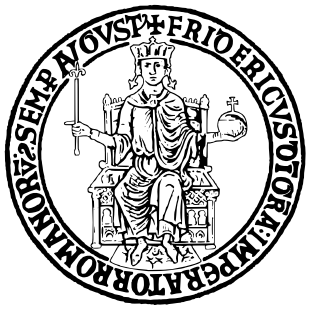
Università degli Studi di Napoli Federico II



Scuola Politecnica e delle Scienze di Base

Dipartimento di Ingegneria Elettrica e Tecnologie dell'Informazione

Corso di Laurea Magistrale in Informatica

Corso di Parallel and Distributed Computing

Prodotto matrice-matrice

Relatore
Prof. Giuliano Laccetti
Prof.ssa Valeria Mele

Candidato
Fabrizio Vitale N97/0449
Giovanni Falcone N97/0451
Luigi Mangiacapra N97/0454

Anno Accademico 2023-2024

Indice

Capitolo 1

Descrizione del problema

Il goal del problema è il calcolo del prodotto matrice-matrice $A \times B = C$ dove $A \in \mathbb{R}^{m \times m}$, $B \in \mathbb{R}^{m \times m}$ e $C \in \mathbb{R}^{m \times m}$, in un ambiente di calcolo parallelo su architettura MIMD a memoria distribuita, utilizzando la libreria MPI sfruttando la strategia BMR.
Le matrici verranno distibuite su una griglia bidimensionale di p processori dove p è un quadrato perfetto.
In particolare, verranno utilizzate matrici di dimensioni differenti, per ef- fettuare tale prodotto e tramite dei grafici verranno mostrate le differenze tra le varie dimensioni della matrice in termini di tempo di esecuzione, speed up ed efficienza.
Il linguaggio scelto è il C.

Capitolo 2

Descrizione algoritmo

In questo capitolo descriveremo la struttura del programma, ovvero come abbiamo organizzato i vari file *.c*, *.h* e *.pbs* e come è strutturato il suddetto programma, indicando i parametri di input/output e gli eventuali errori.

2.1 Struttura del programma

Innanzitutto descriviamo come abbiamo suddiviso i vari file e cosa ciascuno di essi contiene. La struttura del programma è così suddivisa:

```
/
├── Main.c
├── Lib.c
├── Lib.h
└── pbs_exec.pbs
```

dove

- `Main.c` è file principale che contiene il main del programma che richiama le funzioni della libreria `MPI` e `Lib.h`.
- `Lib.c` è il file che contiene le funzioni relative al calcolo del prodotto matrice-vettore e le loro inizializzazioni.
- `Lib.h` è il file che contiene i diversi prototipi.
- `pbs_exec.pbs`: il pbs utilizzato per raccogliere i tempi da analizzare.

2.2 Descrizione del programma

2.2.1 Distribuzione matrici

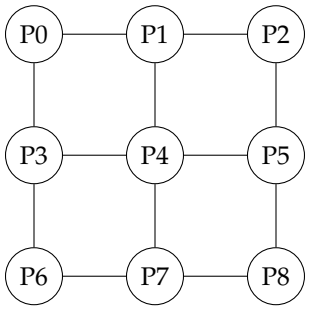
Una volta create le matrici quello che occorre fare è distribuire equamente i blocchi di entrambi le matrici a ciascun processore. Per fare ciò si è deciso di utilizzare `MPI_Scatterv` mediante la quale il processore con rank 0 invia un

singolo blocco a ciascun processore. In particolare verrà inviato un singolo elemento (il blocco, appunto) di un nuovo tipo creato mediante `MPI_Type_Vector`. Di seguito la funzione che si occupa della distribuzione:

```
1  void matrix_distribution(int nproc, double *source, double *dest, int *↩
       displs, int n_loc, int stride){
2      MPI_Datatype vectorType, block_Type;
3
4      MPI_Type_vector(n_loc, n_loc, stride, MPI_DOUBLE, &vectorType);
5      MPI_Type_create_resized(vectorType, 0, sizeof(double), &block_Type)↩
       ;
6      MPI_Type_commit(&block_Type);
7
8      int s[nproc];
9      for (int i = 0; i < nproc; i++)
10         s[i] = 1;
11
12     MPI_Scatterv(source, s, displs, block_Type, dest, n_loc * n_loc, ↩
       MPI_DOUBLE, 0, MPI_COMM_WORLD);
13
14     // free mpi types
15     MPI_Type_free(&vectorType);
16     MPI_Type_free(&block_Type);
17 }
```

2.2.2 Strategia BMR

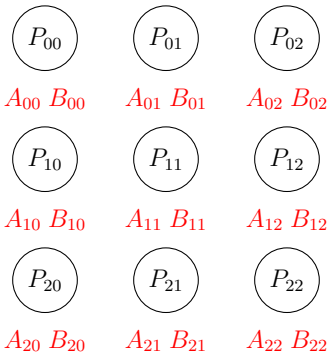
Nello specifico le matrici devono essere distribuite a $p \times p$ processori (per valori m multipli di p), disposti secondo una topologia a griglia bidimensionale di seguito un esempio con una griglia con 9 processori:



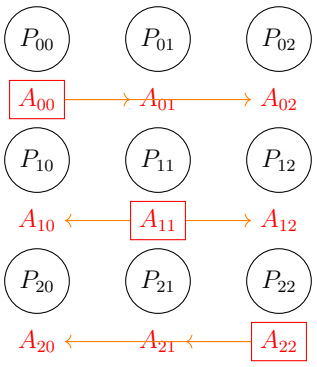
L'algoritmo implementato dovrà soddisfare la strategia di comunicazione BMR (Broadcast Multiply Rolling) e costituito da diverse funzioni per la generazione della griglia bidimensionale e una per calcolare il prodotto di ogni sottoblocco della matrice posseduto da un processore p.

Per effettuare la strategia BMR bisogna innanzitutto distribuire i sottoblocchi della matrice per ogni processore. Per poter effettuare la distribuzione è

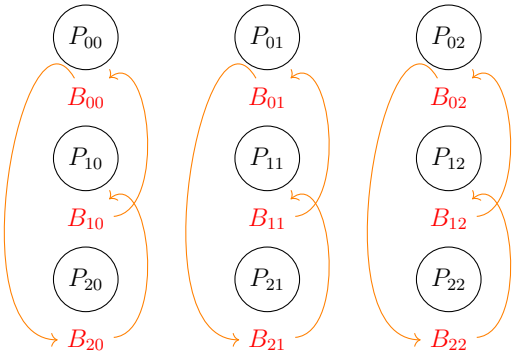
necessario che la dimensione della matrice (m) sia multiplo di p. Una volta effettuata la distribuzione, i dati saranno distribuiti in questo modo:



Con i dati così distribuiti vogliamo che il singolo processo P_{ij} calcoli il singolo sottoblocco C_{ij}, quindi solo i processi sulla diagonale della griglia, cioè P_{ii} possono calcolare il risultato parziale per il sottoblocco C_{ii}. Nel nostro esempio i processi sulla diagonale sono P_{00}, P_{11} e P_{22}.
Questi, durante la fase di *broadcast*, inviano il proprio blocco (A_{00}, A_{11} e A_{22}) ai processori che si trovano sulla loro stessa riga, cioè i processi P_{ij} con $i \neq j$. Questa sarà la situazione durante l'invio del blocco da parte dei processi sulla diagonale:



Dopo aver fatto il *broadcast* ed i prodotti locali per ogni processore, che poteva svolgere questa operazione, viene eseguito il *rolling* della matrice B tramite la griglia di processori organizzati in colonna, lo scambio di dati avviene con il processore precedente situato sulla stessa colonna.



Dopo aver effettuato il *rolling*, le operazioni di broadcast, prodotti locali e rolling vengono ripetuti per le diagonali successive rispetto alla principale, in quest'esempio avremo prima P_{01}, P_{12} e P_{20} e poi successivamente P_{02}, P_{10} e P_{21}. Dopo aver effettuato p passi ogni processore P_{ij} avrà calcolato il corrispondente sottoblocco C_{ij} sommando le componenti ottenute nei passaggi precedenti:

C_{00}	C_{01}	C_{02}
$A_{00}B_{00} + A_{01}B_{10} + A_{02}B_{20}$	$A_{00}B_{01} + A_{01}B_{11} + A_{02}B_{21}$	$A_{00}B_{02} + A_{01}B_{12} + A_{02}B_{22}$
C_{10}	C_{11}	C_{12}
$A_{10}B_{00} + A_{11}B_{20} + A_{12}B_{20}$	$A_{10}B_{01} + A_{11}B_{11} + A_{12}B_{21}$	$A_{10}B_{02} + A_{11}B_{12} + A_{12}B_{22}$
C_{20}	C_{21}	C_{22}
$A_{20}B_{00} + A_{21}B_{10} + A_{22}B_{20}$	$A_{20}B_{01} + A_{21}B_{11} + A_{22}B_{21}$	$A_{20}B_{02} + A_{21}B_{12} + A_{22}B_{22}$

Di seguito troviamo il codice per poter eseguire l'algoritmo BMR. In particolare ad ogni step calcoliamo il processore mittente della diagonale rispetto a quello step che provvederà ad inviare ai processori della sua riga il suo blocco A. Successivamente viene effettuato il prodotto locale e avviene la fase di rolling.

```
1 void BMR(int menum, int n_loc, int grid_dim, double *C_loc, double *↩
      A_loc, double *B_loc, int *coordinate, MPI_Comm *grid, MPI_Comm *↩
      gridr, MPI_Comm *gridc){
2
```

```
3     int step, tag, j, i, sender, menumRow, menumCol, senderCol, ↩
      receiverCol;
4     double **bufferA;
5     MPI_Status status;
6
7     // create a tmp matrix
8     create_matrix(&bufferA, n_loc);
9
10    MPI_Comm_rank(*gridc, &menumCol);
11    MPI_Comm_rank(*gridr, &menumRow);
12
13    for (step = 0; step < grid_dim; step++){
14        if (coordinate[1] == (coordinate[0] + step) % grid_dim){
15            sender = menumRow;
16            copyMatrix(bufferA, A_loc, n_loc);
17        }
18        else
19            sender = (coordinate[0] + step) % grid_dim;
20
21        for (j = 0; j < n_loc; j++)
22            MPI_Bcast(bufferA[j], n_loc, MPI_DOUBLE, sender, *gridr);
23
24        localProduct(bufferA, B_loc, C_loc, n_loc);
25
26        if (menumCol - 1 < 0)
27            receiverCol = (menumCol - 1) + grid_dim;
28        else
29            receiverCol = (menumCol - 1) % grid_dim;
30
31        if (menumCol + 1 < 0)
32            senderCol = (menumCol + 1) + grid_dim;
33        else
34            senderCol = (menumCol + 1) % grid_dim;
35
36        for (j = 0; j < n_loc * n_loc; j++){
37            MPI_Send(&B_loc[j], 1, MPI_DOUBLE, receiverCol,
38            20 + receiverCol, *gridc);
39
40            MPI_Recv(&B_loc[j], 1, MPI_DOUBLE, senderCol,
41            20 + menumCol, *gridc, &status);
42        }
43    }
44 }
```

2.3 Input/output

Il programma prende in input 1 solo parametro, ossia N che indica il numero di righe e di colonne di entrambe le matrici.

In output il programma restituisce la matrice risultante e il tempo impiegato. Tuttavia, nel programma viene stampato solo il risultato in quanto la

stampa della matrice restituita è superflua (soprattutto nei casi in cui i dati sono troppo grandi).

2.4 Errori

Se non vengono rispettate le seguenti linee guida, il programma, semplicemente, terminerà:

- N deve essere necessariamente un intero positivo.
- Il numero dei processori fornito tramite pbs deve essere un quadrato perfetto, altrimenti non sarà possibile creare la griglia quadrata.

Capitolo 3

Descrizione routine

In questo capitolo vengono trattate le funzioni utilizzate per lo scopo del problema: nella sezione 3.1 discuteremo delle funzioni della libreria `MPI` utilizzate, mentre nella sezione 3.2 discuteremo delle funzioni di supporto utilizzate per risolvere il nostro problema.

3.1 Funzioni *MPI* utilizzate

3.1.1 *MPI_Init*

```
int MPI_Init(int *argc, char ***argv)
```

Descrizione: Inizializza l'ambiente di esecuzione MPI.

Parametri di input:

- `argc`: puntatore al numero di parametri
- `argv`: vettore di argomenti

Errors: Restituisce il codice `MPI_SUCCESS` in caso di successo, `MPI_ERR_OTHER` altrimenti.

3.1.2 *MPI_Comm_rank*

```
int MPI_Comm_rank(MPI_Comm comm, int *rank)
```

Descrizione: Determina l'identificativo del processo chiamante nel comunicatore.

Parametri di input:

- `comm`: comunicatore

Parametri di output:

- `rank`: id del processo chiamante nel gruppo `comm`

Errors: Restituisce `MPI_SUCCESS` se la routine termina con successo, `MPI_ERR_COMM` altrimenti (comunicatiore invalido, e.g comunicatore NULL).

3.1.3 *MPI_Comm_size*

```
int MPI_Comm_size(MPI_Comm comm, int *size)
```

Descrizione: Restituisce la dimensione del gruppo associato al comunicatore.

Parametri di input:

- `comm`: comunicatore

Parametri di output:

- `size`: numero di processori nel gruppo `comm`.

Errors: Restituisce `MPI_SUCCESS` se la routine termina con successo, `MPI_ERR_COMM` in caso di comunicatore non valido o `MPI_ERR_ARG` se un argomento non è valido e non è identificato da una classe di errore specificata.

3.1.4 *MPI_Bcast*

```
int MPI_Bcast(void *buffer, int count, MPI_Datatype datatype,
    int root, MPI_Comm comm)
```

Descrizione: Invia un messaggio in broadcast dal processo con id `root` a tutti gli altri processi del gruppo.

Parametri di input:

- `buffer`: puntatore al buffer
- `count`: numero di elementi del buffer
- `datatype`: tipo di dato del buffer
- `root`: id del processo da cui ricevere
- `comm`: communicator

Errors: Restituisce `MPI_SUCCESS` se la routine termina con successo o un codice di errore altrimenti (`MPI_ERR_COMM`, `MPI_ERR_COUNT`, `MPI_ERR_TYPE`, `MPI_ERR_BUFFER`, `MPI_ERR_ROOT`).

3.1.5 *MPI_send*

```
int MPI_Send(const void *buf, int count, MPI_Datatype datatype, int ↩
    dest, int tag, MPI_Comm comm)
```

Descrizione: Invia un messaggio in modalità standard e bloccante.

Parametri di input:

- `buf`: puntatore al buffer
- `count`: numero di elementi del buffer
- `datatype`: tipo di dato del buffer
- `dest`: identificativo del processo destinatario
- `tag`: identificativo del messaggio
- `comm`: comunicatore

Errors: Restituisce `MPI_SUCCESS` se la routine termina con successo o un codice di errore altrimenti (`MPI_ERR_COMM`, `MPI_ERR_COUNT`, `MPI_ERR_TYPE`, `MPI_ERR_BUFFER`, `MPI_ERR_RANK`).

3.1.6 *MPI_Recv*

```
int MPI_Recv(void *buf, int count, MPI_Datatype datatype,
    int source, int tag, MPI_Comm comm, MPI_Status *status)
```

Descrizione: Funzione di ricezione, bloccante: ritorna solo dopo che il buffer di ricezione contiene il nuovo messaggio ricevuto.

Parametri di input:

- `count`: numero di elementi del buffer
- `datatype`: tipo di dato del buffer
- `source`: identificativo del processo mittente
- `tag`: identificativo del messaggio
- `comm`: comunicatore

Parametri di output:

- `buf`: puntatore al buffer di ricezione
- `status`: racchiude informazioni sulla ricezione del messaggio

Errors: Restituisce `MPI_SUCCESS` se la routine termina con successo o un codice di errore altrimenti (`MPI_ERR_COMM`, `MPI_ERR_COUNT`, `MPI_ERR_TYPE`, `MPI_ERR_BUFFER`, `MPI_ERR_RANK`).

3.1.7 *MPI_Wtime*

```
double MPI_Wtime()
```

Descrizione: Restituisce un tempo in secondi.

Output: tempo in secondi (double).

3.1.8 *MPI_Reduce*

```
int MPI_Reduce(const void *sendbuf, void *recvbuf, int count,
               MPI_Datatype datatype, MPI_Op op, int root,
               MPI_Comm comm)
```

Descrizione: esegue un'operazione di riduzione globale (come somma, massimo, AND logico, ecc.) su tutti i membri di un gruppo.

Parametri di input:

- `sendbuf`: puntatore al buffer
- `count`: numero di elementi del buffer
- `datatype`: tipo di dato del buffer
- `op`: operazione di riduzione (`MPI_MAX`, `MPI_MIN`, `MPI_SUM`, ...)
- `root`: identificativo del processo che visualizzerà il risultato
- `comm`: comunicatore

Parametri di output:

- `recvbuff`: puntatore al buffer di ricezione

Errors: Restituisce `MPI_SUCCESS` se la routine termina con successo o un codice di errore altrimenti (`MPI_ERR_COMM`, `MPI_ERR_COUNT`, `MPI_ERR_TYPE`, `MPI_ERR_BUFFER`).

3.1.9 *MPI_Barrier*

```
int MPI_Barrier(MPI_Comm comm)
```

Descrizione: Fornisce un meccanismo sincronizzare per tutti i processi del gruppo: ogni processo si blocca finchè tutti gli altri processi del gruppo non hanno eseguito anch'essi tale routine.

Parametri di input:

- `comm`: communicator

Errors: Restituisce `MPI_SUCCESS` se la routine termina con successo, `MPI_ERR_COMM` altrimenti.

3.1.10 *MPI_Abort*

```
int MPI_Abort(MPI_Comm comm, int errorcode)
```

Descrizione: Interrompe tutti i processi appartenenti al gruppo `comm`.

Parametri di input:

- `comm`: communicator
- `errorcode`: codice di errore da restituire all'ambiente di invocazione

Errors: Restituisce solo `MPI_SUCCESS`

3.1.11 *MPI_Type_Vector*

```
int MPI_Type_vector(int count, int blocklength, int stride,
                    MPI_Datatype oldtype,MPI_Datatype *newtype)
```

Descrizione: Crea un nuovo tipo vettoriale. (Nel nostro caso è usata per spedire blocchi di un nuovo tipo.)

Parametri di input:

- `count`: intero positivo che indica il numero di blocchi
- `blocklength`: intero positivo che indica il numero di elementi nel blocco
- `stride`: intero che indica il numero di elementi che ci sono tra un blocco e l'altro
- `oldType`: vecchio tipo

Parametri di output:

- `newtype`: nuovo tipo del blocco

Errors: Restituisce `MPI_SUCCESS` se termina con successo o con un codice di errore altrimenti (`MPI_ERR_ARG`, `MPI_ERR_OTHER`).

3.1.12 *MPI_Scatterv*

```
int MPI_Scatterv(const void *sendbuf, const int *sendcounts,
                 const int *displs, MPI_Datatype sendtype,
                 void *recvbuf, int recvcount, MPI_Datatype recvtype,
                 int root, MPI_Comm comm)
```

Descrizione: Invia parti del buffer al gruppo `comm`.

Parametri di input:

- `sendbuf`: indirizzo del buffer (significativo solo per il processore che invia)
- `sendcounts`: array di interi che specifica il numero di elementi che ciascun processore deve ricevere
- `displs`: array di interi che specifica l'offset da cui partire per l'invio dei dati
- `sendType`: tipo di dato da inviare
- `recvcount`: numero di elementi che il processore riceverà
- `recvType`: tipo di dato da ricevere
- `root`: rank del processo destinatario
- `comm`: communicator

Parametri di output:

- `recvbuf`: indirizzo del buffer in cui ricevere i dati

Errors: Restituisce `MPI_SUCCESS` se termina con successo o con un codice di errore altrimenti (`MPI_ERR_COMM`, `MPI_ERR_COUNT`, `MPI_ERR_TYPE`, `MPI_ERR_BUFFER`).

3.1.13 *MPI_Finalize*

```
int MPI_Finalize()
```

Descrizione: Termina l'ambiente di esecuzione MPI. Tutti i processi devono chiamare questa routine prima di uscire. Il numero di processi in esecuzione dopo la chiamata di questa routine non è definito.

Errors: Restituisce solo `MPI_SUCCESS`

3.2 Funzioni *ausiliare* utilizzate

3.2.1 La funzione *fill_matrix*

```
void fill_matrix(double *matrix, int N);
```

Descrizione: Riempe la matrice con valori randomici double da 1 a 100.

Parametri di input:

- `matrix`: puntatore alla matrice
- `N`: dimensione matrice

3.2.2 La funzione *print_matrix*

```
void print_matrix(double *matrix, int N);
```

Descrizione: Stampa la matrice su sdt output.

Parametri di input:

- `matrix`: puntatore alla matrice
- `N`: dimensione matrice

3.2.3 La funzione *read_input*

```
void read_input(int argc, char **argv, int *N, int *M, int *n_thread);
```

Descrizione: Verifica se il numero di input fornito è corretto e se questi hanno valori "legali".

Parametri di input:

- `argc`: numero di parametri forniti in input da terminale
- `argv`: vettore di argomenti

Parametri di output:

- `N`: puntatore all'intero che conterrà la dimensione della matrice

Errors: Esce dal programma se Ns è minore di 1.

3.2.4 La funzione *initialize_matrix*

```
void initialize_matrix(double **matrix, int N);
```

Descrizione: Alloca la matrice.

Parametri di input:

- `N`: numero di righe e colonne della matrice

Parametri di output:

- `A`: puntatore all matrice da inizializzare

Errors: Esce dal programma se non è possibile allocare memoria per la matrice.

3.2.5 La funzione *check_if_grid_can_be_created*

```
void check_if_grid_can_be_created(int nproc);
```

Descrizione: Verifica se è possibile creare la griglia.

Parametri di input:

- `nproc`: numero di processori

Errors: Esce dal programma se non è possibile creare la griglia, ovvero se *nproc* non è un quadrato perfetto.

3.2.6 La funzione *get_offset*

```
void get_offset(int *displs, int row_grid, int col_grid, int n_loc,
                int N);
```

Descrizione: Calcola l'offset per ciascun processore al fine di poter capire quale blocco della matrice distribuire.

Parametri di input:

- `row_grid`: numero di righe della griglia
- `col_grid`: numero di colonne della griglia
- `n_loc`: numero di elementi che ciascun processore riceverà
- `N`: dimensione della matrice

Parametri di output:

- `displs`: puntatore all'array di interi

3.2.7 La funzione *matrix_distribution*

```
void matrix_distribution(int nproc, double *source, double *dest,
                         int *displs, int n_loc, int stride);
```

Descrizione: Distribuisce equamente un blocco della matrice a ciascun processore.

Parametri di input:

- `nproc`: numero di processori che dovranno ricevere i blocchi
- `source`: matrice 1D quadrata da distribuire
- `displs`: array di offset interi

- `n_loc`: numero di elementi della riga (risp. colonna) che il processore riceverà (essendo la matrice quadrata farà il quadrato di `n_loc`)
- `stride`: numero di elementi tra un blocco e l'altro

Parametri di output:

- `dest`: matrice 1D quadrata che conterrà il blocco spedito

3.2.8 La funzione *copy_matrix*

```
void copyMatrix(double **dest, double *source, int size)
```

Descrizione: Copia una matrice 1D in una nuova matrice 2D.

Parametri di input:

- `source`: matrice 1D da copiare
- `size`: dimensione della matrice da copiare

Parametri di output:

- `dest`: matrice 2D

3.2.9 La funzione *create_matrix*

```
void create_matrix(double ***matrix, int dim)
```

Descrizione: Inizializza una matrice 2D.

Parametri di input:

- `dim`: dimensione della matrice da inizializzare

Parametri di output:

- `matrix`: matrice 2D

Errors: Esce dal programma se non è possibile allocare memoria per la matrice.

3.2.10 La funzione *localProduct*

```
void localProduct(double **A, double *B, double *res, int size)
```

Descrizione: Effettua il prodotto matrice-matrice e lo salva in res.

Parametri di input:

- `A`: matrice 2D
- `B`: matrice 1D
- `size`: dimensione delle matrici (la dimensione di A e B deve essere la stessa)

Parametri di output:

- `res`: risultato del prodotto: matrice 1D.

3.2.11 La funzione *BMR*

```
void BMR(int menum, int n_loc, int grid_dim, double *C_loc,
        double *A_loc, double *B_loc, int *coordinate,
        MPI_Comm *grid, MPI_Comm *gridr, MPI_Comm *gridc);
```

Descrizione: Applica la strategia BMR per effettuare il prodotto tra le matrici `A_loc` e `B_loc` per ottenere `C_loc`.

Parametri di input:

- `menum`: rank processore
- `n_loc`: numero di elementi di riga/colonna che ciascun processore possiede
- `grid_dim`: dimensione della griglia
- `A_loc`: matrice 1D
- `B_loc`: matrice 1D
- `coordinate`: vettore di coordinate del processore all'interno della griglia
- `grid`: comunicatore per i processi all'interno della griglia
- `gridr`: comunicatore di riga
- `gridc`: comunicatore di colonna

Parametri di output:

- `C_loc`: risultato del prodotto: matrice 1D.

3.2.12 La funzione *create_grid*

```
void create_grid(MPI_Comm *griglia, MPI_Comm *grigliar,
                 MPI_Comm *grigliac, int menum, int nproc, int row,
                 int col, int *coordinate);
```

Descrizione: Crea la griglia e le sotto griglie righe e colonne.

Parametri di input:

- `menum`: rank processore
- `nproc`: numero di processori
- `row`: numero di righe della griglia di processori
- `col`: numero di colonne della griglia di processori

Parametri di output:

- `griglia`: comunicatore per i processi all'interno della griglia
- `grigliar`: comunicatore di riga
- `grigliac`: comunicatore di colonna
- `coordinate`: vettore di coordinate del processore all'interno della griglia

Capitolo 4

Testing

In questo capitolo mostriamo il pbs utilizzato e il relativo output. Semplicemente, nel pbs, iteriamo il numero di volte in cui deve essere eseguito il programma e con quanti processori dovrà essere eseguito. Quindi, per ciascun processore P_i, dove $i \in \{1, 4\}$, eseguiremo il programma dieci volte in modo da poter fare poi una media dei tempi ottenuti in seguito.

N.B.: nel seguente esempio calcoliamo il tempo impiegato direttamente con 4 processori. Per $P1$ è stato eseguito un programma sequenziale esterno.

4.1 File *pbs* utilizzato

```
#!/bin/bash

#PBS -q studenti
#PBS -l nodes=4:ppn=4
#PBS -N Main
#PBS -o Main.out
#PBS -e Main.err

cat $PBS_NODEFILE
echo ----------------------------
sort -u $PBS_NODEFILE > hostlist

NCPU=$(wc -l < hostlist)
echo ----------------------------
echo 'This job is allocated on '${NCPU}' cpu(s)'' on host:'
cat hostlist
echo ----------------------------

PBS_O_WORKDIR=$PBS_O_HOME/ProgettoMatMat

for n in 100 500 800 1000
do
    for i in {1..10}
    do
        echo "************************************************"
        echo "* ITERATION " $i" --- N = " $n "                *"
```

```
27          echo "***********************************************"
28
29          echo -------------------------
30          echo "Eseguo: /usr/lib64/openmpi/1.4-gcc/bin/mpicc -o ↩
    $PBS_O_WORKDIR/Main $PBS_O_WORKDIR/Main.c $PBS_O_WORKDIR/Lib.c"
31          /usr/lib64/openmpi/1.4-gcc/bin/mpicc -o $PBS_O_WORKDIR/Main ↩
    $PBS_O_WORKDIR/Main.c $PBS_O_WORKDIR/Lib.c -lm -std=c99
32
33          echo "Eseguo: /usr/lib64/openmpi/1.4-gcc/bin/-machinefile ↩
    hotlist -np $NCPU $PBS_O_WORKDIR/Main"
34          /usr/lib64/openmpi/1.4-gcc/bin/mpiexec -machinefile hostlist -↩
    np $NCPU $PBS_O_WORKDIR/Main $n
35      done
36  done
```

4.2 Esempio di output del *pbs*

Naturalmente viste le notevoli dimensioni delle matrici di input la stampa del risultato finale (matrice risultante) è omessa. Di seguito, quindi, l'output con i tempi raccolti.

```
1  wn280.scope.unina.it
2  wn280.scope.unina.it
3  wn280.scope.unina.it
4  wn280.scope.unina.it
5  wn279.scope.unina.it
6  wn279.scope.unina.it
7  wn279.scope.unina.it
8  wn279.scope.unina.it
9  wn278.scope.unina.it
10 wn278.scope.unina.it
11 wn278.scope.unina.it
12 wn278.scope.unina.it
13 wn277.scope.unina.it
14 wn277.scope.unina.it
15 wn277.scope.unina.it
16 wn277.scope.unina.it
17 --------------------------
18 --------------------------
19 This job is allocated on 4 cpu(s) on host:
20 wn277.scope.unina.it
21 wn278.scope.unina.it
22 wn279.scope.unina.it
23 wn280.scope.unina.it
24 --------------------------
25 ******************************************************
26 * ITERATION  1  --- N =  100                         *
27 ******************************************************
28 -------------------------
29 Eseguo: /usr/lib64/openmpi/1.4-gcc/bin/mpicc -o /homes/DMA/PDC/2024/↩
    FLCGNN97K/ProgettoMatMat/Main /homes/DMA/PDC/2024/FLCGNN97K/↩
```

```
    ProgettoMatMat/Main.c /homes/DMA/PDC/2024/FLCGNN97K/ProgettoMatMat/↩
    Lib.c
Eseguo: /usr/lib64/openmpi/1.4-gcc/bin/-machinefile hotlist -np 4 /↩
    homes/DMA/PDC/2024/FLCGNN97K/ProgettoMatMat/Main
Il tempo impiegato per svolgere l'algoritmo 0.012014.
*****************************************************
* ITERATION  2  --- N =  100                        *
*****************************************************
-------------------------
Eseguo: /usr/lib64/openmpi/1.4-gcc/bin/mpicc -o /homes/DMA/PDC/2024/↩
    FLCGNN97K/ProgettoMatMat/Main /homes/DMA/PDC/2024/FLCGNN97K/↩
    ProgettoMatMat/Main.c /homes/DMA/PDC/2024/FLCGNN97K/ProgettoMatMat/↩
    Lib.c
Eseguo: /usr/lib64/openmpi/1.4-gcc/bin/-machinefile hotlist -np 4 /↩
    homes/DMA/PDC/2024/FLCGNN97K/ProgettoMatMat/Main
Il tempo impiegato per svolgere l'algoritmo 0.011640.
*****************************************************
* ITERATION  3  --- N =  100                        *
*****************************************************
-------------------------
Eseguo: /usr/lib64/openmpi/1.4-gcc/bin/mpicc -o /homes/DMA/PDC/2024/↩
    FLCGNN97K/ProgettoMatMat/Main /homes/DMA/PDC/2024/FLCGNN97K/↩
    ProgettoMatMat/Main.c /homes/DMA/PDC/2024/FLCGNN97K/ProgettoMatMat/↩
    Lib.c
Eseguo: /usr/lib64/openmpi/1.4-gcc/bin/-machinefile hotlist -np 4 /↩
    homes/DMA/PDC/2024/FLCGNN97K/ProgettoMatMat/Main
Il tempo impiegato per svolgere l'algoritmo 0.011639.
*****************************************************
* ITERATION  4  --- N =  100                        *
*****************************************************
-------------------------
Eseguo: /usr/lib64/openmpi/1.4-gcc/bin/mpicc -o /homes/DMA/PDC/2024/↩
    FLCGNN97K/ProgettoMatMat/Main /homes/DMA/PDC/2024/FLCGNN97K/↩
    ProgettoMatMat/Main.c /homes/DMA/PDC/2024/FLCGNN97K/ProgettoMatMat/↩
    Lib.c
Eseguo: /usr/lib64/openmpi/1.4-gcc/bin/-machinefile hotlist -np 4 /↩
    homes/DMA/PDC/2024/FLCGNN97K/ProgettoMatMat/Main
Il tempo impiegato per svolgere l'algoritmo 0.011646.
*****************************************************
* ITERATION  5  --- N =  100                        *
*****************************************************
-------------------------
Eseguo: /usr/lib64/openmpi/1.4-gcc/bin/mpicc -o /homes/DMA/PDC/2024/↩
    FLCGNN97K/ProgettoMatMat/Main /homes/DMA/PDC/2024/FLCGNN97K/↩
    ProgettoMatMat/Main.c /homes/DMA/PDC/2024/FLCGNN97K/ProgettoMatMat/↩
    Lib.c
Eseguo: /usr/lib64/openmpi/1.4-gcc/bin/-machinefile hotlist -np 4 /↩
    homes/DMA/PDC/2024/FLCGNN97K/ProgettoMatMat/Main
Il tempo impiegato per svolgere l'algoritmo 0.011699.
*****************************************************
* ITERATION  6  --- N =  100                        *
*****************************************************
-------------------------
Eseguo: /usr/lib64/openmpi/1.4-gcc/bin/mpicc -o /homes/DMA/PDC/2024/↩
```

```
      FLCGNN97K/ProgettoMatMat/Main /homes/DMA/PDC/2024/FLCGNN97K/↩
      ProgettoMatMat/Main.c /homes/DMA/PDC/2024/FLCGNN97K/ProgettoMatMat/↩
      Lib.c
65 Eseguo: /usr/lib64/openmpi/1.4-gcc/bin/-machinefile hotlist -np 4 /↩
      homes/DMA/PDC/2024/FLCGNN97K/ProgettoMatMat/Main
66 Il tempo impiegato per svolgere l'algoritmo 0.011632.
67 *****************************************************
68 * ITERATION  7  --- N =  100                        *
69 *****************************************************
70 -------------------------
71 Eseguo: /usr/lib64/openmpi/1.4-gcc/bin/mpicc -o /homes/DMA/PDC/2024/↩
      FLCGNN97K/ProgettoMatMat/Main /homes/DMA/PDC/2024/FLCGNN97K/↩
      ProgettoMatMat/Main.c /homes/DMA/PDC/2024/FLCGNN97K/ProgettoMatMat/↩
      Lib.c
72 Eseguo: /usr/lib64/openmpi/1.4-gcc/bin/-machinefile hotlist -np 4 /↩
      homes/DMA/PDC/2024/FLCGNN97K/ProgettoMatMat/Main
73 Il tempo impiegato per svolgere l'algoritmo 0.011616.
74 *****************************************************
75 * ITERATION  8  --- N =  100                        *
76 *****************************************************
77 -------------------------
78 Eseguo: /usr/lib64/openmpi/1.4-gcc/bin/mpicc -o /homes/DMA/PDC/2024/↩
      FLCGNN97K/ProgettoMatMat/Main /homes/DMA/PDC/2024/FLCGNN97K/↩
      ProgettoMatMat/Main.c /homes/DMA/PDC/2024/FLCGNN97K/ProgettoMatMat/↩
      Lib.c
79 Eseguo: /usr/lib64/openmpi/1.4-gcc/bin/-machinefile hotlist -np 4 /↩
      homes/DMA/PDC/2024/FLCGNN97K/ProgettoMatMat/Main
80 Il tempo impiegato per svolgere l'algoritmo 0.011679.
81 *****************************************************
82 * ITERATION  9  --- N =  100                        *
83 *****************************************************
84 -------------------------
85 Eseguo: /usr/lib64/openmpi/1.4-gcc/bin/mpicc -o /homes/DMA/PDC/2024/↩
      FLCGNN97K/ProgettoMatMat/Main /homes/DMA/PDC/2024/FLCGNN97K/↩
      ProgettoMatMat/Main.c /homes/DMA/PDC/2024/FLCGNN97K/ProgettoMatMat/↩
      Lib.c
86 Eseguo: /usr/lib64/openmpi/1.4-gcc/bin/-machinefile hotlist -np 4 /↩
      homes/DMA/PDC/2024/FLCGNN97K/ProgettoMatMat/Main
87 Il tempo impiegato per svolgere l'algoritmo 0.011641.
88 *****************************************************
89 * ITERATION  10 --- N =  100                        *
90 *****************************************************
91 -------------------------
92 Eseguo: /usr/lib64/openmpi/1.4-gcc/bin/mpicc -o /homes/DMA/PDC/2024/↩
      FLCGNN97K/ProgettoMatMat/Main /homes/DMA/PDC/2024/FLCGNN97K/↩
      ProgettoMatMat/Main.c /homes/DMA/PDC/2024/FLCGNN97K/ProgettoMatMat/↩
      Lib.c
93 Eseguo: /usr/lib64/openmpi/1.4-gcc/bin/-machinefile hotlist -np 4 /↩
      homes/DMA/PDC/2024/FLCGNN97K/ProgettoMatMat/Main
94 Il tempo impiegato per svolgere l'algoritmo 0.011708.
95 *****************************************************
96 * ITERATION  1  --- N =  500                        *
97 *****************************************************
98 -------------------------
```

```
Eseguo: /usr/lib64/openmpi/1.4-gcc/bin/mpicc -o /homes/DMA/PDC/2024/↩
    FLCGNN97K/ProgettoMatMat/Main /homes/DMA/PDC/2024/FLCGNN97K/↩
    ProgettoMatMat/Main.c /homes/DMA/PDC/2024/FLCGNN97K/ProgettoMatMat/↩
    Lib.c
Eseguo: /usr/lib64/openmpi/1.4-gcc/bin/-machinefile hotlist -np 4 /↩
    homes/DMA/PDC/2024/FLCGNN97K/ProgettoMatMat/Main
Il tempo impiegato per svolgere l'algoritmo 0.535740.
*****************************************************
* ITERATION  2  --- N =  500                        *
*****************************************************
-------------------------
Eseguo: /usr/lib64/openmpi/1.4-gcc/bin/mpicc -o /homes/DMA/PDC/2024/↩
    FLCGNN97K/ProgettoMatMat/Main /homes/DMA/PDC/2024/FLCGNN97K/↩
    ProgettoMatMat/Main.c /homes/DMA/PDC/2024/FLCGNN97K/ProgettoMatMat/↩
    Lib.c
Eseguo: /usr/lib64/openmpi/1.4-gcc/bin/-machinefile hotlist -np 4 /↩
    homes/DMA/PDC/2024/FLCGNN97K/ProgettoMatMat/Main
Il tempo impiegato per svolgere l'algoritmo 0.538902.
*****************************************************
* ITERATION  3  --- N =  500                        *
*****************************************************
-------------------------
Eseguo: /usr/lib64/openmpi/1.4-gcc/bin/mpicc -o /homes/DMA/PDC/2024/↩
    FLCGNN97K/ProgettoMatMat/Main /homes/DMA/PDC/2024/FLCGNN97K/↩
    ProgettoMatMat/Main.c /homes/DMA/PDC/2024/FLCGNN97K/ProgettoMatMat/↩
    Lib.c
Eseguo: /usr/lib64/openmpi/1.4-gcc/bin/-machinefile hotlist -np 4 /↩
    homes/DMA/PDC/2024/FLCGNN97K/ProgettoMatMat/Main
Il tempo impiegato per svolgere l'algoritmo 0.532983.
*****************************************************
* ITERATION  4  --- N =  500                        *
*****************************************************
-------------------------
Eseguo: /usr/lib64/openmpi/1.4-gcc/bin/mpicc -o /homes/DMA/PDC/2024/↩
    FLCGNN97K/ProgettoMatMat/Main /homes/DMA/PDC/2024/FLCGNN97K/↩
    ProgettoMatMat/Main.c /homes/DMA/PDC/2024/FLCGNN97K/ProgettoMatMat/↩
    Lib.c
Eseguo: /usr/lib64/openmpi/1.4-gcc/bin/-machinefile hotlist -np 4 /↩
    homes/DMA/PDC/2024/FLCGNN97K/ProgettoMatMat/Main
Il tempo impiegato per svolgere l'algoritmo 0.532853.
*****************************************************
* ITERATION  5  --- N =  500                        *
*****************************************************
-------------------------
Eseguo: /usr/lib64/openmpi/1.4-gcc/bin/mpicc -o /homes/DMA/PDC/2024/↩
    FLCGNN97K/ProgettoMatMat/Main /homes/DMA/PDC/2024/FLCGNN97K/↩
    ProgettoMatMat/Main.c /homes/DMA/PDC/2024/FLCGNN97K/ProgettoMatMat/↩
    Lib.c
Eseguo: /usr/lib64/openmpi/1.4-gcc/bin/-machinefile hotlist -np 4 /↩
    homes/DMA/PDC/2024/FLCGNN97K/ProgettoMatMat/Main
Il tempo impiegato per svolgere l'algoritmo 0.532972.
*****************************************************
* ITERATION  6  --- N =  500                        *
*****************************************************
```

```
133 -------------------------
134 Eseguo: /usr/lib64/openmpi/1.4-gcc/bin/mpicc -o /homes/DMA/PDC/2024/↩
        FLCGNN97K/ProgettoMatMat/Main /homes/DMA/PDC/2024/FLCGNN97K/↩
        ProgettoMatMat/Main.c /homes/DMA/PDC/2024/FLCGNN97K/ProgettoMatMat/↩
        Lib.c
135 Eseguo: /usr/lib64/openmpi/1.4-gcc/bin/-machinefile hotlist -np 4 /↩
        homes/DMA/PDC/2024/FLCGNN97K/ProgettoMatMat/Main
136 Il tempo impiegato per svolgere l'algoritmo 0.532777.
137 ******************************************************
138 * ITERATION  7  --- N =  500                         *
139 ******************************************************
140 -------------------------
141 Eseguo: /usr/lib64/openmpi/1.4-gcc/bin/mpicc -o /homes/DMA/PDC/2024/↩
        FLCGNN97K/ProgettoMatMat/Main /homes/DMA/PDC/2024/FLCGNN97K/↩
        ProgettoMatMat/Main.c /homes/DMA/PDC/2024/FLCGNN97K/ProgettoMatMat/↩
        Lib.c
142 Eseguo: /usr/lib64/openmpi/1.4-gcc/bin/-machinefile hotlist -np 4 /↩
        homes/DMA/PDC/2024/FLCGNN97K/ProgettoMatMat/Main
143 Il tempo impiegato per svolgere l'algoritmo 0.534231.
144 ******************************************************
145 * ITERATION  8  --- N =  500                         *
146 ******************************************************
147 -------------------------
148 Eseguo: /usr/lib64/openmpi/1.4-gcc/bin/mpicc -o /homes/DMA/PDC/2024/↩
        FLCGNN97K/ProgettoMatMat/Main /homes/DMA/PDC/2024/FLCGNN97K/↩
        ProgettoMatMat/Main.c /homes/DMA/PDC/2024/FLCGNN97K/ProgettoMatMat/↩
        Lib.c
149 Eseguo: /usr/lib64/openmpi/1.4-gcc/bin/-machinefile hotlist -np 4 /↩
        homes/DMA/PDC/2024/FLCGNN97K/ProgettoMatMat/Main
150 Il tempo impiegato per svolgere l'algoritmo 0.533562.
151 ******************************************************
152 * ITERATION  9  --- N =  500                         *
153 ******************************************************
154 -------------------------
155 Eseguo: /usr/lib64/openmpi/1.4-gcc/bin/mpicc -o /homes/DMA/PDC/2024/↩
        FLCGNN97K/ProgettoMatMat/Main /homes/DMA/PDC/2024/FLCGNN97K/↩
        ProgettoMatMat/Main.c /homes/DMA/PDC/2024/FLCGNN97K/ProgettoMatMat/↩
        Lib.c
156 Eseguo: /usr/lib64/openmpi/1.4-gcc/bin/-machinefile hotlist -np 4 /↩
        homes/DMA/PDC/2024/FLCGNN97K/ProgettoMatMat/Main
157 Il tempo impiegato per svolgere l'algoritmo 0.534631.
158 ******************************************************
159 * ITERATION  10 --- N =  500                         *
160 ******************************************************
161 -------------------------
162 Eseguo: /usr/lib64/openmpi/1.4-gcc/bin/mpicc -o /homes/DMA/PDC/2024/↩
        FLCGNN97K/ProgettoMatMat/Main /homes/DMA/PDC/2024/FLCGNN97K/↩
        ProgettoMatMat/Main.c /homes/DMA/PDC/2024/FLCGNN97K/ProgettoMatMat/↩
        Lib.c
163 Eseguo: /usr/lib64/openmpi/1.4-gcc/bin/-machinefile hotlist -np 4 /↩
        homes/DMA/PDC/2024/FLCGNN97K/ProgettoMatMat/Main
164 Il tempo impiegato per svolgere l'algoritmo 0.533500.
165 ******************************************************
166 * ITERATION  1  --- N =  800                         *
```

```
167 ******************************************************
168 ------------------------
169 Eseguo: /usr/lib64/openmpi/1.4-gcc/bin/mpicc -o /homes/DMA/PDC/2024/↩
        FLCGNN97K/ProgettoMatMat/Main /homes/DMA/PDC/2024/FLCGNN97K/↩
        ProgettoMatMat/Main.c /homes/DMA/PDC/2024/FLCGNN97K/ProgettoMatMat/↩
        Lib.c
170 Eseguo: /usr/lib64/openmpi/1.4-gcc/bin/-machinefile hotlist -np 4 /↩
        homes/DMA/PDC/2024/FLCGNN97K/ProgettoMatMat/Main
171 Il tempo impiegato per svolgere l'algoritmo 2.456495.
172 ******************************************************
173 * ITERATION  2  --- N =  800                          *
174 ******************************************************
175 ------------------------
176 Eseguo: /usr/lib64/openmpi/1.4-gcc/bin/mpicc -o /homes/DMA/PDC/2024/↩
        FLCGNN97K/ProgettoMatMat/Main /homes/DMA/PDC/2024/FLCGNN97K/↩
        ProgettoMatMat/Main.c /homes/DMA/PDC/2024/FLCGNN97K/ProgettoMatMat/↩
        Lib.c
177 Eseguo: /usr/lib64/openmpi/1.4-gcc/bin/-machinefile hotlist -np 4 /↩
        homes/DMA/PDC/2024/FLCGNN97K/ProgettoMatMat/Main
178 Il tempo impiegato per svolgere l'algoritmo 2.472614.
179 ******************************************************
180 * ITERATION  3  --- N =  800                          *
181 ******************************************************
182 ------------------------
183 Eseguo: /usr/lib64/openmpi/1.4-gcc/bin/mpicc -o /homes/DMA/PDC/2024/↩
        FLCGNN97K/ProgettoMatMat/Main /homes/DMA/PDC/2024/FLCGNN97K/↩
        ProgettoMatMat/Main.c /homes/DMA/PDC/2024/FLCGNN97K/ProgettoMatMat/↩
        Lib.c
184 Eseguo: /usr/lib64/openmpi/1.4-gcc/bin/-machinefile hotlist -np 4 /↩
        homes/DMA/PDC/2024/FLCGNN97K/ProgettoMatMat/Main
185 Il tempo impiegato per svolgere l'algoritmo 2.450653.
186 ******************************************************
187 * ITERATION  4  --- N =  800                          *
188 ******************************************************
189 ------------------------
190 Eseguo: /usr/lib64/openmpi/1.4-gcc/bin/mpicc -o /homes/DMA/PDC/2024/↩
        FLCGNN97K/ProgettoMatMat/Main /homes/DMA/PDC/2024/FLCGNN97K/↩
        ProgettoMatMat/Main.c /homes/DMA/PDC/2024/FLCGNN97K/ProgettoMatMat/↩
        Lib.c
191 Eseguo: /usr/lib64/openmpi/1.4-gcc/bin/-machinefile hotlist -np 4 /↩
        homes/DMA/PDC/2024/FLCGNN97K/ProgettoMatMat/Main
192 Il tempo impiegato per svolgere l'algoritmo 2.476957.
193 ******************************************************
194 * ITERATION  5  --- N =  800                          *
195 ******************************************************
196 ------------------------
197 Eseguo: /usr/lib64/openmpi/1.4-gcc/bin/mpicc -o /homes/DMA/PDC/2024/↩
        FLCGNN97K/ProgettoMatMat/Main /homes/DMA/PDC/2024/FLCGNN97K/↩
        ProgettoMatMat/Main.c /homes/DMA/PDC/2024/FLCGNN97K/ProgettoMatMat/↩
        Lib.c
198 Eseguo: /usr/lib64/openmpi/1.4-gcc/bin/-machinefile hotlist -np 4 /↩
        homes/DMA/PDC/2024/FLCGNN97K/ProgettoMatMat/Main
199 Il tempo impiegato per svolgere l'algoritmo 2.422084.
200 ******************************************************
```

```
* ITERATION  6  --- N =  800                          *
******************************************************
-------------------------
Eseguo: /usr/lib64/openmpi/1.4-gcc/bin/mpicc -o /homes/DMA/PDC/2024/↩
    FLCGNN97K/ProgettoMatMat/Main /homes/DMA/PDC/2024/FLCGNN97K/↩
    ProgettoMatMat/Main.c /homes/DMA/PDC/2024/FLCGNN97K/ProgettoMatMat/↩
    Lib.c
Eseguo: /usr/lib64/openmpi/1.4-gcc/bin/-machinefile hotlist -np 4 /↩
    homes/DMA/PDC/2024/FLCGNN97K/ProgettoMatMat/Main
Il tempo impiegato per svolgere l'algoritmo 2.459333.
******************************************************
* ITERATION  7  --- N =  800                          *
******************************************************
-------------------------
Eseguo: /usr/lib64/openmpi/1.4-gcc/bin/mpicc -o /homes/DMA/PDC/2024/↩
    FLCGNN97K/ProgettoMatMat/Main /homes/DMA/PDC/2024/FLCGNN97K/↩
    ProgettoMatMat/Main.c /homes/DMA/PDC/2024/FLCGNN97K/ProgettoMatMat/↩
    Lib.c
Eseguo: /usr/lib64/openmpi/1.4-gcc/bin/-machinefile hotlist -np 4 /↩
    homes/DMA/PDC/2024/FLCGNN97K/ProgettoMatMat/Main
Il tempo impiegato per svolgere l'algoritmo 2.457622.
******************************************************
* ITERATION  8  --- N =  800                          *
******************************************************
-------------------------
Eseguo: /usr/lib64/openmpi/1.4-gcc/bin/mpicc -o /homes/DMA/PDC/2024/↩
    FLCGNN97K/ProgettoMatMat/Main /homes/DMA/PDC/2024/FLCGNN97K/↩
    ProgettoMatMat/Main.c /homes/DMA/PDC/2024/FLCGNN97K/ProgettoMatMat/↩
    Lib.c
Eseguo: /usr/lib64/openmpi/1.4-gcc/bin/-machinefile hotlist -np 4 /↩
    homes/DMA/PDC/2024/FLCGNN97K/ProgettoMatMat/Main
Il tempo impiegato per svolgere l'algoritmo 2.474622.
******************************************************
* ITERATION  9  --- N =  800                          *
******************************************************
-------------------------
Eseguo: /usr/lib64/openmpi/1.4-gcc/bin/mpicc -o /homes/DMA/PDC/2024/↩
    FLCGNN97K/ProgettoMatMat/Main /homes/DMA/PDC/2024/FLCGNN97K/↩
    ProgettoMatMat/Main.c /homes/DMA/PDC/2024/FLCGNN97K/ProgettoMatMat/↩
    Lib.c
Eseguo: /usr/lib64/openmpi/1.4-gcc/bin/-machinefile hotlist -np 4 /↩
    homes/DMA/PDC/2024/FLCGNN97K/ProgettoMatMat/Main
Il tempo impiegato per svolgere l'algoritmo 2.437597.
******************************************************
* ITERATION  10 --- N =  800                          *
******************************************************
-------------------------
Eseguo: /usr/lib64/openmpi/1.4-gcc/bin/mpicc -o /homes/DMA/PDC/2024/↩
    FLCGNN97K/ProgettoMatMat/Main /homes/DMA/PDC/2024/FLCGNN97K/↩
    ProgettoMatMat/Main.c /homes/DMA/PDC/2024/FLCGNN97K/ProgettoMatMat/↩
    Lib.c
Eseguo: /usr/lib64/openmpi/1.4-gcc/bin/-machinefile hotlist -np 4 /↩
    homes/DMA/PDC/2024/FLCGNN97K/ProgettoMatMat/Main
Il tempo impiegato per svolgere l'algoritmo 2.455658.
```

```
235 *****************************************************
236 * ITERATION  1  --- N =  1000                       *
237 *****************************************************
238 -------------------------
239 Eseguo: /usr/lib64/openmpi/1.4-gcc/bin/mpicc -o /homes/DMA/PDC/2024/↩
        FLCGNN97K/ProgettoMatMat/Main /homes/DMA/PDC/2024/FLCGNN97K/↩
        ProgettoMatMat/Main.c /homes/DMA/PDC/2024/FLCGNN97K/ProgettoMatMat/↩
        Lib.c
240 Eseguo: /usr/lib64/openmpi/1.4-gcc/bin/-machinefile hotlist -np 4 /↩
        homes/DMA/PDC/2024/FLCGNN97K/ProgettoMatMat/Main
241 Il tempo impiegato per svolgere l'algoritmo 4.484400.
242 *****************************************************
243 * ITERATION  2  --- N =  1000                       *
244 *****************************************************
245 -------------------------
246 Eseguo: /usr/lib64/openmpi/1.4-gcc/bin/mpicc -o /homes/DMA/PDC/2024/↩
        FLCGNN97K/ProgettoMatMat/Main /homes/DMA/PDC/2024/FLCGNN97K/↩
        ProgettoMatMat/Main.c /homes/DMA/PDC/2024/FLCGNN97K/ProgettoMatMat/↩
        Lib.c
247 Eseguo: /usr/lib64/openmpi/1.4-gcc/bin/-machinefile hotlist -np 4 /↩
        homes/DMA/PDC/2024/FLCGNN97K/ProgettoMatMat/Main
248 Il tempo impiegato per svolgere l'algoritmo 4.533177.
249 *****************************************************
250 * ITERATION  3  --- N =  1000                       *
251 *****************************************************
252 -------------------------
253 Eseguo: /usr/lib64/openmpi/1.4-gcc/bin/mpicc -o /homes/DMA/PDC/2024/↩
        FLCGNN97K/ProgettoMatMat/Main /homes/DMA/PDC/2024/FLCGNN97K/↩
        ProgettoMatMat/Main.c /homes/DMA/PDC/2024/FLCGNN97K/ProgettoMatMat/↩
        Lib.c
254 Eseguo: /usr/lib64/openmpi/1.4-gcc/bin/-machinefile hotlist -np 4 /↩
        homes/DMA/PDC/2024/FLCGNN97K/ProgettoMatMat/Main
255 Il tempo impiegato per svolgere l'algoritmo 4.500911.
256 *****************************************************
257 * ITERATION  4  --- N =  1000                       *
258 *****************************************************
259 -------------------------
260 Eseguo: /usr/lib64/openmpi/1.4-gcc/bin/mpicc -o /homes/DMA/PDC/2024/↩
        FLCGNN97K/ProgettoMatMat/Main /homes/DMA/PDC/2024/FLCGNN97K/↩
        ProgettoMatMat/Main.c /homes/DMA/PDC/2024/FLCGNN97K/ProgettoMatMat/↩
        Lib.c
261 Eseguo: /usr/lib64/openmpi/1.4-gcc/bin/-machinefile hotlist -np 4 /↩
        homes/DMA/PDC/2024/FLCGNN97K/ProgettoMatMat/Main
262 Il tempo impiegato per svolgere l'algoritmo 4.484229.
263 *****************************************************
264 * ITERATION  5  --- N =  1000                       *
265 *****************************************************
266 -------------------------
267 Eseguo: /usr/lib64/openmpi/1.4-gcc/bin/mpicc -o /homes/DMA/PDC/2024/↩
        FLCGNN97K/ProgettoMatMat/Main /homes/DMA/PDC/2024/FLCGNN97K/↩
        ProgettoMatMat/Main.c /homes/DMA/PDC/2024/FLCGNN97K/ProgettoMatMat/↩
        Lib.c
268 Eseguo: /usr/lib64/openmpi/1.4-gcc/bin/-machinefile hotlist -np 4 /↩
        homes/DMA/PDC/2024/FLCGNN97K/ProgettoMatMat/Main
```

```
Il tempo impiegato per svolgere l'algoritmo 4.512730.
*****************************************************
* ITERATION  6  --- N =  1000                       *
*****************************************************
-------------------------
Eseguo: /usr/lib64/openmpi/1.4-gcc/bin/mpicc -o /homes/DMA/PDC/2024/FLCGNN97K/ProgettoMatMat/Main /homes/DMA/PDC/2024/FLCGNN97K/ProgettoMatMat/Main.c /homes/DMA/PDC/2024/FLCGNN97K/ProgettoMatMat/Lib.c
Eseguo: /usr/lib64/openmpi/1.4-gcc/bin/-machinefile hotlist -np 4 /homes/DMA/PDC/2024/FLCGNN97K/ProgettoMatMat/Main
Il tempo impiegato per svolgere l'algoritmo 4.528573.
*****************************************************
* ITERATION  7  --- N =  1000                       *
*****************************************************
-------------------------
Eseguo: /usr/lib64/openmpi/1.4-gcc/bin/mpicc -o /homes/DMA/PDC/2024/FLCGNN97K/ProgettoMatMat/Main /homes/DMA/PDC/2024/FLCGNN97K/ProgettoMatMat/Main.c /homes/DMA/PDC/2024/FLCGNN97K/ProgettoMatMat/Lib.c
Eseguo: /usr/lib64/openmpi/1.4-gcc/bin/-machinefile hotlist -np 4 /homes/DMA/PDC/2024/FLCGNN97K/ProgettoMatMat/Main
Il tempo impiegato per svolgere l'algoritmo 4.518036.
*****************************************************
* ITERATION  8  --- N =  1000                       *
*****************************************************
-------------------------
Eseguo: /usr/lib64/openmpi/1.4-gcc/bin/mpicc -o /homes/DMA/PDC/2024/FLCGNN97K/ProgettoMatMat/Main /homes/DMA/PDC/2024/FLCGNN97K/ProgettoMatMat/Main.c /homes/DMA/PDC/2024/FLCGNN97K/ProgettoMatMat/Lib.c
Eseguo: /usr/lib64/openmpi/1.4-gcc/bin/-machinefile hotlist -np 4 /homes/DMA/PDC/2024/FLCGNN97K/ProgettoMatMat/Main
Il tempo impiegato per svolgere l'algoritmo 4.505378.
*****************************************************
* ITERATION  9  --- N =  1000                       *
*****************************************************
-------------------------
Eseguo: /usr/lib64/openmpi/1.4-gcc/bin/mpicc -o /homes/DMA/PDC/2024/FLCGNN97K/ProgettoMatMat/Main /homes/DMA/PDC/2024/FLCGNN97K/ProgettoMatMat/Main.c /homes/DMA/PDC/2024/FLCGNN97K/ProgettoMatMat/Lib.c
Eseguo: /usr/lib64/openmpi/1.4-gcc/bin/-machinefile hotlist -np 4 /homes/DMA/PDC/2024/FLCGNN97K/ProgettoMatMat/Main
Il tempo impiegato per svolgere l'algoritmo 4.473488.
*****************************************************
* ITERATION  10 --- N =  1000                       *
*****************************************************
-------------------------
Eseguo: /usr/lib64/openmpi/1.4-gcc/bin/mpicc -o /homes/DMA/PDC/2024/FLCGNN97K/ProgettoMatMat/Main /homes/DMA/PDC/2024/FLCGNN97K/ProgettoMatMat/Main.c /homes/DMA/PDC/2024/FLCGNN97K/ProgettoMatMat/Lib.c
Eseguo: /usr/lib64/openmpi/1.4-gcc/bin/-machinefile hotlist -np 4 /
```

```
    homes/DMA/PDC/2024/FLCGNN97K/ProgettoMatMat/Main
Il tempo impiegato per svolgere l'algoritmo 4.507233.
```

4.3 Esempio di output breve

Di seguito un esempio di output con le matrici di input e il risultato finale stampato su stdout.

```
wn280.scope.unina.it
wn280.scope.unina.it
wn280.scope.unina.it
wn280.scope.unina.it
wn279.scope.unina.it
wn279.scope.unina.it
wn279.scope.unina.it
wn279.scope.unina.it
wn278.scope.unina.it
wn278.scope.unina.it
wn278.scope.unina.it
wn278.scope.unina.it
wn277.scope.unina.it
wn277.scope.unina.it
wn277.scope.unina.it
wn277.scope.unina.it
---------------------------
---------------------------
This job is allocated on 4 cpu(s) on host:
wn277.scope.unina.it
wn278.scope.unina.it
wn279.scope.unina.it
wn280.scope.unina.it
---------------------------
echo "************************************************"
echo "* ITERATION " $i" --- N = " $n "                *"
echo "************************************************"
-------------------------
Eseguo: /usr/lib64/openmpi/1.4-gcc/bin/mpicc -o /homes/DMA/PDC/2024/↩
    FLCGNN97K/ProgettoMatMat/Main /homes/DMA/PDC/2024/FLCGNN97K/↩
    ProgettoMatMat/Main.c /homes/DMA/PDC/2024/FLCGNN97K/ProgettoMatMat/↩
    Lib.c
Eseguo: /usr/lib64/openmpi/1.4-gcc/bin/-machinefile hotlist -np 4 /↩
    homes/DMA/PDC/2024/FLCGNN97K/ProgettoMatMat/Main
-----------------------------------------
---          Matrix           ---
-----------------------------------------
[ 53.59, 44.59, 9.50, 16.17
  54.70, 77.43, 6.69, 84.89
  18.08, 46.63, 86.43, 63.62
  74.95, 94.49, 41.37, 89.80 ]
-----------------------------------------
-----------------------------------------
---          Matrix           ---
```

```
41 ------------------------------------------
42 [ 32.74, 52.74, 92.81, 55.40
43   62.96, 52.95, 64.81, 21.18
44   28.89, 5.66, 21.27, 97.48
45   21.80, 57.91, 56.40, 74.38 ]
46 ------------------------------------------
47
48 result of P0:
49 [ 5188.74, 6177.29
50   8709.42, 11938.26 ]
51
52 result of P1:
53 [ 8977.25, 6041.85
54   15024.68, 11635.78 ]
55
56 result of P2:
57 [ 7411.13, 7595.56
58   11555.55, 14390.40 ]
59
60 result of P3:
61 [ 10126.65, 15146.36
62   19024.77, 16865.18 ]
63
64 Il tempo impiegato per svolgere l'algoritmo 0.000498.
```

Capitolo 5

Analisi performance

In questo capitolo analizzeremo il tempo medio impiegato, lo speed up e l'efficienza al variare del numero di processori per ogni matrice di grandezza diversa, più precisamente:

- La dimensione della matrice è $N \in \{100, 500, 800, 1000\}$
- P_i con $i \in 1, 4$
- Per calcolare il tempo medio, per ciascun caso il programma è stato eseguito esattamente 10 volte per considerare, appunto, la media aritmetica
- Una volta ricavato il tempo medio è stato possibile calcolare lo speed up e l'efficienza mediante le seguenti formule:
  - Speed up $S(P) = \frac{T(1)}{T(P)}$, ricordando che lo speed up ideale è uguale a P
  - Efficienza $E(P) = \frac{S(P)}{P}$, tenendo presente che l'efficienza ideale è uguale ad 1

Tutti i test sono stati effettuati con matrici contenenti valori casuali di tipo $double$ nell'intervallo $[1, 100]$.

5.1 Analisi con $N = 100$

N. processori	P1	P4
	0.01190615	0.012014
	0.01096320	0.011640
	0.01105809	0.011639
	0.01190591	0.011646
	0.01191998	0.011699
	0.01191306	0.011632
	0.01192617	0.011616
	0.01190495	0.011679
	0.01168609	0.011641
	0.01170421	0.011708
Tempo medio	0.0116887	0.01170
Speed up	1	0.9990341
Efficienza	1	0.2497585

Tabella 5.1: Analisi con $N = 100$

5.2 Analisi con $N = 500$

N. processori	P1	P4
	1.467198	0.535740
	1.465302	0.538902
	1.467164	0.532983
	1.465655	0.532853
	1.466269	0.532972
	1.472749	0.532777
	1.462238	0.534231
	1.469395	0.533562
	1.470305	0.534631
	1.466950	0.533500
Tempo medio	1.4673225	0.5342
Speed up	1	2.74676619
Efficienza	1	0.6866915

Tabella 5.2: Analisi con $N = 500$

5.3 Analisi con $N = 800$

N. processori	P1	P4
	6.895639	2.456495
	6.864784	2.472614
	6.886846	2.450653
	6.865991	2.476957
	6.867698	2.422084
	6.867506	2.459333
	6.864762	2.457622
	6.868934	2.474622
	6.867530	2.437597
	6.862895	2.455658
Tempo medio	6.8712585	2.4564
Speed up	1	2.797288
Efficienza	1	0.699322

Tabella 5.3: Analisi con $N = 800$

5.4 Analisi con $N = 1000$

N. processori	P1	P4
	14.05452	4.484400
	13.77244	4.533177
	14.16935	4.500911
	13.77154	4.484229
	14.02327	4.512730
	13.77362	4.528573
	14.21127	4.518036
	13.76472	4.505378
	14.12304	4.473488
	13.74386	4.507233
Tempo medio	13.940763	4.5048
Speed up	1	3.0946463
Efficienza	1	0.773661575

Tabella 5.4: Analisi con $N = 1000$

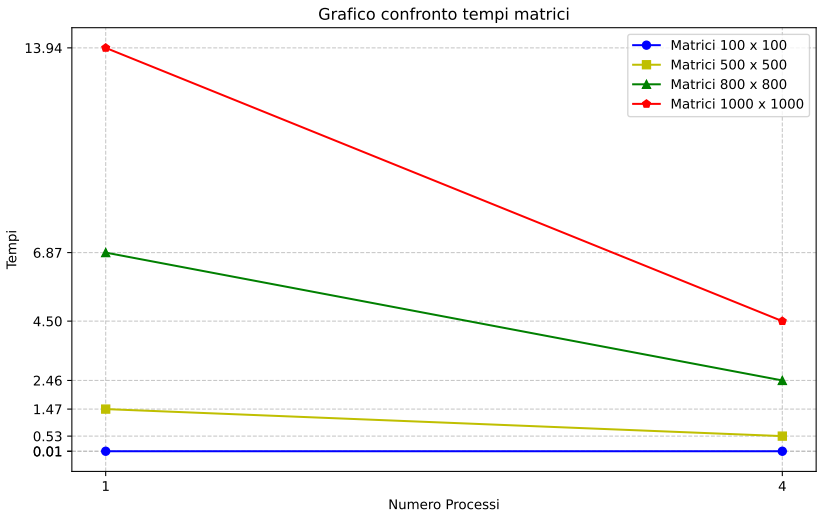


Figura 5.1: Grafico confronto dei tempi con 1 e 4 processori

Nel grafico 5.1 si evince, innanzitutto, quanto i tempi del prodotto matrice matrice siano differenti se calcolati in modo sequenziale o con la strategia BMR. Nello specifico si nota come con l'aumentare della grandezza del problema, e quindi delle matrici utilizzate per il prodotto, il distacco dei tempi nei due approcci diventa sempre più grande. Difatti con matrici piccole (100x100) il distacco si nota poco, rendendo inutile la strategia BMR e preferendo quella sequenziale poiché più semplice da implementare e perché priva di tempi di comunicazione che potrebbero solo procurare inutili ritardi.

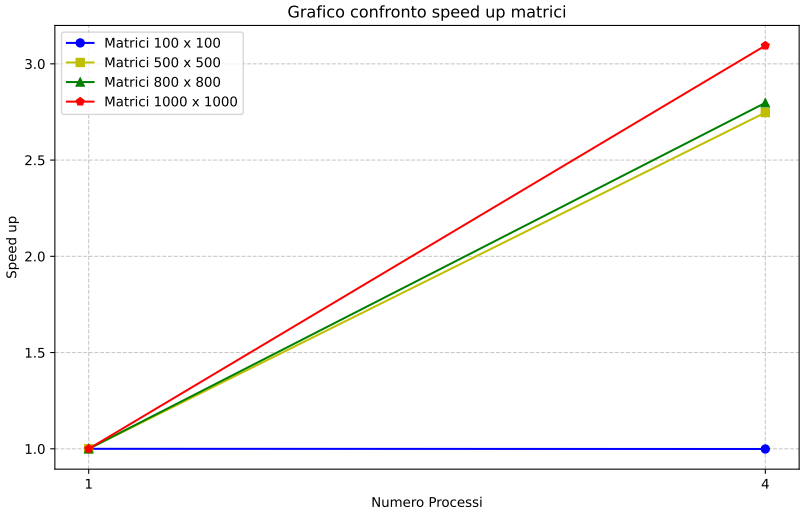


Figura 5.2: Grafico confronto dello speed up con 1 e 4 processori

Dal grafico 5.2, invece, si comprende quanto l'applicazione del BMR nel caso di matrici 100x100 è inutile in quanto lo speed up per $p = 4$ sia davvero distante da quello ideale. Inoltre si nota come all'aumentare della grandezza del problema per $p = 4$ lo speed up si avvicini sempre di più a quello ideale rendendo a tutti gli effetti la strategia BMR più performante. Da ciò si intuisce quanto la strategia BMR sia più utile con matrici molto grandi.

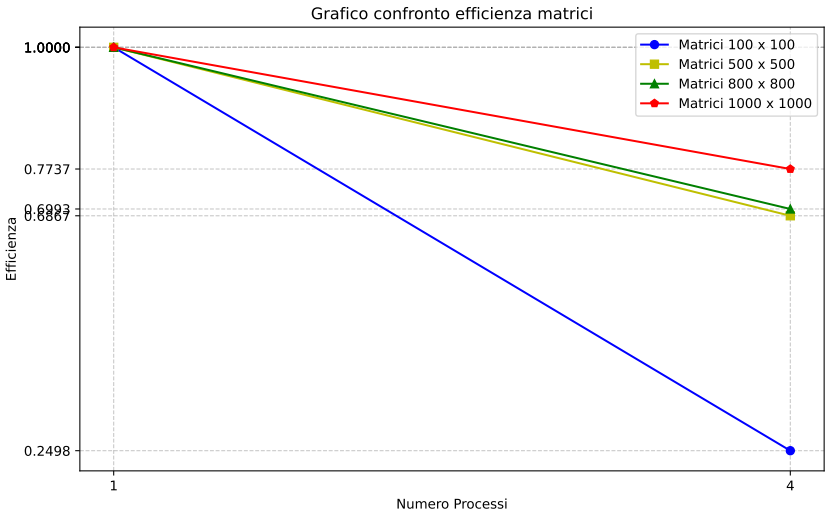


Figura 5.3: Grafico confronto dell'efficienza con 1 e 4 processori

Nel grafico 5.3 si intuisce quanto degrada l'efficienza con l'utilizzo del BMR rispetto al caso sequenziale. Nello specifico il degrado avviene più rapidamente nel caso di dimensioni piccole del problema (e.g. matrici 100x100) e quanto all'aumentare della grandezza del problema l'efficienza si avvicini a quella ideale.

Capitolo 6

Source code

6.1 *Main.c*

```
1     /**
2  * @author Fabrizio Vitale
3  * @author Giovanni Falcone
4  * @author Luigi Mangiacapra
5  */
6
7 #include <stdio.h>
8 #include <stdlib.h>
9 #include <time.h>
10 #include <string.h>
11 #include <math.h>
12 #include <unistd.h>
13
14 #include "mpi.h"
15 #include "Lib.h"
16
17 int main(int argc, char **argv)
18 {
19     int menum;                        // id processo
20     int nproc;                        // numero di processi
21     int row_grid, col_grid, dimGrid; // numero di righe e di colonne ↩
       della griglia di processori
22     int N;                            // numero di righe e colonne della↩
        matrice di valori numerici
23     int *coordinate;                  // coordinate griglia
24     double *A_loc, *B_loc, *C_loc;    // matrice di elementi locale
25     double *A, *B;                    // matrice di elementi fornita in ↩
       input
26     double startTime, stopTime;
27     MPI_Comm comm_grid;                    // griglia
28     MPI_Comm comm_grid_row, comm_grid_col; // sotto griglie
29
30     MPI_Init(&argc, &argv);
31     MPI_Comm_rank(MPI_COMM_WORLD, &menum);
32     MPI_Comm_size(MPI_COMM_WORLD, &nproc);
33
```

```
34      srand(time(NULL));
35      // get input data and initialize the matrix
36      if (menum == 0){
37          read_input(argc, argv, &N);
38
39          // get dim grid
40          dimGrid = sqrt(nproc);
41          // exit if matrix cannot be divided equally
42          const int rest = N % dimGrid;
43          const int is_there_a_rest = rest != 0;
44          if (is_there_a_rest){
45              fprintf(stderr, "Matrix cannot be divided equally!\n");
46              MPI_Abort(MPI_COMM_WORLD, EXIT_FAILURE);
47              exit(EXIT_FAILURE);
48          }
49
50          // if the grid cannot be created exit from program
51          check_if_grid_can_be_created(nproc);
52
53          // create and fill A matrix
54          initialize_matrix(&A, N);
55          fill_matrix(A, N);
56          // create and fill B matrix
57          initialize_matrix(&B, N);
58          fill_matrix(B, N);
59      }
60
61      // send data from process with rand 0 to all process
62      MPI_Bcast(&N, 1, MPI_INT, 0, MPI_COMM_WORLD);
63      MPI_Bcast(&dimGrid, 1, MPI_INT, 0, MPI_COMM_WORLD);
64
65      // compute column number of grid
66      col_grid = row_grid = dimGrid;
67
68      // else create grid
69      const int dim = 2;
70      coordinate = (int *)calloc(dim, sizeof(int));
71      create_grid(&comm_grid, &comm_grid_row, &comm_grid_col, menum, ↩
        nproc, row_grid, col_grid, coordinate);
72
73      const int n_loc = N / row_grid; // number of rows of each process
74      const int stride = N;           // movement between one cell and ↩
        another
75
76      // defination of displs for MPI_Scatterv
77      int *displs = malloc(sizeof(int) * row_grid * col_grid);
78      // compute offset in order to understand where to start from matrix↩
         to send a block
79      get_offset(displs, row_grid, col_grid, n_loc, N);
80
81      // Allocazione della matrice locale su ogni processo
82      A_loc = (double *)calloc(n_loc * n_loc, sizeof(double));
83      B_loc = (double *)calloc(n_loc * n_loc, sizeof(double));
84      C_loc = (double *)calloc(n_loc * n_loc, sizeof(double));
```

```
85      if (A_loc == NULL || B_loc == NULL || C_loc == NULL)
86      {
87          fprintf(stderr, "Error allocating memory for the local matrix!\n");
88          MPI_Finalize();
89          return EXIT_FAILURE;
90      }
91
92      matrix_distribution(nproc, A, A_loc, displs, n_loc, stride);
93      matrix_distribution(nproc, B, B_loc, displs, n_loc, stride);
94
95      // apply BMR strategy and get time
96      MPI_Barrier(comm_grid);
97      startTime = MPI_Wtime();
98      BMR(menum, n_loc, dimGrid, C_loc, A_loc, B_loc, coordinate, &comm_grid, &comm_grid_row, &comm_grid_col);
99      stopTime = MPI_Wtime();
100
101     MPI_Barrier(comm_grid);
102
103     // compute total time
104     double timetot;
105     double timeP = stopTime - startTime;
106     MPI_Reduce(&timeP, &timetot, 1, MPI_DOUBLE, MPI_MAX, 0, MPI_COMM_WORLD);
107     // print total time
108     if(menum == 0) printf("\nTempo impiegato per svolgere l'algoritmo: %lf seconds\n", timetot);
109
110     // debug
111     // in order to print in sequential way each result
112     // MPI_Barrier(comm_grid);
113     // for (int i = 0; i < nproc; i++)
114     // {
115     //     MPI_Barrier(comm_grid);
116     //     if (menum == i)
117     //     {
118     //         if (menum == 0)
119     //             printf("\n");
120     //         printf("result of P%d:\n", menum);
121     //         print_matrix(C_loc, n_loc);
122     //     }
123     // }
124
125     // deallocate memory
126     if (menum == 0)
127     {
128         free(A);
129         free(B);
130     }
131
132     free(displs);
133     free(A_loc);
134     free(B_loc);
```

```
135     free(C_loc);
136     free(coordinate);
137 
138     MPI_Finalize();
139 
140     return 0;
141 }
```

6.2 *Lib.c*

```
1  /**
2   * @author Fabrizio Vitale
3   * @author Giovanni Falcone
4   * @author Luigi Mangiacapra
5   */
6 
7  #include <stdio.h>
8  #include <stdlib.h>
9  #include <time.h>
10 #include <math.h>
11 #include "mpi.h"
12 
13 void read_input(int argc, char **argv, int *N){
14     if (argc != 2){
15         fprintf(stderr, "Usage: <number of row and column>\n");
16         fprintf(stderr, "Exit from program...");
17         MPI_Abort(MPI_COMM_WORLD, EXIT_FAILURE);
18         exit(EXIT_FAILURE);
19     }
20 
21     *N = atoi(argv[1]);
22 
23     if (*N < 1){
24         fprintf(stderr, "Usage: <N> must equal or greater than 1\n");
25         fprintf(stderr, "Exit from program...");
26         MPI_Abort(MPI_COMM_WORLD, EXIT_FAILURE);
27         exit(EXIT_FAILURE);
28     }
29 }
30 
31 void check_if_grid_can_be_created(int nproc){
32     double root = sqrt(nproc);
33 
34     if (root != (int)root){
35         fprintf(stdout, "Grid cannot be created!\n");
36         MPI_Abort(MPI_COMM_WORLD, EXIT_FAILURE);
37         exit(EXIT_FAILURE);
38     }
39 }
40 
41 void initialize_matrix(double **matrix, int N)
42 {
```

```
43      *matrix = (double *)calloc(N * N, sizeof(double));
44
45      if (*matrix == NULL){
46          fprintf(stderr, "Error allocating memory for the 'matrix'
47          array!\n");
48          MPI_Abort(MPI_COMM_WORLD, EXIT_FAILURE);
49          exit(EXIT_FAILURE);
50      }
51  }
52
53  void print_matrix(double *matrix, int N){
54      printf("[");
55      for (int i = 0; i < N; i++){
56          (i == 0) ? printf(" ") : printf("  ");
57          for (int j = 0; j < N; j++){
58              // computes the index of the array
59              int index = i * N + j;
60
61              printf("%.2lf", matrix[index]);
62
63              // adds the comma if it's not the last element of the row
64              if (j < N - 1) printf(", ");
65          }
66
67          // adds the semicolon if it's not the last row
68          if (i < N - 1) printf("\n");
69      }
70      printf(" ]\n");
71  }
72
73  void fill_matrix(double *matrix, int N)
74  {
75      int index = 0;
76
77      for (int i = 0; i < N; i++){
78          for (int j = 0; j < N; j++)
79              matrix[index++] = ((double) rand() / RAND_MAX) * (100 - 1) ↩
    + 1;
80      }
81
82      // printf("-----------------------------------------\n--- \t\t ↩
    Matrix \t\t  --- \n"
83      //         "-----------------------------------------\n");
84      // print_matrix(matrix, N);
85      // printf("-----------------------------------------\n");
86  }
87
88  void print_array(double *array, int size){
89      printf("displs: [ ");
90      for (int i = 0; i < size; i++)
91          printf("%2f ", array[i]);
92      printf("]\n\n");
93  }
94
```

```
95 void create_grid(MPI_Comm *griglia, MPI_Comm *grigliar, MPI_Comm *↩
       grigliac, int menum, int nproc, int riga, int col, int *coordinate){
96     int dim = 2, ndim[2], reorder, *period, vc[2];
97 
98     ndim[0] = riga;
99     ndim[1] = col;
100    period = (int *)calloc(dim, sizeof(int));
101    period[0] = period[1] = 0;
102    reorder = 0;
103 
104    // grid definition
105    MPI_Cart_create(MPI_COMM_WORLD, dim, ndim, period, reorder, griglia↩
       );
106 
107    // each process calculates its own coordinates
108    MPI_Cart_coords(*griglia, menum, 2, coordinate);
109 
110    // division into rows of the communicator
111    vc[0] = 0;
112    vc[1] = 1;
113    MPI_Cart_sub(*griglia, vc, grigliar);
114 
115    // division into columns of the communicator
116    vc[0] = 1;
117    vc[1] = 0;
118    MPI_Cart_sub(*griglia, vc, grigliac);
119 }
120 
121 void get_offset(int *displs, int row_grid, int col_grid, int n_loc, int↩
        N){
122    int offset = 0;
123 
124    for (int i = 0; i < row_grid; i++){
125        for (int j = 0; j < col_grid; j++)
126            displs[i * col_grid + j] = i * N * n_loc + j * n_loc;
127    }
128 }
129 
130 void matrix_distribution(int nproc, double *source, double *dest, int *↩
       displs, int n_loc, int stride){
131    MPI_Datatype vectorType, block_Type;
132 
133    MPI_Type_vector(n_loc, n_loc, stride, MPI_DOUBLE, &vectorType);
134    MPI_Type_create_resized(vectorType, 0, sizeof(double), &block_Type)↩
       ;
135    MPI_Type_commit(&block_Type);
136 
137    int s[nproc];
138    for (int i = 0; i < nproc; i++)
139        s[i] = 1;
140 
141    MPI_Scatterv(source, s, displs, block_Type, dest, n_loc * n_loc, ↩
       MPI_DOUBLE, 0, MPI_COMM_WORLD);
142 
```

```
143     // free mpi types
144     MPI_Type_free(&vectorType);
145     MPI_Type_free(&block_Type);
146 }
147 
148 void copyMatrix(double **dest, double *source, int size){
149     for (int i = 0; i < size; i++){
150         for (int j = 0; j < size; j++)
151             dest[i][j] = source[i * size + j];
152     }
153 }
154 
155 void localProduct(double **A, double *B, double *res, int size){
156     for (int i = 0; i < size; i++){
157         for (int j = 0; j < size; j++){
158             for (int k = 0; k < size; k++){
159                 res[i * size + j] += A[i][k] * B[k * size + j];
160             }
161         }
162     }
163 }
164 
165 void create_matrix(double ***matrix, int dim){
166     *matrix = (double **)calloc(dim, sizeof(double *));
167     for (int i = 0; i < dim; i++)
168         (*matrix)[i] = (double *)calloc(dim, sizeof(double));
169 }
170 
171 void BMR(int menum, int n_loc, int grid_dim, double *C_loc, double *↩
    A_loc, double *B_loc, int *coordinate, MPI_Comm *grid, MPI_Comm *↩
    gridr, MPI_Comm *gridc){
172 
173     int step, tag, j, i, sender, menumRow, menumCol, senderCol, ↩
    receiverCol;
174     double **bufferA;
175     MPI_Status status;
176 
177     // create a tmp matrix
178     create_matrix(&bufferA, n_loc);
179 
180     MPI_Comm_rank(*gridc, &menumCol);
181     MPI_Comm_rank(*gridr, &menumRow);
182 
183     for (step = 0; step < grid_dim; step++){
184         if (coordinate[1] == (coordinate[0] + step) % grid_dim){
185             sender = menumRow;
186             copyMatrix(bufferA, A_loc, n_loc);
187         }
188         else
189             sender = (coordinate[0] + step) % grid_dim;
190 
191         for (j = 0; j < n_loc; j++)
192             MPI_Bcast(bufferA[j], n_loc, MPI_DOUBLE, sender, *gridr);
193 
```

```
194         localProduct(bufferA, B_loc, C_loc, n_loc);
195 
196         if (menumCol - 1 < 0)
197             receiverCol = (menumCol - 1) + grid_dim;
198         else
199             receiverCol = (menumCol - 1) % grid_dim;
200 
201         if (menumCol + 1 < 0)
202             senderCol = (menumCol + 1) + grid_dim;
203         else
204             senderCol = (menumCol + 1) % grid_dim;
205 
206         for (j = 0; j < n_loc * n_loc; j++){
207             MPI_Send(&B_loc[j], 1, MPI_DOUBLE, receiverCol,
208             20 + receiverCol, *gridc);
209 
210             MPI_Recv(&B_loc[j], 1, MPI_DOUBLE, senderCol,
211             20 + menumCol, *gridc, &status);
212         }
213     }
214 }
```

6.3 *Lib.h*

```
1  #ifndef LIB_H
2  #define LIB_H
3 
4  #include "mpi.h"
5 
6  /**
7   * @brief Reads input arguments and exits from program if they are ↩
       incorrect.
8   *
9   * @param argc Number of command-line arguments.
10  * @param argv Array of command-line argument strings.
11  * @param N Pointer to the number of matrix size variable.
12  */
13 void read_input(int argc, char **argv, int *N);
14 
15 /**
16  * @brief Initializes a 1D matrix with calloc.
17  *
18  * @param matrix Pointer to the matrix.
19  * @param N Matrix size.
20  */
21 void initialize_matrix(double **matrix, int N);
22 
23 /**
24  * @brief Fills a 1D matrix with random values.
25  *
26  * @param matrix Pointer to the matrix.
27  * @param N matrix size.
```

```
28  */
29 void fill_matrix(double *matrix, int N);
30
31 /**
32  * @brief Prints the contents of a 1D matrix.
33  *
34  * @param matrix Pointer to the matrix.
35  * @param N Matrix size.
36  */
37 void print_matrix(double *matrix, int N);
38
39 /**
40  * @brief Prints the contents of an array.
41  *
42  * @param array Pointer to the array.
43  * @param size Size of the array.
44  */
45 void print_array(double *array, int size);
46
47 /**
48  * @brief Checks if a grid can be created i.e if nproc is not a perfect square then the grid cannot be created.
49  *
50  * @param nproc Number of MPI processes.
51  */
52 void check_if_grid_can_be_created(int nproc);
53
54 /**
55  * @brief Create a grid object.
56  *
57  * @param griglia grid communicator.
58  * @param grigliar row grid communicator.
59  * @param grigliac column grid communicator.
60  * @param menum Process ID.
61  * @param nproc Number of MPI processes.
62  * @param row Grid row number.
63  * @param col Grid column number.
64  * @param coordinate The grid coordinates of a process
65  */
66 void create_grid(MPI_Comm *griglia, MPI_Comm *grigliar, MPI_Comm *grigliac, int menum, int nproc, int row, int col, int *coordinate);
67
68 /**
69  * @brief Get the offset in order to split the matrix in subblocks.
70  *
71  * @param displs The array which will contain the offset for each process
72  * @param row_grid Grid row number.
73  * @param col_grid Grid column number.
74  * @param n_loc Numer of elements for a row (resp. column)
75  * @param N Matrix size.
76  */
77 void get_offset(int *displs, int row_grid, int col_grid, int n_loc, int N);
```

```
78
79 /**
80  * @brief Divides the matrix into equal subblocks. Each process will ↩
       have one subblock.
81  *
82  * @param nproc number of process.
83  * @param source The 1D matrix to split.
84  * @param dest The 1D matrix that each process will receive.
85  * @param displs The array of offset.
86  * @param n_loc Numer of elements for a row (resp. column)
87  * @param stride Movement between one cell and another.
88  */
89 void matrix_distribution(int nproc, double *source, double *dest, int *↩
       displs, int n_loc, int stride);
90
91 /**
92  * @brief Apply BMR strategy between A_loc and B_loc in order to get ↩
       C_loc.
93  *
94  * @param menum Process ID.
95  * @param n_loc Numer of elements for a row (resp. column).
96  * @param grid_dim Grid size.
97  * @param C_loc Dest matrix which will contain partial result for each ↩
       process.
98  * @param A_loc Subblock matrix
99  * @param B_loc Subblock matrix
100 * @param coordinate The grid coordinates of a process
101 * @param grid Grid communicator
102 * @param gridr Grid row communicator
103 * @param gridc Grid column communicator
104 */
105 void BMR(int menum, int n_loc, int grid_dim, double *C_loc, double *↩
       A_loc, double *B_loc, int *coordinate, MPI_Comm *grid, MPI_Comm *↩
       gridr, MPI_Comm *gridc);
106
107 /**
108 * @brief Copy the source matrix to dest.
109 *
110 * @param dest Destination matrix
111 * @param source Source matrix
112 * @param size Size of source matrix
113 */
114 void copyMatrix(double **dest, double *source, int size);
115
116 /**
117 * @brief Apply the product between matrix A and B and save it in res.
118 *
119 * @param A matrix 2D.
120 * @param B matrix 1D.
121 * @param res result matrix
122 * @param size Numer of elements for a row (resp. column).
123 */
124 void localProduct(double **A, double *B, double *res, int size);
125
```

```c
/**
 * @brief Create a 2D matrix object
 *
 * @param matrix Pointer to 2D matrix
 * @param dim Size matrix
 */
void create_matrix(double ***matrix, int dim);

#endif
```

6.4 *Sequential.c*

Di seguito il codice per permettere l'esecuzione del prodotto matrice-matrice con 1 solo processore.

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include "mpi.h"

int main(int argc, char *argv[])
{
    int **matA,
        **matB,
        **matRes;
    int n, i, j, k;
    double t0, t1 = 0, time;

    MPI_Init(&argc, &argv);

    n = atoi(argv[1]);

    // alloco matrice a
    matA = (int **)calloc(n, sizeof(int *));
    for (i = 0; i < n; i++)
    {
        matA[i] = (int *)calloc(n, sizeof(int));
    }
    // riempio matrice a
    for (i = 0; i < n; i++)
    {
        for (j = 0; j < n; j++)
        {
            matA[i][j] = rand() % 50;
        }
    }

    // alloco matrice b
    matB = (int **)calloc(n, sizeof(int *));
    for (i = 0; i < n; i++)
    {
        matB[i] = (int *)calloc(n, sizeof(int));
```

```
38      }
39      // riempio matrice b
40      for (i = 0; i < n; i++)
41      {
42          for (j = 0; j < n; j++)
43          {
44              matB[i][j] = rand() % 50;
45          }
46      }
47
48      // alloco matrice ris
49      matRes = (int **)calloc(n, sizeof(int *));
50      for (i = 0; i < n; i++)
51      {
52          matRes[i] = (int *)calloc(n, sizeof(int));
53      }
54
55      // prodotto matrice-matrice
56      t0 = MPI_Wtime();
57      for (i = 0; i < n; i++)
58      {
59          for (j = 0; j < n; j++)
60          {
61              matRes[i][j] = 0;
62              for (k = 0; k < n; k++)
63              {
64                  matRes[i][j] = matRes[i][j] + matA[i][k] * matB[k][j];
65              }
66          }
67      }
68      t1 = MPI_Wtime();
69      // printf("\n\n");
70      //  stampa matrice A
71      printf("Matrice A:\n");
72      for (i = 0; i < n; i++)
73      {
74          for (j = 0; j < n; j++)
75          {
76              printf("%d ", matA[i][j]);
77          }
78          printf("\n");
79      }
80      printf("\n\n");
81
82      // stampa matrice B
83      printf("Matrice B:\n");
84      for (i = 0; i < n; i++)
85      {
86          for (j = 0; j < n; j++)
87          {
88              printf("%d ", matB[i][j]);
89          }
90          printf("\n");
91      }
```

```
92      printf("\n\n");
93
94      // stampa matrice Res
95      printf("Matrice Risultato:\n");
96      for (i = 0; i < n; i++)
97      {
98          for (j = 0; j < n; j++)
99          {
100             printf("%d ", matRes[i][j]);
101         }
102         printf("\n");
103     }
104
105     time = t1 - t0;
106
107     printf("Il tempo di esecuzione e' stato: %e\n", time);
108     MPI_Finalize();
109
110     return 0;
111 }
```