero delle dimensioni della griglia
* **dims**: Numero di processi per dimensione
* **periods**: Array logico dove si specifica se le dimensioni della griglia sono periodiche
* **reorder**: Il ranking può essere riordinato

**output:**

* **comm\_cart**: Comunicatore con la nuova topologia
* **IERROR**: Solo per fortran, stato errore
* **int MPI\_Cart\_coords(MPI\_Comm comm, int rank, int maxdims, int coords[])**

**input:**

* **comm**: Comunicatore con la struttura cartesiana
* **Rank**: Rank di un processo dentro il Gruppo di comm
* **Maxdims**: Lunghezza del vettore cords nel programma chiamante

**output**:

* **cords**: coordinate del processore specificato
* **IERROR**: Solo per fortran, stato errore
* **int MPI\_Cart\_sub(MPI\_Comm comm, const int remain\_dims[], MPI\_Comm \*comm\_new)**

**input**:

* **comm:** Comunicatore con la struttura cartesiana
* **remain\_dims:** L’i-esimo elemento indica se l’i-esima dimensione deve essere mantenuta.

**output:**

* **comm\_new:** Comunicatore contenente la sotto-griglia che include il processo chiamante
* **IERROR:** Solo per fortran, stato errore
* **int MPI\_Reduce(const void \*sendbuf, void \*recvbuf, int count, MPI\_Datatype datatype, MPI\_Op op, int root, MPI\_Comm comm)**

**input:**

* **sendbuf:** Indirizzo del buffer da inviare
* **count:** Numero di elementi del buffer da inviare
* **datatype:** Tipo di dato degli elementi del buffer da inviare
* **op:** Operazione da eseguire
* **root:** Rank del processore radice
* **comm:** Comunicatore

**output:**

* **recvbuf:** Indirizzo del buffer ricevente
* **request:** Gestore richiesta
* **IERROR:** Solo per fortran, stato errore

## 5.2 Subroutine personali

* **void print\_matrix(float \*matrix, int rows, int columns)**

**descrizione:**

* funzione per stampare a video una matrice di float

**input:**

* **matrix:** matrice da stampare;
* **rows:** numero di righe della matrice
* **columns:** numero di colonne della matrice
* **void create\_grid(MPI\_Comm \*grid, MPI\_Comm \*grid\_rows, MPI\_Comm \*grid\_columns, int rank\_processor, int rows, int columns, int \*coordinates)**

**descrizione:**

* funzione per creare una griglia di processori

**input:**

* **rank\_processor:** rank del processore corrente
* **rows:** righe che deve avere la griglia da creare
* **columns:** colonne che deve avere la griglia da creare

**output:**

* **grid:** comunicatore della griglia da creare
* **grid\_rows:** comunicatore della riga corrente della griglia
* **grid\_columns:** comunicatore della colonna corrente della griglia
* **coordinates:** coordinate del processore corrente
* **void free\_matrix(int \*\*matrix, int rows)**

**descrizione:**

* funzione per deallocare matrici

**input:**

* **matrix:** matrice da deallocare
* **rows:** numero righe della matrice da deallocare
* **float get\_random\_number(float min, float max)**

**descrizione:**

* funzione per generare un numero reale casuale

**input:**

* **min**: limite inferiore del numero casuale da generare
* **max**: limire superiore del numero casuale da generare

**output:**

* **valore di ritorno**: numero casuale generato
* **float\* init\_matrix(int rows, int columns)**

**descrizione:**

* funzione per generare una matrice con numero reali casuali

**input:**

* **rows:** righe della matrice da generare
* **columns:** colonne della matrice da generare

**output:**

* **valore di ritorno:** matrice generata con numeri casuali
* **int get\_grid\_rows\_from\_argv(int argc, char \*argv[])**

**descrizione:**

* funzione per ottenere il parametro grid\_rows dal vettore argv

**input:**

* **argc**: numero di argomenti passati in input
* **argv**: argomenti passati in input a riga di comando

**output:**

* **valore di ritorno**: parametro grid\_rows passato in input a riga di comando
* **int get\_grid\_columns\_from\_argv(int argc, char \*argv[])**

**descrizione:**

* funzione per ottenere il parametro grid\_columns dal vettore argv

**input:**

* **argc**: numero di argomenti passati in input
* **argv**: argomenti passati in input a riga di comando

**output:**

* **valore di ritorno**: parametro grid\_rows passato in input a riga di comando
* **int get\_size\_matrix\_from\_argv(int argc, char \*argv[])**

**descrizione:**

* funzione per ottenere il parametro size\_matrix dal vettore argv

**input:**

* **argc**: numero di argomenti passati in input
* **argv**: argomenti passati in input a riga di comando

**output:**

* **valore di ritorno**: parametro size\_matrix passato in input a riga di comando
* **int get\_hide\_input\_matrices\_from\_argv(int argc, char \*argv[])**

**descrizione:**

* funzione per ottenere il parametro hide\_input\_matrices dal vettore argv

**input:**

* **argc**: numero di argomenti passati in input
* **argv**: argomenti passati in input a riga di comando

**output:**

* **valore di ritorno**: parametro hide\_input\_matrices passato in input a riga di comando
* **int get\_hide\_output\_matrix\_from\_argv(int argc, char \*argv[])**

**descrizione:**

* funzione per ottenere il parametro hide\_output\_matrix dal vettore argv

**input:**

* **argc**: numero di argomenti passati in input
* **argv**: argomenti passati in input a riga di comando

**output:**

* **valore di ritorno**: parametro hide\_output\_matrix passato in input a riga di comando
* **void check\_correctness\_grid\_rows\_columns(int processors, int grid\_rows, int grid\_columns)**

**descrizione:**

* funzione per verificare se grid\_rows \* grid\_columns != number\_of\_processors

**input:**

* **processors**: numero di processori attivi
* **grid\_rows**: numero di righe della griglia
* **grid\_columns**: numero di colonne della griglia
* **void send\_submatrix(int rank\_processor, int processors, int processor\_destination, float \*matrix\_entry\_point, int size\_matrices, int size\_submatrices, MPI\_Comm comm)**

**descrizione:**

* funzione per mandare un sotto-blocco della matrice a un altro processore

**input:**

* **rank\_processor:** rank del processore corrente
* **processors:** numero di processor attivi
* **processor\_destination:** processore che deve avere la sottomatrice
* **matrix\_entry\_point:** puntatore della prima cella della sottomatrice da inviare
* **size\_matrices:** dimensione della sottomatrice
* **comm:** gestore comunicatore
* **int\* get\_matrix\_entry\_points(int processors, float \*matrix, int size\_matrices, int size\_submatrices, MPI\_Comm grid)**

**descrizione:**

* funzione per ottenere un puntatore all’entry point della sottomatrice da inviare

**input:**

* **processors**: numero di processori attivi
* **matrix**: matrice da suddividere in sotto-blocchi
* **size\_matrices**: grandezza della matrice da suddividere in sotto-blocco
* **size\_submatrices**: grandezza delle sottomatrici da inviare
* **grid**: gestore comunicatore

**output**:

* **valore di ritorno**: puntatore della entry point della sottomatrice
* **void distribute\_submatrices(int rank\_processor, int processors, float \*matrix, int size\_matrices, float \*submatrix, int size\_submatrices, MPI\_Comm grid)**

**descrizione:**

* funzione per distribuire una matrice agli altri processori

**input:**

* **rank\_processor:** rank del processore corrente
* **processors:** numero di processori attivi
* **matrix:** matrice da distribuire
* **size\_matrices:** grandezza della matrice da distribuire
* **grid:** gestore comunicatore
* **float\* get\_submatrix\_processor\_root(float \*matrix, int size\_matrix, int size\_submatrix)**

**descrizione:**

* funzione per ottenere la sottomatrice del processore radice

**input:**

* **matrix:** matrice da cui ricavare la sottomatrice
* **size\_matrix:** grandezza della matrice

**output:**

* **valore di ritorno**: sotto-blocco del processore radice

* **void execute\_matrix\_multiplication(float \*matrix\_a, float \*matrix\_b, float \*destination, int size\_matrix)**

**descrizione:**

* funzione per eseguire il prodotto da due matrici

**input:**

* **matrix\_a:** prima matrice da moltiplicare
* **matrix\_b:** seconda matrice da moltiplicare
* **size\_matrix:** grandezza delle matrici

**output:**

* **destination:** matrice in cui memorizzare il risultato
* **float\* get\_block\_partial\_product(float \*submatrix\_a, float \*submatrix\_b, int size\_submatrix, int grid\_rows, int grid\_columns, int \*coordinates, MPI\_Comm grid\_rows\_comm, MPI\_Comm grid\_columns\_comm)**

**descrizione:**

* funzione per eseguire l’algoritmo **Broadcast Multiply Rolling**

**input:**

* **submatrix\_a:** sottomatrice della matrice A posseduta dal processo corrente
* **submatrix\_b:** sottomatrice della matrice B posseduta dal processo corrente
* **size\_submatrix:** dimensione delle sottomatrici
* **grid\_rows:** righe della griglia di processori
* **grid\_columns:** colonne della griglia di processori
* **coordinates:** coordinate del processore corrente
* **grid\_rows\_comm:** comunicatore righe della griglia
* **grid\_columns\_comm:** comunicatore colonne della griglia

**output:**

* **valore di ritorno:** sotto-blocco del prodotto tra matrici
* **void copy\_block(float \*matrix, int size\_matrix, float \*block, int size\_block, int row\_stride, int column\_stride)**

**descrizione:**

* funzione per copiare un sotto-blocco nella matrice nella posizione corretta

**input:**

* **size\_matrix:** dimensione della matrice data in input
* **block:** sotto-blocco da copiare nella matrice
* **size\_block:** grandezza sotto-blocco
* **row\_stride:** passo che ci deve essere tra una riga e un’altra
* **column\_stride:** passo che ci deve essere tra una colonna e un’altra

**output:**

* **matrix**: matrice con il sotto-blocco copiato al suo interno
* **void get\_multiplication\_product(int rank\_processor, int processors, float \*product\_matrix, float size\_product\_matrix, float \*submatrix\_block, float size\_submatrix, MPI\_Comm grid\_comm)**

**descrizione:**

* funzione per unire tutti i sotto-blocchi generati dagli altri processori

**input:**

* **rank\_processor:** rank del processore corrente
* **processors:** numero di processor attivi
* **size\_product\_matrix:** grandezza della matrice passata in input
* **submatrix\_block:** sotto-blocco da copiare nella matrice prodotto finale
* **size\_submatrix:** grandezza del sotto-blocco della matrice
* **grid\_comm:** gestore comunicatore

**output:**

* **product\_matrix:** matrice dove verranno uniti tutti i sotto-blocchi

# 6. Analisi dei tempi

## 6.1 Analisi preliminare con matrici 12x12

Prima di mostrare i tempi presi dal cluster al variare della dimensione del problema, in questa sezione verrà introdotta una analisi preliminare del numero di istruzioni che vengono eseguite sequenzialmente e in parallelo. Il caso preso in considerazione è quello in cui si hanno due matrici 12x12. L’analisi può essere generalizzata e applicata anche agli altri casi. Sia p il numero di processori che vanno a formare una griglia quadrata q x q, i sotto-blocchi delle matrici da moltiplicare, saranno delle sottomatrici di dimensione (n / q) x (n / q), dove n è la dimensione della matrice quadrata. Quindi ogni processore dovrà eseguire delle moltiplicazioni e delle somme tra le componenti di queste sottomatrici. Eseguendo il tempo il calcolo del parametro T(1) e T(p) avremo che:

* T(1) = n2 \* (2 \* n – 1) \* tcalc;
* T(p) = (n / q)2 \* (2 \* (n / q) – 1) \* log2(p) \* tcalc + (n / q)2 \* (log2(p) - 1) \* tcalc;

Quindi a partire da queste formule è possibile calcolarsi lo speed up:

* S(p) = (n2 \* (2 \* n – 1) \* tcalc) / ((n / q)2 \* (2 \* (n / q) – 1) \* log2(p) \* tcalc + (n / q)2 \* (log2(p) - 1) \* tcalc);

In seguito, è possibile calcolarsi l’efficienza dell’algoritmo:

* E(p) = (n2 \* (2 \* n – 1) \* tcalc) / ((n / q)2 \* (2 \* (n / q) – 1) \* log2(p) \* tcalc + (n / q)2 \* (log2(p) - 1) \* tcalc) / p;

Di seguito, è riportata la tabella dei parametri all’aumentare del numero dei processori:

|  |  |  |  |
| --- | --- | --- | --- |
|  | T(p) | S(p) | E(p) |
| Griglia 1x1 | 3312 | 1 | 1 |
| Griglia 2x2 | 828 | 4 | 1 |
| Griglia 3x3 | 368 | 9 | 1 |
| Griglia 4x4 | 207 | 16 | 1 |

## 6.2 Analisi dei tempi di esecuzione con matrici 40x40

Nella seguente tabella e nel seguente grafico sono riportati i tempi registrati incrementando il numero di processori utilizzati, fissata la dimensione del problema a 40⨯40.

|  |  |
| --- | --- |
| **#Processori** | **T(p)** |
| **1** | 0,000768 |
| **4** | 0,000375 |

In questo scenario, cioè con dimensione della matrice pari a 40 righe e 40 colonne, si può osservare dalla tabella e dal grafico del tempo di esecuzione dell’algoritmo del prodotto matrice-matrice che si registrano tempi migliori passando da 1 processore (esecuzione sequenziale) a 4 processori che lavorano in parallelo.

## 6.3 Analisi dei tempi di esecuzione con matrici 80x80

Nella seguente tabella e nel seguente grafico sono riportati i tempi registrati incrementando il numero di processori utilizzati, fissata la dimensione del problema a 80⨯80.

|  |  |
| --- | --- |
| **#Processori** | **T(p)** |
| **1** | 0,006114 |
| **4** | 0,001915 |

Dalla tabella e dal grafico del tempo di esecuzione dell’algoritmo del prodotto matrice-matrice evince che per questo particolare scenario si hanno tempi migliori passando da 1 processore (esecuzione sequenziale) a 4 processori che lavorano in parallelo.

## 6.4 Analisi dei tempi di esecuzione con matrici 100x100

Nella seguente tabella e nel seguente grafico sono riportati i tempi registrati incrementando il numero di processori utilizzati, fissata la dimensione del problema a 100⨯100.

|  |  |
| --- | --- |
| **#Processori** | **T(p)** |
| **1** | 0,011913 |
| **4** | 0,00319 |

Infine, anche in questo scenario, cioè con una matrice di dimensione 100 righe e 100 colonne, analizzando il grafico è possibile confermare che l’andamento del tempo di esecuzione diminuisce utilizzando 4 processori che eseguono la computazione in parallelo, rispetto al caso sequenziale.

## 6.5 Speed-up ed efficienza al variare del numero di processori e dimensione della matrice

### 6.5.1 Speed-up

La seguente tabella riporta i valori di **speed-up** calcolati al variare del numero di processori e della dimensione della matrice.

|  |  |  |  |
| --- | --- | --- | --- |
| **#Processori** | **40x40** | **80x80** | **100x100** |
| **1** | 1 | 1 | 1 |
| **4** | 2,048 | 3,192689 | 3,734483 |

Table 1: SpeedUp al variare del numero di processori e della dimensione della matrice

Di seguito è proposto un grafico che riporta l’andamento dello speed up per gli scenari considerati nel presente studio, ovvero, considerando matrici con le seguenti dimensioni: 40⨯40, 80⨯80 e 100⨯100.

Dal grafico evince che fissata la dimensione della matrice a oppure a e passando da 1 a 4 processori si osserva che lo speed-up si avvicina di meno a quello ideale.

Tuttavia, aumentando la dimensione del problema a è possibile osservare che si registrano speed-up migliori e che quindi tendono maggiormente a quello ideale, rispetto alle dimensioni considerate precedentemente.

### 6.5.2 Efficienza

La seguente tabella riporta i valori di **efficienza** calcolati al variare del numero di processori e della dimensione della matrice.

|  |  |  |  |
| --- | --- | --- | --- |
| **#Processori** | **40x40** | **80x80** | **100x100** |
| **1** | 1 | 1 | 1 |
| **4** | 0,512 | 0,798172 | 0,933621 |

Table 2: Efficienza al variare del numero di processori e della dimensione della matrice

Di seguito è proposto un grafico che riporta l’andamento dell’efficienza per gli scenari considerati nel presente studio, ovvero, considerando matrici con le seguenti dimensioni: 40⨯40, 80⨯80 e 100⨯100.

Dal grafico evince che fissata la dimensione della matrice a oppure a e passando da 1 a 4 processori si osserva che l’efficienza si avvicina di meno a quella ideale.

Tuttavia, aumentando la dimensione del problema a è possibile osservare che si registrano valori per l’efficienza migliori e che quindi tendono maggiormente a quella ideale, rispetto alle dimensioni considerate precedentemente.

# 7. Esempi d’uso

Per lanciare l’esecuzione del calcolo del prodotto tra matrice e matrice basta aggiungere i parametri corrispondenti per il corretto funzionamento. Per esempio con questo file pbs:

#!/bin/bash

#PBS -q studenti

#PBS -l nodes=8:ppn=8

#PBS -N matrix\_x\_matrix\_show\_output

#PBS -o matrix\_x\_matrix\_show\_output.out

#PBS -e matrix\_x\_matrix\_show\_output.err

sort -u $PBS\_NODEFILE > hostlist

NCPU=`wc -l < hostlist`

echo ------------------------------------------------------

echo ' This job is allocated on '${NCPU}' cpu(s)'

echo 'Job is running on node(s): '

cat hostlist

PBS\_O\_WORKDIR=$PBS\_O\_HOME/matrix\_x\_matrix

echo ------------------------------------------------------

echo PBS: qsub is running on $PBS\_O\_HOST

echo PBS: originating queue is $PBS\_O\_QUEUE

echo PBS: executing queue is $PBS\_QUEUE

echo PBS: working directory is $PBS\_O\_WORKDIR

echo PBS: execution mode is $PBS\_ENVIRONMENT

echo PBS: job identifier is $PBS\_JOBID

echo PBS: job name is $PBS\_JOBNAME

echo PBS: node file is $PBS\_NODEFILE

echo PBS: current home directory is $PBS\_O\_HOME

echo PBS: PATH = $PBS\_O\_PATH

echo ------------------------------------------------------

echo "Eseguo/usr/lib64/openmpi/1.4-gcc/bin/mpicc -o $PBS\_O\_WORKDIR/matrix\_x\_matrix\_show\_output $PBS\_O\_WORKDIR/main.c"

/usr/lib64/openmpi/1.4-gcc/bin/mpicc -o $PBS\_O\_WORKDIR/matrix\_x\_matrix\_show\_output $PBS\_O\_WORKDIR/main.c

echo "Executing it with 4 processor"

echo "Eseguo /usr/lib64/openmpi/1.4-gcc/bin/mpiexec -machinefile hostlist -n 4 $PBS\_O\_WORKDIR/matrix\_x\_matrix\_show\_output 2 2 4 0 0"

/usr/lib64/openmpi/1.4-gcc/bin/mpiexec -machinefile hostlist -n 4 $PBS\_O\_WORKDIR/matrix\_x\_matrix\_show\_output 2 2 4 0 0

Si sta indicando che si vuole generare una griglia di processori con 2 righe e 2 colonne, due matrici 4 x 4 e che deve essere stampato a video sia le matrici generate casualmente, sia il risultato prodotto dall’algoritmo. Questo è un esempio di file **.out** prodotto:

------------------------------------------------------

This job is allocated on 8 cpu(s)

Job is running on node(s):

wn273.scope.unina.it

wn274.scope.unina.it

wn275.scope.unina.it

wn276.scope.unina.it

wn277.scope.unina.it

wn278.scope.unina.it

wn279.scope.unina.it

wn280.scope.unina.it

------------------------------------------------------

PBS: qsub is running on ui-studenti.scope.unina.it

PBS: originating queue is studenti

PBS: executing queue is studenti

PBS: working directory is /homes/DMA/PDC/2022/PCCGPP98C/matrix\_x\_matrix

PBS: execution mode is PBS\_BATCH

PBS: job identifier is 4007089.torque02.scope.unina.it

PBS: job name is matrix\_x\_matrix\_show\_output

PBS: node file is /var/spool/pbs/aux//4007089.torque02.scope.unina.it

PBS: current home directory is /homes/DMA/PDC/2022/PCCGPP98C

PBS: PATH = /usr/lib64/openmpi/1.2.7-gcc/bin:/usr/kerberos/bin:/opt/exp\_soft/unina.it/intel/composer\_xe\_2013\_sp1.3.174/bin/intel64:/opt/exp\_soft/unina.it/intel/composer\_xe\_2013\_sp1.3.174/mpirt/bin/intel64:/opt/exp\_soft/unina.it/intel/composer\_xe\_2013\_sp1.3.174/bin/intel64:/opt/exp\_soft/unina.it/intel/composer\_xe\_2013\_sp1.3.174/bin/intel64\_mic:/opt/exp\_soft/unina.it/intel/composer\_xe\_2013\_sp1.3.174/debugger/gui/intel64:/opt/d-cache/srm/bin:/opt/d-cache/dcap/bin:/opt/edg/bin:/opt/glite/bin:/opt/globus/bin:/opt/lcg/bin:/usr/local/bin:/bin:/usr/bin:/opt/exp\_soft/HADOOP/hadoop-1.0.3/bin:/opt/exp\_soft/unina.it/intel/composerxe/bin/intel64/:/opt/exp\_soft/unina.it/MPJExpress/mpj-v0\_38/bin:/homes/DMA/PDC/2022/PCCGPP98C/bin

------------------------------------------------------

Eseguo/usr/lib64/openmpi/1.4-gcc/bin/mpicc -o /homes/DMA/PDC/2022/PCCGPP98C/matrix\_x\_matrix/matrix\_x\_matrix\_show\_output /homes/DMA/PDC/2022/PCCGPP98C/matrix\_x\_matrix/main.c

Executing it with 4 processor

Eseguo /usr/lib64/openmpi/1.4-gcc/bin/mpiexec -machinefile hostlist -n 4 /homes/DMA/PDC/2022/PCCGPP98C/matrix\_x\_matrix/matrix\_x\_matrix\_show\_output 2 2 4 0 0

---------------------------------------------

Matrix A:

Row: 1

48.375690 19.402634 80.371536 98.165825

Row: 2

14.022713 -29.488129 49.957424 -23.367783

Row: 3

-58.891632 -74.670441 40.550392 6.979027

Row: 4

-6.822571 77.565216 -13.497009 -6.394936

---------------------------------------------

Matrix B:

Row: 1

-84.900620 -85.883606 55.407257 37.321716

Row: 2

26.548210 -32.507446 -88.409836 13.448372

Row: 3

-66.707916 32.582321 -13.647392 75.325363

Row: 4

-44.519287 -6.492737 -3.007980 -96.143600

---------------------------------------------

Result:

Row: 1

-13323.710938 -2804.082520 -427.162231 -1317.603027

Row: 2

-4265.632812 1533.712280 2772.501709 6136.508301

Row: 3

1.836060 8761.083984 2764.177979 -818.648499

Row: 4

3823.513672 -2333.743408 -7032.112793 386.660828

---------------------------------------------

Time elapsed: 0.000177

In quest’altro esempio si sta indicando che l’algoritmo viene eseguito due volte, una volta con un solo processore e la seconda volta con quattro processori. Da notare che nel primo caso, la griglia avrà una riga e una colonna, nel secondo due righe e due colonne. Gli ultimi due parametri indicano che si vogliono nascondere le matrici generate e il prodotto tra queste due:

#!/bin/bash

#PBS -q studenti

#PBS -l nodes=8:ppn=8

#PBS -N matrix\_x\_matrix\_40\_x\_40

#PBS -o matrix\_x\_matrix\_40\_x\_40.out

#PBS -e matrix\_x\_matrix\_40\_x\_40.err

sort -u $PBS\_NODEFILE > hostlist

NCPU=`wc -l < hostlist`

echo ------------------------------------------------------

echo ' This job is allocated on '${NCPU}' cpu(s)'

echo 'Job is running on node(s): '

cat hostlist

PBS\_O\_WORKDIR=$PBS\_O\_HOME/matrix\_x\_matrix

echo ------------------------------------------------------

echo PBS: qsub is running on $PBS\_O\_HOST

echo PBS: originating queue is $PBS\_O\_QUEUE

echo PBS: executing queue is $PBS\_QUEUE

echo PBS: working directory is $PBS\_O\_WORKDIR

echo PBS: execution mode is $PBS\_ENVIRONMENT

echo PBS: job identifier is $PBS\_JOBID

echo PBS: job name is $PBS\_JOBNAME

echo PBS: node file is $PBS\_NODEFILE

echo PBS: current home directory is $PBS\_O\_HOME

echo PBS: PATH = $PBS\_O\_PATH

echo ------------------------------------------------------

echo "Eseguo/usr/lib64/openmpi/1.4-gcc/bin/mpicc -o $PBS\_O\_WORKDIR/matrix\_x\_matrix\_40\_x\_40 $PBS\_O\_WORKDIR/main.c"

/usr/lib64/openmpi/1.4-gcc/bin/mpicc -o $PBS\_O\_WORKDIR/matrix\_x\_matrix\_40\_x\_40 $PBS\_O\_WORKDIR/main.c

echo "Executing it with 1 processor"

echo "Eseguo /usr/lib64/openmpi/1.4-gcc/bin/mpiexec -machinefile hostlist -n 1 $PBS\_O\_WORKDIR/matrix\_x\_matrix\_40\_x\_40 1 1 40 1 1"

/usr/lib64/openmpi/1.4-gcc/bin/mpiexec -machinefile hostlist -n 1 $PBS\_O\_WORKDIR/matrix\_x\_matrix\_40\_x\_40 1 1 40 1 1

echo ""

echo "Executing it with 4 processor"

echo "Eseguo /usr/lib64/openmpi/1.4-gcc/bin/mpiexec -machinefile hostlist -n 4 $PBS\_O\_WORKDIR/matrix\_x\_matrix\_40\_x\_40 2 2 40 1 1"

/usr/lib64/openmpi/1.4-gcc/bin/mpiexec -machinefile hostlist -n 4 $PBS\_O\_WORKDIR/matrix\_x\_matrix\_40\_x\_40 2 2 40 1 1

Questo è un esempio di file **.out** generato:

------------------------------------------------------

This job is allocated on 8 cpu(s)

Job is running on node(s):

wn273.scope.unina.it

wn274.scope.unina.it

wn275.scope.unina.it

wn276.scope.unina.it

wn277.scope.unina.it

wn278.scope.unina.it

wn279.scope.unina.it

wn280.scope.unina.it

------------------------------------------------------

PBS: qsub is running on ui-studenti.scope.unina.it

PBS: originating queue is studenti

PBS: executing queue is studenti

PBS: working directory is /homes/DMA/PDC/2022/PCCGPP98C/matrix\_x\_matrix

PBS: execution mode is PBS\_BATCH

PBS: job identifier is 4007088.torque02.scope.unina.it

PBS: job name is matrix\_x\_matrix\_40\_x\_40

PBS: node file is /var/spool/pbs/aux//4007088.torque02.scope.unina.it

PBS: current home directory is /homes/DMA/PDC/2022/PCCGPP98C

PBS: PATH = /usr/lib64/openmpi/1.2.7-gcc/bin:/usr/kerberos/bin:/opt/exp\_soft/unina.it/intel/composer\_xe\_2013\_sp1.3.174/bin/intel64:/opt/exp\_soft/unina.it/intel/composer\_xe\_2013\_sp1.3.174/mpirt/bin/intel64:/opt/exp\_soft/unina.it/intel/composer\_xe\_2013\_sp1.3.174/bin/intel64:/opt/exp\_soft/unina.it/intel/composer\_xe\_2013\_sp1.3.174/bin/intel64\_mic:/opt/exp\_soft/unina.it/intel/composer\_xe\_2013\_sp1.3.174/debugger/gui/intel64:/opt/d-cache/srm/bin:/opt/d-cache/dcap/bin:/opt/edg/bin:/opt/glite/bin:/opt/globus/bin:/opt/lcg/bin:/usr/local/bin:/bin:/usr/bin:/opt/exp\_soft/HADOOP/hadoop-1.0.3/bin:/opt/exp\_soft/unina.it/intel/composerxe/bin/intel64/:/opt/exp\_soft/unina.it/MPJExpress/mpj-v0\_38/bin:/homes/DMA/PDC/2022/PCCGPP98C/bin

------------------------------------------------------

Eseguo/usr/lib64/openmpi/1.4-gcc/bin/mpicc -o /homes/DMA/PDC/2022/PCCGPP98C/matrix\_x\_matrix/matrix\_x\_matrix\_40\_x\_40 /homes/DMA/PDC/2022/PCCGPP98C/matrix\_x\_matrix/main.c

Executing it with 1 processor

Eseguo /usr/lib64/openmpi/1.4-gcc/bin/mpiexec -machinefile hostlist -n 1 /homes/DMA/PDC/2022/PCCGPP98C/matrix\_x\_matrix/matrix\_x\_matrix\_40\_x\_40 1 1 40 1 1

---------------------------------------------

Time elapsed: 0.000768

Executing it with 4 processor

Eseguo /usr/lib64/openmpi/1.4-gcc/bin/mpiexec -machinefile hostlist -n 4 /homes/DMA/PDC/2022/PCCGPP98C/matrix\_x\_matrix/matrix\_x\_matrix\_40\_x\_40 2 2 40 1 1

---------------------------------------------

Time elapsed: 0.000375

Questo esempio è simile al primo caso mostrato, con la differenza che le matrici generate verranno nascoste, siccome il quarto parametro è impostato a 1. Questo è il file **.pbs** usato:

#!/bin/bash

#PBS -q studenti

#PBS -l nodes=8:ppn=8

#PBS -N matrix\_x\_matrix\_hide\_generated\_matrices

#PBS -o matrix\_x\_matrix\_hide\_generated\_matrices.out

#PBS -e matrix\_x\_matrix\_hide\_generated\_matrices.err

sort -u $PBS\_NODEFILE > hostlist

NCPU=`wc -l < hostlist`

echo ------------------------------------------------------

echo ' This job is allocated on '${NCPU}' cpu(s)'

echo 'Job is running on node(s): '

cat hostlist

PBS\_O\_WORKDIR=$PBS\_O\_HOME/matrix\_x\_matrix

echo ------------------------------------------------------

echo PBS: qsub is running on $PBS\_O\_HOST

echo PBS: originating queue is $PBS\_O\_QUEUE

echo PBS: executing queue is $PBS\_QUEUE

echo PBS: working directory is $PBS\_O\_WORKDIR

echo PBS: execution mode is $PBS\_ENVIRONMENT

echo PBS: job identifier is $PBS\_JOBID

echo PBS: job name is $PBS\_JOBNAME

echo PBS: node file is $PBS\_NODEFILE

echo PBS: current home directory is $PBS\_O\_HOME

echo PBS: PATH = $PBS\_O\_PATH

echo ------------------------------------------------------

echo "Eseguo/usr/lib64/openmpi/1.4-gcc/bin/mpicc -o $PBS\_O\_WORKDIR/matrix\_x\_matrix\_hide\_generated\_matrices $PBS\_O\_WORKDIR/main.c"

/usr/lib64/openmpi/1.4-gcc/bin/mpicc -o $PBS\_O\_WORKDIR/matrix\_x\_matrix\_hide\_generated\_matrices $PBS\_O\_WORKDIR/main.c

echo "Executing it with 4 processor"

echo "Eseguo /usr/lib64/openmpi/1.4-gcc/bin/mpiexec -machinefile hostlist -n 4 $PBS\_O\_WORKDIR/matrix\_x\_matrix\_hide\_generated\_matrices 2 2 4 1 0"

/usr/lib64/openmpi/1.4-gcc/bin/mpiexec -machinefile hostlist -n 4 $PBS\_O\_WORKDIR/matrix\_x\_matrix\_hide\_generated\_matrices 2 2 4 1 0

Questo è un esempio di file **.out** generato:

------------------------------------------------------

This job is allocated on 8 cpu(s)

Job is running on node(s):

wn273.scope.unina.it

wn274.scope.unina.it

wn275.scope.unina.it

wn276.scope.unina.it

wn277.scope.unina.it

wn278.scope.unina.it

wn279.scope.unina.it

wn280.scope.unina.it

------------------------------------------------------

PBS: qsub is running on ui-studenti.scope.unina.it

PBS: originating queue is studenti

PBS: executing queue is studenti

PBS: working directory is /homes/DMA/PDC/2022/PCCGPP98C/matrix\_x\_matrix

PBS: execution mode is PBS\_BATCH

PBS: job identifier is 4007091.torque02.scope.unina.it

PBS: job name is matrix\_x\_matrix\_hide\_generated\_matrices

PBS: node file is /var/spool/pbs/aux//4007091.torque02.scope.unina.it

PBS: current home directory is /homes/DMA/PDC/2022/PCCGPP98C

PBS: PATH = /usr/lib64/openmpi/1.2.7-gcc/bin:/usr/kerberos/bin:/opt/exp\_soft/unina.it/intel/composer\_xe\_2013\_sp1.3.174/bin/intel64:/opt/exp\_soft/unina.it/intel/composer\_xe\_2013\_sp1.3.174/mpirt/bin/intel64:/opt/exp\_soft/unina.it/intel/composer\_xe\_2013\_sp1.3.174/bin/intel64:/opt/exp\_soft/unina.it/intel/composer\_xe\_2013\_sp1.3.174/bin/intel64\_mic:/opt/exp\_soft/unina.it/intel/composer\_xe\_2013\_sp1.3.174/debugger/gui/intel64:/opt/d-cache/srm/bin:/opt/d-cache/dcap/bin:/opt/edg/bin:/opt/glite/bin:/opt/globus/bin:/opt/lcg/bin:/usr/local/bin:/bin:/usr/bin:/opt/exp\_soft/HADOOP/hadoop-1.0.3/bin:/opt/exp\_soft/unina.it/intel/composerxe/bin/intel64/:/opt/exp\_soft/unina.it/MPJExpress/mpj-v0\_38/bin:/homes/DMA/PDC/2022/PCCGPP98C/bin

------------------------------------------------------

Eseguo/usr/lib64/openmpi/1.4-gcc/bin/mpicc -o /homes/DMA/PDC/2022/PCCGPP98C/matrix\_x\_matrix/matrix\_x\_matrix\_hide\_generated\_matrices /homes/DMA/PDC/2022/PCCGPP98C/matrix\_x\_matrix/main.c

Executing it with 4 processor

Eseguo /usr/lib64/openmpi/1.4-gcc/bin/mpiexec -machinefile hostlist -n 4 /homes/DMA/PDC/2022/PCCGPP98C/matrix\_x\_matrix/matrix\_x\_matrix\_hide\_generated\_matrices 2 2 4 1 0

---------------------------------------------

Result:

Row: 1

213.258575 696.672485 348.577667 -582.628296

Row: 2

-598.733032 770.455200 1585.496826 1836.915039

Row: 3

8830.206055 -6555.803223 4176.314941 -1063.381348

Row: 4

10181.612305 -11209.970703 9612.801758 730.882324

---------------------------------------------

Time elapsed: 0.000178

In quest’ultimo esempio, simile a quello precedentemente mostrato, si può vedere che verrà nascosta la matrice prodotta da quelle generate casualmente, visto che il quinto parametro è impostato a 1. Questo è il file **.pbs** usato:

#!/bin/bash

#PBS -q studenti

#PBS -l nodes=8:ppn=8

#PBS -N matrix\_x\_matrix\_hide\_result

#PBS -o matrix\_x\_matrix\_hide\_result.out

#PBS -e matrix\_x\_matrix\_hide\_result.err

sort -u $PBS\_NODEFILE > hostlist

NCPU=`wc -l < hostlist`

echo ------------------------------------------------------

echo ' This job is allocated on '${NCPU}' cpu(s)'

echo 'Job is running on node(s): '

cat hostlist

PBS\_O\_WORKDIR=$PBS\_O\_HOME/matrix\_x\_matrix

echo ------------------------------------------------------

echo PBS: qsub is running on $PBS\_O\_HOST

echo PBS: originating queue is $PBS\_O\_QUEUE

echo PBS: executing queue is $PBS\_QUEUE

echo PBS: working directory is $PBS\_O\_WORKDIR

echo PBS: execution mode is $PBS\_ENVIRONMENT

echo PBS: job identifier is $PBS\_JOBID

echo PBS: job name is $PBS\_JOBNAME

echo PBS: node file is $PBS\_NODEFILE

echo PBS: current home directory is $PBS\_O\_HOME

echo PBS: PATH = $PBS\_O\_PATH

echo ------------------------------------------------------

echo "Eseguo/usr/lib64/openmpi/1.4-gcc/bin/mpicc -o $PBS\_O\_WORKDIR/matrix\_x\_matrix\_hide\_result $PBS\_O\_WORKDIR/main.c"

/usr/lib64/openmpi/1.4-gcc/bin/mpicc -o $PBS\_O\_WORKDIR/matrix\_x\_matrix\_hide\_result $PBS\_O\_WORKDIR/main.c

echo "Executing it with 4 processor"

echo "Eseguo /usr/lib64/openmpi/1.4-gcc/bin/mpiexec -machinefile hostlist -n 4 $PBS\_O\_WORKDIR/matrix\_x\_matrix\_hide\_result 2 2 4 0 1"

/usr/lib64/openmpi/1.4-gcc/bin/mpiexec -machinefile hostlist -n 4 $PBS\_O\_WORKDIR/matrix\_x\_matrix\_hide\_result 2 2 4 0 1

Questo è un esempio di file **.out** prodotto:

------------------------------------------------------

This job is allocated on 8 cpu(s)

Job is running on node(s):

wn273.scope.unina.it

wn274.scope.unina.it

wn275.scope.unina.it

wn276.scope.unina.it

wn277.scope.unina.it

wn278.scope.unina.it

wn279.scope.unina.it

wn280.scope.unina.it

------------------------------------------------------

PBS: qsub is running on ui-studenti.scope.unina.it

PBS: originating queue is studenti

PBS: executing queue is studenti

PBS: working directory is /homes/DMA/PDC/2022/PCCGPP98C/matrix\_x\_matrix

PBS: execution mode is PBS\_BATCH

PBS: job identifier is 4007090.torque02.scope.unina.it

PBS: job name is matrix\_x\_matrix\_hide\_result

PBS: node file is /var/spool/pbs/aux//4007090.torque02.scope.unina.it

PBS: current home directory is /homes/DMA/PDC/2022/PCCGPP98C

PBS: PATH = /usr/lib64/openmpi/1.2.7-gcc/bin:/usr/kerberos/bin:/opt/exp\_soft/unina.it/intel/composer\_xe\_2013\_sp1.3.174/bin/intel64:/opt/exp\_soft/unina.it/intel/composer\_xe\_2013\_sp1.3.174/mpirt/bin/intel64:/opt/exp\_soft/unina.it/intel/composer\_xe\_2013\_sp1.3.174/bin/intel64:/opt/exp\_soft/unina.it/intel/composer\_xe\_2013\_sp1.3.174/bin/intel64\_mic:/opt/exp\_soft/unina.it/intel/composer\_xe\_2013\_sp1.3.174/debugger/gui/intel64:/opt/d-cache/srm/bin:/opt/d-cache/dcap/bin:/opt/edg/bin:/opt/glite/bin:/opt/globus/bin:/opt/lcg/bin:/usr/local/bin:/bin:/usr/bin:/opt/exp\_soft/HADOOP/hadoop-1.0.3/bin:/opt/exp\_soft/unina.it/intel/composerxe/bin/intel64/:/opt/exp\_soft/unina.it/MPJExpress/mpj-v0\_38/bin:/homes/DMA/PDC/2022/PCCGPP98C/bin

------------------------------------------------------

Eseguo/usr/lib64/openmpi/1.4-gcc/bin/mpicc -o /homes/DMA/PDC/2022/PCCGPP98C/matrix\_x\_matrix/matrix\_x\_matrix\_hide\_result /homes/DMA/PDC/2022/PCCGPP98C/matrix\_x\_matrix/main.c

Executing it with 4 processor

Eseguo /usr/lib64/openmpi/1.4-gcc/bin/mpiexec -machinefile hostlist -n 4 /homes/DMA/PDC/2022/PCCGPP98C/matrix\_x\_matrix/matrix\_x\_matrix\_hide\_result 2 2 4 0 1

---------------------------------------------

Matrix A:

Row: 1

-13.301025 -29.606201 21.379051 87.400421

Row: 2

63.122704 -10.359703 -50.184769 -72.810455

Row: 3

40.806992 4.305405 -97.157578 74.157013

Row: 4

12.183815 30.251602 -94.433945 -27.762222

---------------------------------------------

Matrix B:

Row: 1

97.612274 -93.538254 -30.692703 3.700653

Row: 2

81.906586 -18.328056 87.003265 -33.821518

Row: 3

21.019997 25.427216 -37.980438 52.847767

Row: 4

16.729515 69.870270 17.822601 -96.571503

---------------------------------------------

Time elapsed: 0.000176