

Università degli Studi di Napoli Federico II
Dipartimento di Ingegneria Elettrica e delle Tecnologie dell'Informazione

Corso di Laurea Magistrale in Informatica

Corso di Parallel and Distributed Computing

Prof. Giuliano Laccetti – Prof.^{ssa} Valeria Mele

**Sviluppo di un algoritmo
per il calcolo del prodotto matrice-matrice
in ambiente di calcolo parallelo
su architettura MIMD a memoria distribuita**

CANDIDATI:

CAROFANO Mario Gabriele - N97000437

NOVIELLO Francesco - N97000436

Anno Accademico 2023-2024

Indice

1 Definizione e analisi del problema	1
1.1 Il prodotto matrice-matrice	2
1.2 Broadcast Multiply Rolling	3
2 Errori, costanti e librerie	5
2.1 Errori	5
2.2 Costanti	7
2.3 Librerie	12
2.3.1 Libreria mpi.h	12
2.3.2 Libreria auxfunc.h	16
2.3.3 Libreria menufunc.h	18
2.3.4 Libreria csvfunc.h	19
2.3.5 Altre librerie	20
3 Creazione del file di esecuzione .pbs	23
3.1 Inizializzazione dell'ambiente di lavoro	23
3.2 Scelta dell'operazione da effettuare	25
3.3 Scelta della dimensioni delle matrici e dell'input	25
3.4 Applicazione della suite di testing	27
3.5 Scrittura del file .pbs	28
4 Implementazione dell'algoritmo	29
4.1 Inizializzazione dell'ambiente di lavoro	29
4.2 Inizializzazione dell'ambiente MPI	33
4.3 Lettura dei dati	33
4.4 Creazione della griglia bidimensionale	35
4.5 Creazione dei file di output	35
4.6 Lettura e/o generazione degli elementi delle matrici	35
4.7 Distribuzione delle matrici globali	37
4.8 Applicazione del BMR per il calcolo del prodotto matrice-matrice	37
4.9 Calcolo dei tempi di esecuzione	37
4.10 Stampa dell'output e terminazione	38
5 Analisi dei tempi e delle prestazioni	39

6 Conclusioni	43
6.1 Futuri sviluppi	44
A Elenco dei listati	45
A.1 constants.c	45
A.2 auxfunc.c	46
A.3 csvfunc.c	54
A.4 menufunc.c	56
A.5 menu.c	62
A.6 prova3.c	65
A.7 media-csv.c	72
Sitografia	75

Capitolo 1

Definizione e analisi del problema

L'algoritmo che viene presentato in questa relazione ha l'obiettivo di effettuare il calcolo del prodotto tra una matrice quadrata $A \in \mathbb{R}^{m \times m}$ ed una matrice quadrata $B \in \mathbb{R}^{m \times m}$. L'esecuzione del programma avviene in ambiente di calcolo parallelo su una griglia di $p \times p$ processi (dove il valore di p è un multiplo del valore di m) su architettura MIMD-DM ("*Multiple Instruction, Multiple Data*" a memoria distribuita).

- L'algoritmo è stato implementato in linguaggio C. La sezione 1.1 a pagina 2 illustra nel dettaglio lo pseudo-codice che implementa il calcolo del prodotto fra due matrici quadrate.
- Il programma utilizza la libreria *Message Passing Interface (MPI)* per la gestione dei processori dell'infrastruttura MIMD-DM e dei processi generati durante l'esecuzione.

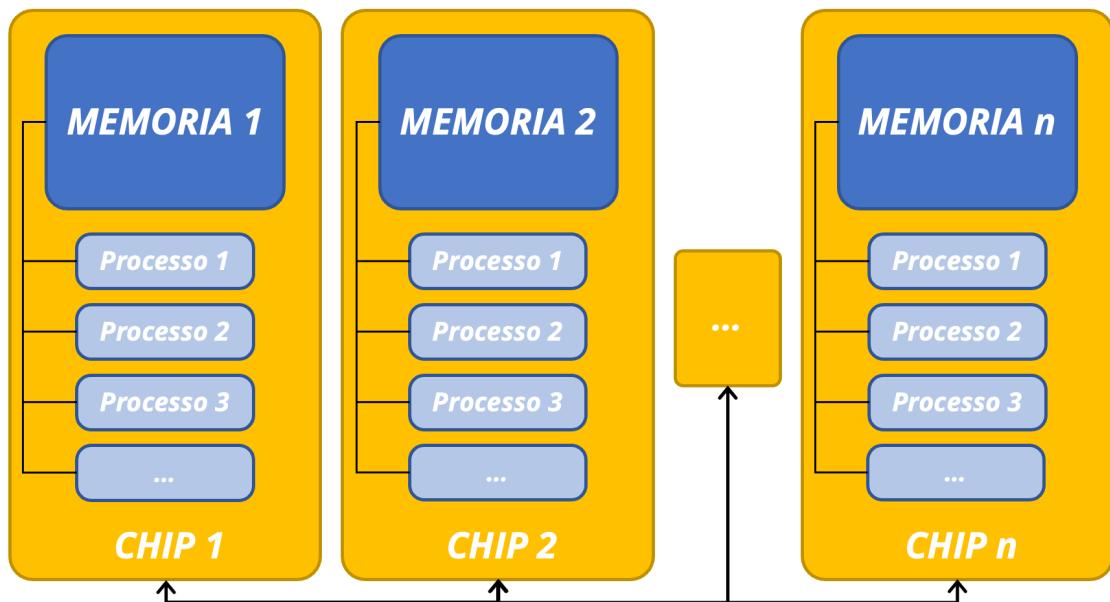


Figura 1.1: Esempio di architettura MIMD-DM

- Dato che il programma è progettato per essere eseguito su un’architettura MIMD-DM, è fondamentale una fase di distribuzione delle matrici sulla griglia di processi in esecuzione per bilanciare in modo ottimale il carico di lavoro e rendere, quindi, l’esecuzione il più efficiente possibile. Inoltre, dalle specifiche dell’algoritmo, è stato richiesto di utilizzare come strategia di comunicazione tra i processi della griglia il BMR (Broadcast Multiply Rolling). La sezione 1.2 a pagina 3 illustra nel dettaglio tale strategia per il calcolo su architettura parallela.

1.1 Il prodotto matrice-matrice

Siano $A = (a_{i,j})$, $B = (b_{i,j})$ due matrici quadrate di dimensione $n \times n$, con $0 \leq i \leq n$ e $0 \leq j \leq n$, e $C = A \cdot B = c_{i,j}$ una matrice quadrata di dimensione $n \times n$, dove $c_{i,j}$ indica il prodotto scalare della i -esima riga di A con la j -esima colonna di B , definito con la seguente equazione:

$$c_{i,j} = \sum_{k=0}^{n-1} a_{i,k} \cdot b_{k,j}, \quad 0 \leq i \leq n, \quad 0 \leq j \leq n \quad (1.1)$$

Cioè, per ottenere la matrice C , si devono eseguire il prodotto riga per colonna tra le righe della matrice A e le colonne della matrice B . Ciascuna di queste operazioni prevede la moltiplicazione degli elementi di riga e di colonna e l’ulteriore somma dei prodotti ottenuti, come mostrato nel seguente pseudo-codice:

```

for  $i \in [1, n]$  do
  for  $j \in [1, n]$  do
     $C[i][j] \leftarrow 0.$ 
    for  $k \in [1, n]$  do
       $C[i][j] \leftarrow C[i][j] + A[i][k] * B[k][j].$ 
    end for
  end for
end for

```

1.2 Broadcast Multiply Rolling

In ambiente di calcolo parallelo su architettura MIMD-DM, si assume che i processori siano distribuiti secondo una griglia bidimensionale quadrata cartesiana e che le sottomatrici quadrate delle matrici A e B in input siano distribuite in modo tale che al processore $p_{i,j}$ vengano assegnati i blocchi $A_{i,j}$ e $B_{i,j}$. Inoltre, si assume che le coordinate i, j dei processi sulla griglia siano crescenti da sinistra verso destra, e dall'alto verso il basso.

L'algoritmo per il prodotto matrice-matrice che utilizza la BMR procede in un numero di iterazioni dipendente dal numero dei processi in esecuzione. In particolare, si parla di \sqrt{p} iterazioni. In ogni iterazione k , per k che parte da 0, si ripetono le operazioni di Broadcast, Multiply e Rolling:

1. *Broadcast*

L'operazione di Broadcast consiste nell'inviare gli elementi delle sottomatrici A locali allocate nei processi della griglia che si trovano sulla k -esima diagonale (per $k = 0$ si intende la diagonale principale) a tutti gli altri processi che si trovano sulla medesima riga.

2. *Multiply*

L'operazione di Multiply consiste nel calcolare il prodotto matrice-matrice con le sottomatrici locali attualmente presenti nella memoria assegnata ad ogni processo della griglia, utilizzando l'implementazione dello pseudo-codice fornito nella sezione 1.1 a pagina 2.

3. *Rolling*

L'operazione di Rolling consiste nell'inviare gli elementi delle sottomatrici temporanee B (per temporaneo si intende la sottomatrice ricevuta all'iterazione precedente) al processo sulla stessa colonna ma sulla riga precedente.

Durante la prima iterazione (cioè per $k = 0$) l'operazione di Rolling non è richiesta in quanto tutti i processi già dispongono delle sottomatrici A, B locali per poter calcolare correttamente il prodotto.

Questa strategia di comunicazione, quindi, si itera fino a $k = \sqrt{p} - 1$ in cui ogni processo della griglia dispone della sottomatrice C locale con la moltiplicazione completata.

Capitolo 2

Errori, costanti e librerie

Si inizia la trattazione dell'algoritmo implementato introducendo alcuni file aggiuntivi realizzati dal team di sviluppo contenenti costanti, codici di errore e funzioni accessorie.

2.1 Errori

Nel file *constants.c* sono definiti i codici di errori, utili a rappresentare le cause di terminazione del programma. Per la loro definizione si utilizza la direttiva del preprocessore `#define` (principalmente, per una miglior leggibilità del codice) in modo tale che, prima dell'inizio della compilazione del programma, esso possa sostituire nel codice queste definizioni con i valori associati. Il codice è disponibile nella sezione A.1 dell'appendice A a pagina 45.

- `#define NOT_ENOUGH_ARGS_ERROR 1`

Si utilizza per segnalare all'utente che l'esecuzione del programma è stata terminata per mancanza di un numero sufficiente di argomenti.

- `#define EMPTY_ARG_ERROR 2`

Si utilizza per segnalare all'utente che l'esecuzione del programma è stata terminata siccome un argomento dato in input è vuoto.

- `#define INPUT_ARG_ERROR 3`

Si utilizza per segnalare all'utente che l'esecuzione del programma è stata terminata a causa di un'errata lettura di un argomento dato in input.

- `#define NOT_INT_ARG_ERROR 4`

Si utilizza per segnalare all'utente che l'esecuzione del programma è stata terminata poiché un argomento dato in input non è rappresentabile come intero.

- **#define INPUT_LINE_ERROR 5**

Si utilizza per segnalare all'utente che l'esecuzione del programma è stata terminata a causa di un errata lettura dello 'standard input stream' (stdin).

- **#define NOT_REAL_NUMBER_ERROR 6**

Si utilizza per segnalare all'utente che l'esecuzione del programma è stata terminata siccome non ha inserito un numero reale correttamente.

- **#define NOT_NATURAL_NUMBER_ERROR 7**

Si utilizza per segnalare all'utente che l'esecuzione del programma è stata terminata siccome non ha inserito un numero naturale correttamente.

- **#define NOT_IN_RANGE_ERROR 8**

Si utilizza per segnalare all'utente che l'esecuzione del programma è stata terminata siccome non ha inserito un numero compreso in un certo intervallo.

- **#define FILE_OPENING_ERROR 9**

Si utilizza per segnalare all'utente che l'esecuzione del programma è stata terminata a causa di un errore durante l'apertura di un file.

- **#define FILE_CLOSING_ERROR 10**

Si utilizza per segnalare all'utente che l'esecuzione del programma è stata terminata a causa di un errore durante la chiusura di un file.

- **#define MATRIX_DIMENSION_ERROR 11**

Si utilizza per segnalare all'utente che l'esecuzione del programma è stata terminata siccome ha inserito delle dimensioni per la matrice piu' grandi di quelle del file .csv.

- **#define PROCESSOR_QUANTITY_ERROR 12**

Si utilizza per segnalare all'utente che l'esecuzione del programma è stata terminata siccome ha scelto un numero di processori per il quale non è possibile generare una griglia bidimensionale uniforme.

- **#define ALLOCATION_ERROR 13**

Si utilizza per segnalare all'utente che l'esecuzione del programma è stata terminata siccome il programma non è riuscito ad allocare correttamente le matrici in memoria.

2.2 Costanti

Nel file *constants.c* sono definite anche le costanti, utili per la definizione di alcuni parametri nel codice. La loro definizione è identica a quanto spiegato nella sezione 2.1 a pagina 5.

- **#define DEBUG 1**

Si utilizza per scegliere se stampare o meno nel file di output generale dei messaggi di controllo per verificare lo stato dell'esecuzione del programma.

- **#define DEFAULT_SCELTA 0**

È il valore con cui viene inizializzata la variabile **scelta** nel file *menu.c*.

- Le seguenti 3 costanti sono valori interi che rappresentano le possibili scelte che può inserire l'utente nel menu' principale dell'applicazione.

In particolare:

1. **#define NO_TEST 1**

Si utilizza per calcolare il prodotto matrice-matrice sul cluster senza riservare un processore per ogni nodo. Questa tipologia d'esecuzione è riservata per i casi di test dove è necessario calcolare anche i tempi di esecuzione.

2. **#define TESTING_SUITE 2**

Si utilizza per eseguire la suite di testing progettata per verificare la correttezza dell'algoritmo del calcolo del prodotto matrice-matrice.

3. **#define EXIT_APPLICATION 3**

Si utilizza per terminare il menu' principale dell'applicazione.

- **#define DEFAULT_TEST 0**

È il valore con cui viene inizializzata la variabile **test** nel file *prova3.c*.

- Le seguenti 4 costanti sono valori interi che rappresentano le possibili scelte che può inserire l'utente nel menù della suite di test dell'applicazione.

In particolare:

1. `#define MULTIPLICATION_IDENTITY_TEST 1`

Per verificare la correttezza del prodotto di una matrice reale con la matrice identita' delle stesse dimensioni.

2. `#define MULTIPLICATION_TRANSPOSE_TEST 2`

Per verificare le proprieta' del prodotto matrice-matrice rispetto alle matrici trasposte, cioè una matrice in cui si invertono le righe con le colonne o viceversa.

3. `#define MULTIPLICATION_TRACE_TEST 3`

Per verificare le proprieta' del prodotto matrice-matrice rispetto alle traccia, cioè la somma di tutti gli elementi sulla diagonale principale.

4. `#define EXIT_TEST 4`

Si utilizza per chiudere il menù dell'applicazione dedicato alla scelta dei casi di test.

- `#define DEFAULT_INPUT 0`

È il valore con cui viene inizializzata la variabile `input` nel file `prova3.c`.

- Le seguenti 2 costanti si utilizzano per eseguire l'algoritmo del calcolo del prodotto matrice-matrice utilizzando esclusivamente:

1. `#define VALUES_FROM_INPUT 1`

Valori scelti dall'utente (o generati in modo pseudo-casuale).

2. `#define VALUES_FROM_CSV 2`

Valori recuperati da un file `.csv`.

- Le seguenti 2 costanti si utilizzano per scegliere il formato di visualizzazione le matrici di input e di output nei file di output:

1. `#define TAB_SPACE_PRINT 1`

Indica il formato "tab-space". Utile per visualizzare la matrice in due dimensioni.

2. `#define WOLFRAM_PRINT 2`

Indica il formato "Wolfram Language code". Utile per passare la matrice direttamente sul sito web <https://www.wolframalpha.com/> ed eseguire altre operazioni.

- `#define OP_MAX_QUANTITY 32`

Si utilizza per specificare il massimo numero di elementi della matrice che l'utente può inserire manualmente.

- `#define MIN_COEFF_TEST 1`

- `#define MAX_COEFF_TEST 3`

Si utilizzano, rispettivamente, come coefficienti minimo e massimo per determinare le dimensioni delle matrici nella creazione dei casi di test.

- `#define NO_TIME_CALC 0`

- `#define OK_TIME_CALC 1`

Si utilizza per personalizzare il flusso di esecuzione del programma in quei controlli dove si sceglie se effettuare (`OK_TIME_CALC`) o meno ((`NO_TIME_CALC`) il calcolo dei tempi di esecuzione.

- `#define NOME_PROVA "prova3"`

Si utilizza nella funzione createPBS() per personalizzare il file .pbs con il nome della prova corrente.

- `#define NODE_NUMBER "8"`

Si utilizza nella funzione createPBS() per personalizzare il file .pbs con il numero di nodi con i quali si vuole eseguire il programma.

- `#define NODE_PROCESS "8"`

Si utilizza nella funzione createPBS() per personalizzare il file .pbs con il numero di processori occupati da ogni nodo per l'esecuzione del programma.

- `#define MKDIR_PATH "/bin/mkdir"`

- `#define RM_PATH "/bin/rm"`

Si utilizzano per indicare alle funzioni system() e createPBS() i percorsi sul cluster degli eseguibili 'mkdir' e 'rm'.

- `#define COMPILER_PATH "/usr/lib64/openmpi/1.4-gcc/bin/mpicc"`

Si utilizza per indicare alla funzione createPBS() il percorso sul cluster del compilatore 'mpicc'.

- `#define EXECUTE_PATH "/usr/lib64/openmpi/1.4-gcc/bin/mpiexec"`

Si utilizza per indicare alla funzione createPBS() il percorso sul cluster del programma 'mpiexec' per eseguire jobs in Open MPI.

- **#define PBS_OUTPUT "\$PBS_O_HOME/"NOME_PROVA "/output"**

Si utilizza per indicare alla funzione createPBS() il percorso sul cluster della directory dove saranno salvati tutti i file di output e/o di errore dell'esecuzione.

- **#define PBS_WORKDIR "\$PBS_O_HOME/"NOME_PROVA "/codice"**

Si utilizza per indicare alla funzione createPBS() il percorso sul cluster della directory dove è memorizzato il codice del programma da compilare.

- **#define PBS_EXECUTABLE "\$PBS_O_HOME/"NOME_PROVA "/bin"**

Si utilizza per indicare alla funzione createPBS() il percorso sul cluster della directory dove saranno salvati tutti gli eseguibili generati dalla compilazione.

- **#define CSV_TIME_PATH ""NOME_PROVA "/output"**

Rappresenta il path del file .csv nella quale sono memorizzate tutte le informazioni necessarie allo studio dei casi di test. In particolare, la sua struttura è la seguente:

– **A_rows : int**

Indica il numero di righe della matrice A in input.

– **A_cols : int**

Indica il numero di colonne della matrice A in input.

– **B_rows : int**

Indica il numero di righe della matrice B in input.

– **B_cols : int**

Indica il numero di colonne della matrice B in input.

– **n_proc : int**

Indica il numero di processi attivati nel contesto d'esecuzione.

– **input : int**

Indica la provenienza dei valori di input se non è stato attivato nessun caso di test (può essere **VALUES_FROM_INPUT** o **VALUES_FROM_CSV**).

– **test : int**

Indica il caso di test eseguito.

– **time_tot**

Indica il tempo di esecuzione impiegato per il solo calcolo del prodotto matrice-matrice.

- **#define OP_MAX_VALUE 10**

Si utilizza come limite massimo dei valori della matrice o del vettore quando si sceglie di utilizzare numeri reali casuali.

- **#define ARGS_QUANTITY 8**

Indica il minimo numero di argomenti da passare in input al programma.

- **#define CSV_FIELD_PRECISION 10**

Si utilizza come limite massimo di cifre decimali dei valori della matrice quando si sceglie la lettura da file .csv.

- **#define CSV_FIELDS_SEPARATOR 44**

Indica il carattere utilizzato nel file .csv per separare un campo da quello successivo. Si utilizza il valore corrispondente nella tabella ASCII.

- **#define CSV_ROWS_SEPARATOR 10**

Indica il carattere utilizzato nel file .csv per separare una riga da quella successiva. Si utilizza il valore corrispondente nella tabella ASCII.

- **#define PATH_MAX_LENGTH 255**

Si utilizza come limite massimo per la lunghezza di un percorso di un file.

- **#define BROADCAST_TAG 101**

Si utilizza per assegnare un identificativo univoco alle comunicazioni di MPI durante il broadcast delle sottomatrici.

- **#define ROLLING_TAG 103**

Si utilizza per assegnare un identificativo univoco alle comunicazioni di MPI durante il rolling delle sottomatrici.

2.3 Librerie

In questa sezione sono elencate e descritte le funzioni adoperate dalla libreria *mpi.h* [1], dalle librerie sviluppate dal team di sviluppo e dalle altre librerie disponibili per il linguaggio C.

2.3.1 Libreria mpi.h



Figura 2.1: Il logo di "Open MPI Project"

Il progetto "Open MPI" è un'implementazione open source delle specifiche Message Passing Interface (MPI), sviluppata e mantenuta da un consorzio di partner accademici, di ricerca e industriali. Open MPI combina le competenze, le tecnologie e le risorse di tutta la comunità dell'High Performance Computing per costruire la migliore libreria MPI disponibile.

In particolare, le funzioni utilizzate dal team di sviluppo sono elencate di seguito:

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

Definisce l'insieme dei processi attivati (contesto), assegnandovi un identificativo.

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

Assegna ad ogni processore del communicator l'identificativo `id_proc` (sempre associato al contesto).

In particolare, viene passato `MPI_COMM_WORLD` al parametro in input `comm`, che indica il communicator a cui appartengono tutti i processi attivati.

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

Restituisce ad ogni processore di `comm` il numero di processori nel contesto.

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

Si utilizza questa funzione per inviare gli argomenti letti dal processore 0 (root) a tutti i processi del contesto, incluso se stesso.

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

Il processo che esegue questa funzione spedisce i primi *count* elementi nel buffer *buf* di tipo *datatype* al processo con identificativo *dest*. In particolare, l'identificativo *tag* individua univocamente l'invio nel contesto *comm*.

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

Il processo che esegue questa funzione riceve i primi *count* elementi nel buffer **buf** di tipo *datatype* dal processo con identificativo *source*. In particolare, l'identificativo *tag* individua univocamente la ricezione nel contesto *comm*, mentre *status* ne racchiude alcune informazioni.

- `int MPI_Barrier(MPI_Comm comm)`

Si utilizza questa funzione per restituire il controllo al chiamante solo dopo che tutti i processori del contesto *comm* hanno effettuato la chiamata.

- `double MPI_Wtime()`

Si utilizza per ottenere un valore di tempo in secondi rispetto ad un tempo arbitrario nel passato.

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

Si utilizza questa funzione per eseguire un'operazione collettiva su tutti i processi del contesto.

- `int MPI_Finalize()`

Determina la fine del programma MPI.

Da questo punto in poi non è possibile richiamare altre funzioni della libreria *mpi.h*.

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

Si utilizza per terminare tutti i processi nel communicator *comm* inviando ad ognuno di essi un SIGTERM.

- `int MPI_Cart_create(MPI_Comm comm_old, int ndims, int *dims, int *periods, int reorder, MPI_Comm *comm_cart)`

Questa funzione ritorna un riferimento ad un nuovo communicator, nel parametro in output `comm_cart`, nella quale i processi sono disposti in una griglia, a partire dai processi presenti nel communicator in input `comm_old`.

- Il parametro `ndims` indica il numero di dimensioni della griglia. Per la realizzazione di questo elaborato, è stata utilizzata una griglia bidimensionale: pertanto il valore passato al parametro `ndims` è 2.
- Il parametro `dims` indica quanti processi devono essere disposti sulle dimensioni della griglia. Per la realizzazione di questo elaborato, è stata utilizzata una griglia quadrata: pertanto il numero di processi su ogni dimensione è pari alla radice quadrata del numero di processi totale (se e solo questo è un quadrato perfetto).
- Il parametro `periods` indica quali dimensioni della griglia sono periodiche.
- Il parametro `reorder` indica se i processi possono essere riordinati oppure se è necessario che mantengano lo stesso rank del communicator precedente.

- `int MPI_Cart_coords(MPI_Comm comm, int rank, int maxdims, int *coords)`

Questa funzione restituisce nel vettore in output `coords` di dimensione `maxdims` le coordinate di un processo in un communicator a griglia sulla base del proprio `rank`.

- `int MPI_Cart_sub(MPI_Comm comm, int *remain_dims, MPI_Comm *comm_new)`

Questa funzione ritorna un riferimento ad un nuovo communicator, nel parametro in output `comm_new`. Dopo aver creato correttamente un communicator a griglia utilizzando la funzione `MPI_Cart_create()` nel parametro in input `comm`, si utilizza questa funzione per partizionarlo in un maggior numero di sotto-communicator con un numero di dimensioni minore.

In particolare, il parametro in input `remain_dims` è un vettore di valori booleani che indica alla funzione quali sono le dimensioni da partizionare.

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

Si utilizza questa funzione per costruire un nuovo `MPI_Datatype` da utilizzare successivamente nelle principali funzioni offerte dalla libreria `mpi.h` come, ad esempio, `MPI_Send` e `MPI_Recv`. Ogni blocco del nuovo tipo `newtype` è ottenuto concatenando un numero di blocchi di tipo `oldtype` per un numero di volte pari a `blocklength`.

- `int MPI_Type_create_resized(MPI_Datatype oldtype, MPI_Aint lb, MPI_Aint extent, MPI_Datatype *newtype)`

Si utilizza questa funzione per estendere la struttura del tipo `oldtype` ed ottenere un riferimento ad un nuovo tipo `newtype`, nel quale il limite inferiore è impostato al valore in input `lb` ed il limite superiore è impostato al valore `lb + extent`.

- `int MPI_Type_commit(MPI_Datatype *datatype)`

Si utilizza questa funzione per memorizzare definitivamente la struttura di `datatype`, ad esempio, dopo che si crea un nuovo tipo di dato con la funzione `MPI_Type_vector()` oppure se ne modificano le proprietà con `MPI_Type_create_resized()`.

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

L'esecuzione di questa funzione ha lo stesso risultato dell'esecuzione di un numero di `MPI_Send()` da parte del processo `root` pari al numero di processi in `comm`, e l'esecuzione di una singola `MPI_Recv()` da parte di tutti i processi in `comm` (incluso il processo radice).

Si utilizza questa funzione per consentire al processo `root` di distribuire un numero variabile di blocchi di dati di tipo `sendtype` dal buffer `sendbuf` a tutti i processi di `comm`.

2.3.2 Libreria auxfunc.h

Questa è una delle librerie implementate dal team di sviluppo e contiene alcune funzionalità aggiuntive per l'esecuzione del programma che calcola il prodotto matrice-matrice. Il codice è disponibile nella sezione A.2 dell'appendice A a pagina 46.

- `int argToInt(char *arg)`

La funzione riceve in input una stringa. Verifica che essa sia non vuota e che sia diversa dal null-termination character, quindi la converte in un *int* che verrà restituito in output.

- `double argToDouble(char *arg)`

La funzione riceve in input una stringa. Verifica che essa sia non vuota e che sia diversa dal null-termination character, quindi la converte in un *double* che verrà restituito in output.

- `void getRandomMatrix(double *mat, int mat_rows, int mat_cols)`

La funzione riceve in input il riferimento ad una matrice linearizzata, il numero di righe ed il numero di colonne. Si utilizza questa funzione per caricare all'interno della matrice *mat* elementi generati in modo pseudo-casuale.

- `void getIdentityMatrix(double *mat, int mat_rows, int mat_cols)`

La funzione riceve in input il riferimento ad una matrice linearizzata, il numero di righe ed il numero di colonne. Si utilizza questa funzione per caricare all'interno della matrice *mat* gli elementi tali da costruire la matrice identità (cioè, una matrice nulla con solo sulla diagonale principale il valore 1).

- `void getTransposeMatrix(double *A_mat, int A_rows, int A_cols, double *B_mat, int B_rows, int B_cols)`

La funzione riceve in input il riferimento a 2 matrici linearizzate, il numero delle loro righe ed il numero delle loro colonne. Si utilizza questa funzione per caricare all'interno della matrice *B_mat* gli elementi tali da costruire la matrice trasposta di *A_mat*, cioè invertendo le righe con le colonne o viceversa.

- `double* scatterMatrixToGrid(double* mat, int mat_rows, int mat_cols, int* loc_rows, int* loc_cols, int id_grid, int* grid_dim, int* grid_coords, MPI_Comm comm_grid)`

La funzione riceve in input il riferimento ad una matrice linearizzata, il numero di righe ed il numero di colonne. Restituisce in output il riferimento ad una nuova matrice linearizzata (con il nuovo numero di righe *loc_rows* ed il nuovo numero di colonne *loc_cols*) che contiene il contributo locale distribuito equamente dalla matrice globale *mat* in input tra tutti i processi di *comm* utilizzando la funzione *MPI_Scatterv* della libreria *mpi.h*.

- `void fprintfMatrix(FILE* out_file, double* mat, int rows, int cols, const char* format)`

Consente di stampare gli elementi di una matrice linearizzata `mat` in `out_file` tramite i due formati di visualizzazione introdotti nella sezione 2.2 a pagina 7.

- `void bmr_broadcast(double* mat, double* tmp, int rows, int cols, int step, MPI_Comm comm_grid, MPI_Comm comm_row)`

Questa funzione viene invocata all'interno della funzione `bmr()` per inviare ai processi della griglia appartenenti alla stessa riga `e`, cioè, appartenenti al communicator `comm_row`, il contributo locale della diagonale indicata dall'iterazione corrente `step` (per `step = 0` si intende la diagonale principale).

- `void bmr_multiply(double* A_mat, double* B_mat, double* C_mat, int A_rows, int A_cols, int B_rows, int B_cols, int C_rows, int C_cols, MPI_Comm comm)`

Questa funzione viene invocata all'interno della funzione `bmr()` per calcolare il contributo locale del prodotto matrice-matrice.

- `void bmr_rolling(double* mat, double* tmp, int rows, int cols, int step, MPI_Comm comm_col)`

Questa funzione viene invocata all'interno della funzione `bmr()` per inviare al processo precedente (in base all'iterazione corrente) nello stesso communicator di colonna `comm_col` il contributo locale.

- `void bmr(double* A_mat, double* B_mat, double* C_mat, int A_rows, int A_cols, int B_rows, int B_cols, int C_rows, int C_cols, MPI_Comm comm, MPI_Comm comm_row, MPI_Comm comm_col)`

Questa funzione è l'implementazione della strategia di comunicazione "Broadcast Multiply Rolling". In particolare: l'operazione "Broadcast" è implementata nella funzione `bmr_broadcast()`; l'operazione "Multiply" è implementata nella funzione `bmr_multiply()`; l'operazione "Rolling" è implementata nella funzione `bmr_rolling()`.

Inoltre, se il numero di processori del communicator `comm` è pari a 1, allora questa funzione esegue solo una chiamata alla funzione `bmr_multiply()`, in quanto non è necessario applicare tale strategia.

2.3.3 Libreria menufunc.h

Questa è una delle librerie implementate dal team di sviluppo e contiene alcune funzionalità aggiuntive per la gestione e presentazione del menù, nonchè per la creazione dei file *.pbs*. Il codice è disponibile nella sezione A.4 dell'appendice A a pagina 56.

- `void printTitle()`

Si utilizza questa funzione per stampare all'inizio del menù principale dell'applicazione il titolo dell'elaborato.

- `double getNumberFromInput()`

La funzione effettua l'analisi del valore inserito nello 'standard input stream' e ne restituisce la rappresentazione in un double.

- `int getIntegerFromInput()`

La funzione effettua l'analisi del valore inserito nello 'standard input stream' e ne restituisce la rappresentazione in un intero.

- `void checkScelta(int scelta, int lim_inf, int lim_sup)`

La funzione riceve in input il numero scelto, il limite inferiore e quello superiore del range di definizione e verifica se il numero vi appartiene.

- `void createPBS(int A_rows, int A_cols, int B_rows, int B_cols, int n_proc, int input, int test, int time_calc, int pbs_count)`

La funzione riceve in input le dimensioni delle matrici da dare in input al programma che calcola il prodotto matrice-matrice, il numero di processi da attivare nell'ambiente parallelo, la provenienza dell'input, il caso di test da eseguire e l'opzione di calcolo dei tempi di esecuzione. Si utilizza per creare dinamicamente il file *.pbs* necessario alla compilazione e all'esecuzione del programma per il calcolo del prodotto matrice-matrice sul cluster.

2.3.4 Libreria csvfunc.h

Questa è una delle librerie implementate dal team di sviluppo e contiene alcune funzionalità aggiuntive per la lettura e/o scrittura o, più in generale, per la gestione dei file *.csv* in input/output sul programma che calcola il prodotto matrice-matrice. Il codice è disponibile nella sezione A.3 dell'appendice A a pagina 54.

- `void getDimensionsFromCSV(FILE* csv_file, int* size, int* rows_csv, int * cols_csv)`

Si utilizza per recuperare alcune informazioni riguardo la struttura del file *.csv* in input, tra cui la dimensione, il numero di righe ed il numero di campi separati da virgola.

- `void getMatrixFromCSV(FILE* csv_file, double* mat, int rows_mat, int cols_mat, MPI_Comm comm)`

Si utilizza per copiare gli elementi presenti nel file *.csv* in input, in una matrice linearizzata `mat` di numeri reali di dimensioni `rows_mat, cols_mat`.

- `void writeTimeCSV(const char* csv_path, int A_rows, int A_cols, int B_rows, int B_cols, int n_proc, int input, int test, double time_tot, MPI_Comm comm)`

La funzione riceve in input tutte le informazioni necessarie allo studio dei tempi di esecuzione impiegati dai casi di test. Si utilizza per creare un file *.csv* al percorso specificato dal parametro in input `csv_path`, in modo da studiare l'output dell'esecuzione del programma più semplicemente.

2.3.5 Altre librerie

In questa sezione, sono elencate tutte le altre librerie aggiuntive del linguaggio C utilizzate per l'implementazione sia del programma che mostra il menù che del programma che calcola effettivamente il prodotto matrice-matrice.

- ***stdio.h***[2]

È la libreria utilizzata in C per gestire i flussi di dati in input / output grazie alla definizione di altri tipi di dato, come `FILE`, `size_t` e `ssize_t`, e altre funzioni, come `int fclose (FILE* stream)`, `FILE* fopen (const char* filename, const char* mode)`, `int fprintf (FILE * stream, const char * format, ...)`, la funzione `ssize_t getline(char **lineptr, size_t *n, FILE *stream)` [3] per inizializzare un buffer con caratteri estratti dal flusso di input specificato, `int printf (const char * format, ...)` e `int sprintf (char * str, const char * format, ...)`.

- ***stdlib.h***[4]

Questa libreria definisce alcune funzioni utili per diversi scopi, tra cui:

1. Gestione dinamica della memoria con `void* malloc (size_t num, size_t size)` e `void free (void* ptr)`.
2. Generazione di numeri pseudo-casuali con `int rand (void)` e `void srand (unsigned int seed)`.
3. Comunicazione con l'ambiente shell sottostante utilizzando `int system (const char* command)` e `void exit (int status)`.
4. Conversione degli argomenti passati in input al programma con `long int strtol (const char* str, char** endptr, int base)` e `double strtod (const char* str, char** endptr)`.

- ***errno.h***[5]

All'interno di questa libreria è definita la macro `int errno`, inizializzata a 0, utilizzabile in lettura e scrittura per gestire le casistiche di errore, assegnandole valori diversi da 0.

- ***limits.h***[6]

All'interno di questa libreria sono definite alcune costanti utili alla corretta rappresentazione dei valori interi nel programma, tra cui `int INT_MIN` e `int INT_MAX`.

- ***string.h***[7]

È una libreria che offre funzionalità per la gestione delle stringhe e, in generale, degli array. Nel codice si utilizza la funzione `size_t strlen (const char *str)` per controllare che la lunghezza degli argomenti in input non sia nulla.

- ***math.h***[8]

Questa libreria sono definite funzioni che implementano le più comuni operazioni e trasformazioni matematiche, tra cui `double log2 (double x)`, `double pow (double base, double exponent)`.

- ***time.h***[9]

È una libreria che offre funzionalità per la gestione del tempo. Si utilizza il tipo di dato `time_t` e la funzione `time_t time (time_t* timer)` per ottenere l'istante di tempo corrente (da utilizzare poi come seme per la generazione di numeri pseudo-casuali).

- ***ctype.h***[10]

È una libreria che offre funzionalità per la gestione e trasformazione dei caratteri. Si utilizza la funzione `int isdigit (int c)` per verificare che un carattere inserito nello stream di input sia una cifra valida.

Capitolo 3

Creazione del file di esecuzione .pbs

Questo capitolo è dedicato alla descrizione del menù di creazione dei file *.pbs*.

Si è pensato di implementare questo programma secondario per generare in modo automatizzato differenti file *.pbs* per i differenti test da eseguire, in modo tale da evitare un eccessivo tempo di risposta da parte del cluster che comporterebbe un blocco delle esecuzioni in coda.

Ogni file *.pbs* generato è personalizzato in base alle scelte effettuate dall'utente come descritto nelle seguenti sezioni di questo capitolo, dove si esamineranno in dettaglio (attraverso degli screenshot del terminale) le scelte che può eseguire l'utente che, alla fine dell'esecuzione, diventeranno gli argomenti in input passati al programma del calcolo del prodotto matrice-matrice.

Il codice è disponibile nella sezione A.5 dell'appendice A a pagina 62.

3.1 Inizializzazione dell'ambiente di lavoro

All'avvio del programma, si definiscono e inizializzano le variabili da utilizzare:

- `int mat_dim = 0`

È una variabile di tipo *int* per memorizzare la dimensione delle matrici *A, B*.

- `int A_rows = 0`

- `int A_cols = 0`

- `int B_rows = 0`

- `int B_cols = 0`

Sono delle variabili di tipo *int* per memorizzare singolarmente le dimensioni delle righe della matrice *A*, delle colonne della matrice *A*, delle righe della matrice *B* e delle colonne della matrice *B*, rispettivamente.

- **`int scelta = DEFAULT_SCELTA`**

È una variabile di tipo *int* per memorizzare le scelte fatta dall'utente nel menù principale dell'applicazione.

- **`int input = DEFAULT_INPUT`**

È una variabile di tipo *int* per memorizzare la scelta dell'utente nel menù relativamente all'input dell'algoritmo.

- **`int test = DEFAULT_TEST`**

È una variabile di tipo *int* per memorizzare la scelta dell'utente nel menù della suite di test dell'applicazione.

- **`int i = 0`**

È una variabile di tipo *int* utilizzata come iteratore nei cicli *for*.

- **`int pbs_count = 1`**

È una variabile di tipo *int* per memorizzare la numerazione dei file *.pbs* da creare.

3.2 Scelta dell'operazione da effettuare

Appena il programma si mostra all'utente, si chiede di scegliere se effettuare il calcolo del prodotto matrice-matrice inserendo manualmente i valori da linea di comando o da lettura di file .csv, oppure se eseguire un caso di test dalla suite di testing progettata dal team di sviluppo.

Figura 3.1: Avvio del menù per la creazione del file *.pbs*.

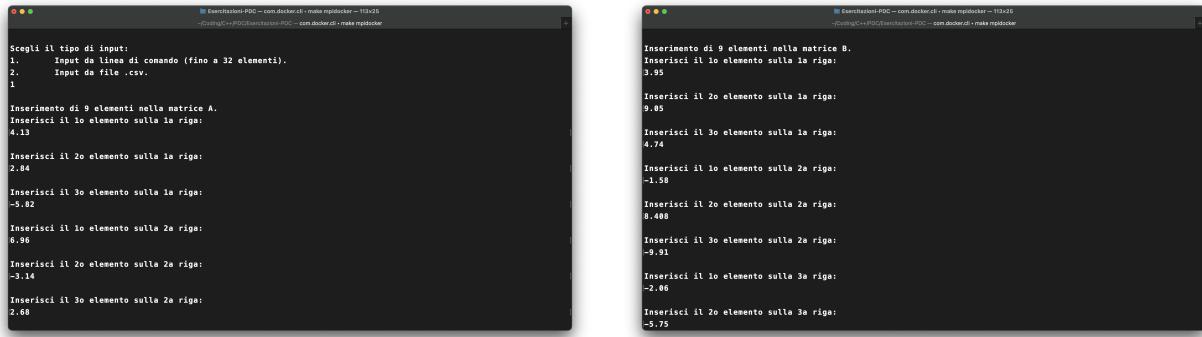
3.3 Scelta della dimensioni delle matrici e dell'input

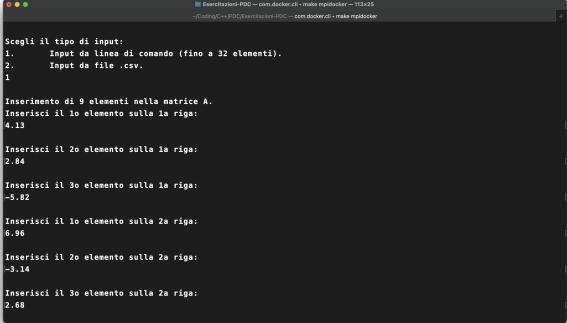
Nel caso in cui l'utente scelga di eseguire il prodotto matrice-matrice senza l'esecuzione di uno dei casi di test proposti, viene chiesta la dimensione delle matrici (che deve essere almeno pari a 1). In particolare, come richiesto dalle specifiche dell'algoritmo, la dimensione è unica e, di conseguenza, entrambe le matrici A, B sono matrici quadrate.

Successivamente, gli viene chiesto il tipo di input per i valori delle matrici che, come detto prima, può essere da linea di comando o da lettura di file .csv.

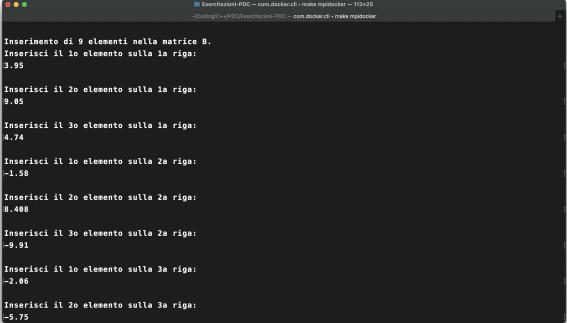
La funzione *createPBS*, quindi, distingue questi due casi:

1. Nel caso *VALUES_FROM_INPUT*, si permette l'inserimento dei valori delle due matrici A, B , solo se tale quantità è minore o uguale a *OP_MAX_QUANTITY* come mostrato nella figura 3.2, altrimenti sono assegnati dei valori generati in modo pseudo-casuale.





(a) Matrice A



(b) Matrice B

Figura 3.2: Inserimento dei valori delle matrici da linea di comando.

2. Nel caso *VALUES_FROM_CSV*, si permette l'inserimento di due pathname relativi ai due file .csv che si vogliono leggere per inserire i valori delle matrici A, B , rispettivamente.

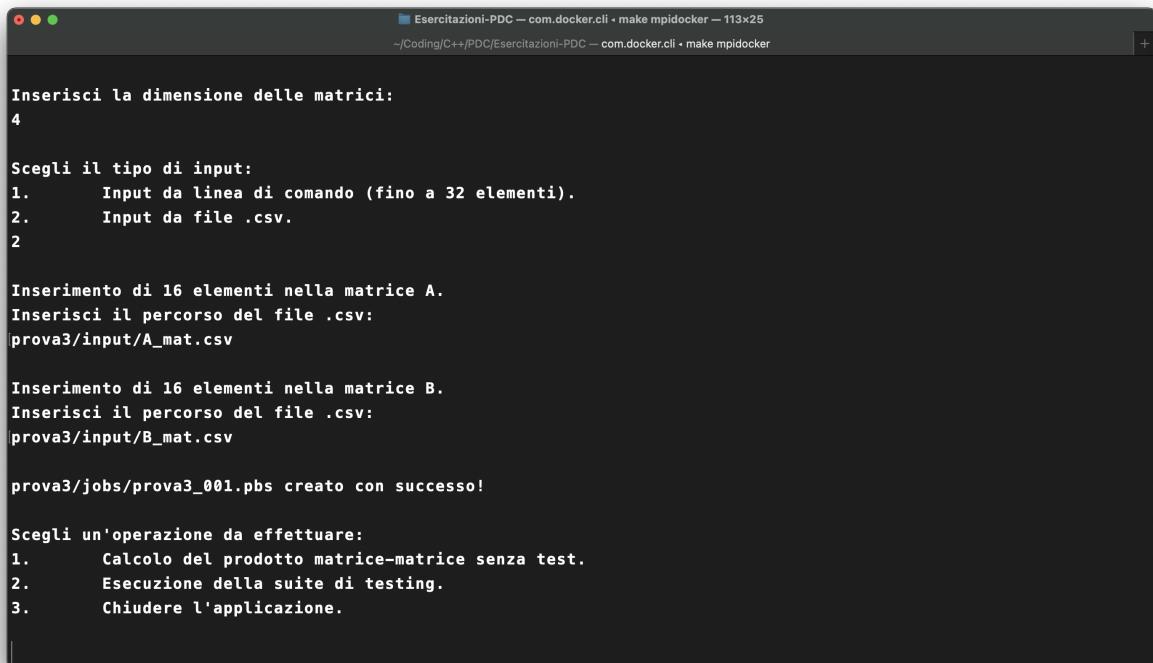
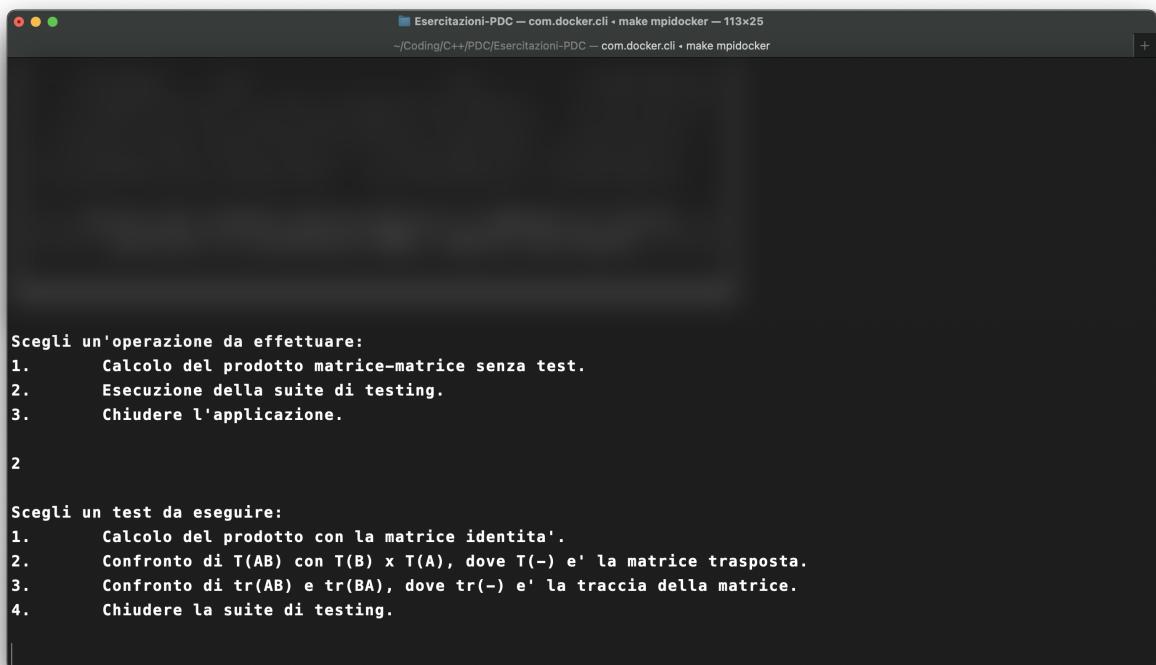


Figura 3.3: Inserimento dei valori da lettura di file .csv.

3.4 Applicazione della suite di testing

Come si vede nella figura 3.1 a pagina 25, dal menù l'utente può procedere alla creazione del file *.pbs* anche selezionando l'esecuzione di un caso di test. Nella suite di testing implementata sono disponibili:

- Calcolo del prodotto con la matrice identità.
- Confronto di $T(AB)$ con $T(B) \cdot T(A)$, dove la funzione $T(-)$ indica la matrice trasposta.
- Confronto di $tr(AB)$ e $tr(BA)$, dove la funzione $tr(-)$ indica la traccia della matrice.



```
Esercitazioni-PDC - com.docker.cli - make mpidocker - 113x25
~/Coding/C++/PDC/Esercitazioni-PDC — com.docker.cli - make mpidocker

Scegli un'operazione da effettuare:
1.    Calcolo del prodotto matrice-matrice senza test.
2.    Esecuzione della suite di testing.
3.    Chiudere l'applicazione.

2

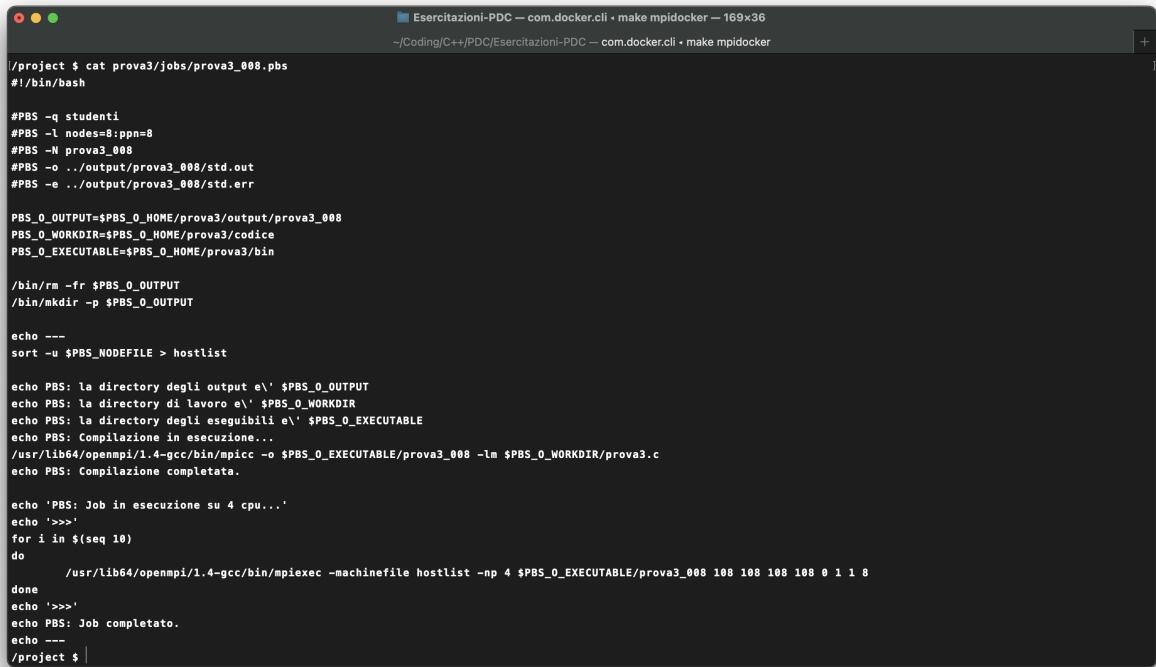
Scegli un test da eseguire:
1.    Calcolo del prodotto con la matrice identita'.
2.    Confronto di T(AB) con T(B) x T(A), dove T(-) e' la matrice trasposta.
3.    Confronto di tr(AB) e tr(BA), dove tr(-) e' la traccia della matrice.
4.    Chiudere la suite di testing.
```

Figura 3.4: Suite di testing.

3.5 Scrrittura del file .pbs

Al termine dell'esecuzione del programma del menù, come si evince dalla figura 3.3 a pagina 26, se il processo di creazione del file *.pbs* termina con successo, allora si mostrano i pathname dove sono collocati i file appena creati.

La figura 3.5 mostra un esempio di file *.pbs* per il prodotto di una matrice 108×108 di valori reali con la matrice identità della stessa dimensione (in riferimento al test MULTIPLICATION_IDENTITY_TEST) da eseguire su 4 processori calcolando, infine, il tempo di esecuzione impiegato da 10 esecuzioni in sequenza (si veda il ciclo for).



```
#!/bin/bash

#PBS -q studenti
#PBS -l nodes=8:ppn=8
#PBS -N prova3_008
#PBS -o .../output/prova3_008/std.out
#PBS -e .../output/prova3_008/std.err

PBS_O_OUTPUT=$PBS_O_HOME/prova3/output/prova3_008
PBS_O_WORKDIR=$PBS_O_HOME/prova3/codice
PBS_O_EXECUTABLE=$PBS_O_HOME/prova3/bin

/bin/rm -fr $PBS_O_OUTPUT
/bin/mkdir -p $PBS_O_OUTPUT

echo ---
sort -u $PBS_NODEFILE > hostlist

echo PBS: la directory degli output e\' $PBS_O_OUTPUT
echo PBS: la directory di lavoro e\' $PBS_O_WORKDIR
echo PBS: la directory degli eseguibili e\' $PBS_O_EXECUTABLE
echo PBS: Compilazione in esecuzione...
/usr/lib64/openmpi/1.4-gcc/bin/mpicc -o $PBS_O_EXECUTABLE/prova3_008 -lm $PBS_O_WORKDIR/prova3.c
echo PBS: Compilazione completata.

echo 'PBS: Job in esecuzione su 4 cpu...'
echo '***'
for i in $(seq 10)
do
    /usr/lib64/openmpi/1.4-gcc/bin/mpiexec -machinefile hostlist -np 4 $PBS_O_EXECUTABLE/prova3_008 108 108 108 108 0 1 1 8
done
echo '***'
echo PBS: Job completato.
echo ---
/project $ |
```

Figura 3.5: Esempio di file *.pbs*

Capitolo 4

Implementazione dell'algoritmo

Questo capitolo è dedicato alla descrizione dell'esecuzione del programma sviluppato per il calcolo del prodotto matrice-matrice in ambiente di calcolo parallelo. Si esamineranno gli input ad esso passati e gli output attesi al termine della sua esecuzione. In particolare, ogni sezione di questo capitolo descrive una diversa fase della sua esecuzione, ognuna delle quali termina con una stampa di controllo nel file di output *std.out*. Il codice è disponibile nella sezione A.6 dell'appendice A a pagina 65.

4.1 Inizializzazione dell'ambiente di lavoro

Nella fase iniziale dell'esecuzione, si definiscono e inizializzano le variabili da utilizzare:

- `int input = DEFAULT_INPUT`
- `int test = DEFAULT_TEST`
- `int pbs_count = 0`

Si veda la descrizione nella sezione 3.1 a pagina 23.

- `int time_calc = NO_TIME_CALC`

È una variabile di tipo intero che memorizza la scelta dell'utente di calcolare o meno il tempo di esecuzione impiegato per il solo prodotto matrice-matrice.

- `int check_test = 1`

È una variabile di tipo intero che memorizza se il caso di test eseguito è terminato con successo o meno. Si veda la sezione 6.1 a pagina 61.

- `int i = 0`
- `int j = 0`
- `int k = 0`

Sono variabili di tipo intero utilizzate genericamente nei cicli *for*.

- `double *A_mat = NULL`
- `int A_rows = 0`
- `int A_cols = 0`

È un array di elementi *double* che ha il compito di memorizzare gli elementi della prima matrice in input (solo il processore radice - con identificativo 0 - lo può allocare). Le variabili di tipo intero *A_rows* e *A_cols*, invece, memorizzano le dimensioni di tale matrice.

- `double *B_mat = NULL`
- `int B_rows = 0`
- `int B_cols = 0`

È un array di elementi *double* che ha il compito di memorizzare gli elementi della seconda matrice in input (solo il processo radice - con identificativo 0 - lo può allocare). Le variabili di tipo intero *B_rows* e *B_cols*, invece, memorizzano le dimensioni di tale matrice.

- `double *loc_A_mat = NULL`
- `int loc_A_rows = 0`
- `int loc_A_cols = 0`

È un array di elementi *double*, allocato da tutti i processi, che ha il compito di memorizzare gli elementi della prima matrice distribuiti equamente dal processo radice. Le variabili di tipo intero *loc_A_rows* e *loc_A_cols*, invece, memorizzano le dimensioni di tale matrice.

- `double *loc_B_mat = NULL`
- `int loc_B_rows = 0`
- `int loc_B_cols = 0`

È un array di elementi *double*, allocato da tutti i processi, che ha il compito di memorizzare gli elementi della seconda matrice distribuiti equamente dal processo radice. Le variabili di tipo intero *loc_B_rows* e *loc_B_cols*, invece, memorizzano le dimensioni di tale matrice.

- `double *loc_C_mat = NULL`
- `int loc_C_rows = 0`
- `int loc_C_cols = 0`

È un array di elementi `double`, allocato da tutti i processi, che ha il compito di memorizzare gli elementi locali del risultato del prodotto matrice-matrice tra le due matrici globali date in input. Le variabili di tipo intero `loc_C_rows` e `loc_C_cols`, invece, memorizzano le dimensioni di tale matrice.

- `double time_start = 0.0`
- `double time_end = 0.0`

Sono variabili di tipo `double` che memorizzano gli istanti di tempo in cui inizia e finisce l'applicazione del BMR per il calcolo del prodotto matrice-matrice, rispettivamente.

- `double time_loc = 0.0`
- `double time_tot = 0.0`

Sono variabili di tipo `double` che memorizzano il tempo, locale di ogni processo e totale di tutti i processi, impiegato dal programma per il calcolo del prodotto matrice-matrice, rispettivamente.

- `double grid_tmp = 0`

È una variabile di tipo `double` utilizzata per memorizzare la radice quadrata del numero totale di processi in esecuzione nel contesto, per verificare la possibilità di costruire una griglia bidimensionale quadrata.

- `int n_proc = 0`

È una variabile di tipo intero che memorizza il numero di processi nel contesto (si veda la descrizione di `MPI_Comm_size()` a pagina 12).

- `int id_proc = 0`

È una variabile di tipo intero che memorizza il rank di un processo nel contesto (si veda la descrizione di `MPI_Comm_rank()` a pagina 12).

- `int grid_rows = 0`

- `int grid_cols = 0`

Sono variabili di tipo `int` che memorizzano le dimensioni della griglia bidimensionale di processi da costruire per l'esecuzione del programma.

- `int *grid_dim = NULL`

È un array di tipo `int` che memorizza le dimensioni della griglia bidimensionale di processi da costruire con la funzione `MPI_Cart_create()` (si veda la descrizione a pagina 12).

- `int *period = NULL`

È un array di tipo `int` che memorizza se le dimensioni della griglia bidimensionale di processi sono periodiche (si veda la descrizione di `MPI_Cart_create()` a pagina 12).

- `int *grid_coords = NULL`

È un array di tipo `int` che memorizza le coordinate di ogni processo nella griglia bidimensionale di processi (si veda la descrizione di `MPI_Cart_coords()` a pagina 12).

- `int *remain_dims = NULL`

È un array di tipo `int` che memorizza su quale dimensione creare un nuovo sotto-comunicatore per la griglia bidimensionale di processi (si veda la descrizione di `MPI_Cart_sub()` a pagina 12).

- `FILE *csv_A_mat = NULL`

- `FILE *csv_B_mat = NULL`

- `FILE *out_file = NULL`

Sono variabili di tipo puntatore a `FILE` che consentono di controllare un file in lettura e/o scrittura. In particolare: l'oggetto `csv_A_mat` punta ad un file `.csv` contenente gli elementi della prima matrice in input; l'oggetto `csv_B_mat` punta ad un file `.csv` contenente gli elementi della seconda matrice in input; l'oggetto `out_file` è utilizzato per creare un nuovo file con estensione `.out` nel quale i processi scrivono le proprie matrici locali in input e output.

- `char out_path[PATH_MAX_LENGTH] = {}`

È una array di tipo `char` di lunghezza `PATH_MAX_LENGTH` utilizzato per memorizzare il pathname del file di output generato da ogni singolo processo.

- *MPI_Comm comm*
- *MPI_Comm comm_row*
- *MPI_Comm comm_col*

Sono variabili di tipo *MPI_Comm* che indicano i comunicatori utilizzati durante l'esecuzione del programma. In particolare: il comunicatore *comm* può indicare alternativamente

MPI_COMM_WORLD oppure il nuovo comunicatore di griglia, rispettivamente se il numero di processi attivati è 1 o maggiore di 1; il comunicatore *comm_row* indica il sotto-comunicatore della griglia di processi contenente tutti i processi sulla stessa riga; il comunicatore *comm_col* indica il sotto-comunicatore della griglia di processi contenente tutti i processi sulla stessa colonna.

4.2 Inizializzazione dell'ambiente MPI

Dopo l'inizializzazione delle variabili, si procede con la configurazione dell'ambiente di lavoro per un calcolatore basato sull'architettura MIMD-DM utilizzando la libreria *MPI*.

Si utilizzano le funzioni *MPI_Init()*, *MPI_Comm_rank()* e *MPI_Comm_size()* in modo tale che, al termine della loro esecuzione, tutti i processi sono capaci di interagire e di eseguire operazioni di sincronizzazione fra di loro, hanno a disposizione il proprio rank nel contesto e, infine, conoscono il numero totale di processi in esecuzione.

4.3 Lettura dei dati

A questo punto, al processo 0 (root) si affida il compito di leggere e verificare i dati forniti in input al programma tramite l'argument vector (*argv*). In particolare, il minimo numero di argomenti richiesti in input dal programma (*argc*) è dettato dal valore della costante *ARGS_QUANTITY* nel file *constants.c*.

1. *A_rows = argToInt(argv[1])*

È il numero di righe della prima matrice in input.

2. *A_cols = argToInt(argv[2])*

È il numero di colonne della prima matrice in input.

3. *B_rows = argToInt(argv[3])*

È il numero di righe della seconda matrice in input.

4. *B_cols = argToInt(argv[4])*

È il numero di colonne della seconda matrice in input.

5. `input = argToInt(argv[5])`

Indica la provenienza dei valori per le due matrici in input. Gli unici due valori che può assumere questa variabile sono specificati nella sezione 2.2 a pagina 7.

6. `test = argToInt(argv[6])`

Indica il caso di test scelto per l'esecuzione. I valori che può assumere questa variabile sono specificati nella sezione 2.2 a pagina 7.

7. `time_calc = argToInt(argv[7])`

Permette di verificare se è richiesto il calcolo, la stampa ed il salvataggio del tempo di esecuzione impiegato. Gli unici due valori che può assumere questa variabile sono specificati nella sezione 2.2 a pagina 7.

8. `pbs_count = argToInt(argv[8])`

È l'identificativo numerico del file `.pbs` associato all'esecuzione del programma. Questa variabile viene utilizzata per indicare correttamente il pathname della directory nella quale salvare tutti i file di output.

Una volta completata la lettura dei dati in input, si procede alla loro verifica sulla base delle specifiche dell'algoritmo:

1. Se le dimensioni delle matrici non sono compatibili per il calcolo del prodotto matrice-matrice, cioè se il numero di righe della matrice B non è uguale al numero di colonne della matrice A , allora l'esecuzione del programma termina con l'errore `MATRIX_DIMENSION_ERROR`.
2. Se il numero di processori è nullo oppure non è un quadrato perfetto (per la costruzione della griglia di processori), allora l'esecuzione del programma termina con l'errore `PROCESSOR_QUANTITY_ERROR`.
3. Se la dimensione m delle matrici non è compatibile con la dimensione p della griglia di processori, cioè m non è multiplo di p , allora l'esecuzione del programma termina con l'errore `MATRIX_DIMENSION_ERROR`.

A questo punto dell'esecuzione del programma, si utilizza la funzione `MPI_Bcast()` per inviare gli argomenti letti dal processo 0 (root) a tutti i processi del contesto, incluso sè stesso.

4.4 Creazione della griglia bidimensionale

Come già spiegato nel capitolo 1 a pagina 1, nelle specifiche dell'algoritmo è stato richiesto di distribuire equamente i processi in esecuzione su di una griglia bidimensionale quadrata cartesiana.

L'implementazione di questo vincolo è stata realizzata utilizzando apposite funzioni della libreria *mpi.h*, tra cui *MPI_Cart_create()*, *MPI_Cart_coords()* e *MPI_Cart_sub*, le cui descrizioni sono fornite nella sezione 2.3.1 a pagina 12.

In particolare, la creazione della griglia bidimensionale viene eseguita dal programma se e solo se:

1. La verifica dei dati in input viene completata con successo (cioè, il numero di processi in esecuzione è un quadrato perfetto).
2. La quantità di processi in esecuzione è strettamente maggiore di 1 (perché in caso contrario non è necessaria la successiva fase di distribuzione delle matrici in input).

4.5 Creazione dei file di output

Per ottenere una maggior leggibilità dell'output finale ma, soprattutto, per evitare che le stampe da parte dei vari processi in esecuzione si sovrappongano, il team di sviluppo ha pensato di redirezionare le stampe delle matrici globali in input (per il solo processo 0), delle sottomatrici locali distribuite e delle sottomatrici locali contenenti il risultato finale in ulteriori file di output (con estensione *.out*) per ogni processo in esecuzione.

In particolare, questi file, tramite l'utilizzo della funzione *sprintf()* offerta dalla libreria *stdio.h* del linguaggio C, hanno un nome personalizzato in base al rank del processo e, se il numero di processi in esecuzione è strettamente maggiore di 1, anche in base alle coordinate cartesiane di tale processo nella griglia costruita durante la fase precedente.

4.6 Lettura e/o generazione degli elementi delle matrici

Una volta completata la distribuzione dei processi sulla griglia bidimensionale quadrata cartesiana, si affida al processo 0 (root) il compito di allocare lo spazio in memoria necessario per la lettura e/o generazione del valore degli elementi delle due matrici globali *A*, *B* in input. Questa sezione tratta in dettaglio l'implementazione dello switch-case del codice A.41 a pagina 67.

- Il caso *DEFAULT_TEST*

In questo caso, l'utente ha scelto dal menù di creazione del *.pbs* di non applicare alcun caso di test, ma di eseguire il programma per il calcolo del prodotto matrice-matrice inserendo manualmente i propri elementi. Si distinguono, quindi, i seguenti 2 sottocasi in base all'input che ha scelto:

1. Il caso *VALUES_FROM_INPUT*

Se la somma del numero di elementi della matrice A e del numero di elementi della matrice B non supera il valore dettato dalla costante *OP_MAX_QUANTITY*, allora gli elementi sono letti dall'argument vector *argv*; invece, se tale quantità è maggiore, allora ad ogni elemento si assegna un numero casuale reale compreso nell'intervallo $[-OP_MAX_VALUE, OP_MAX_VALUE] \in R$ tramite la funzione *getRandomMatrix()* progettata dal team di sviluppo.

2. Il caso *VALUES_FROM_CSV*

Si aprono in sola lettura i file (i cui pathname sono letti dall'argument vector *argv*) utilizzando la funzione *fopen()* della libreria *stdio.h* del linguaggio C. Quindi, si utilizza la funzione *getMatrixFromCSV()* progettata dal team di sviluppo per copiare i valori presenti nei file *.csv* all'interno delle matrici A, B .

- Il caso *MULTIPLICATION_IDENTITY_TEST*

Nel primo caso di test della suite di testing gli elementi della matrice A sono generati dalla funzione *getRandomMatrix()*, mentre gli elementi della matrice B sono generati dalla funzione *getIdentityMatrix()*.

Si sceglie per verificare le proprietà del prodotto matrice-matrice rispetto alla matrice identità I . Infatti, se il prodotto è eseguito correttamente deve risultare alla fine che:

$$C = A \cdot B \iff C = A, \quad \text{con } B = I \quad (4.1)$$

- Il caso *MULTIPLICATION_TRANSPOSE_TEST*

In questo secondo caso di test gli elementi delle matrici A, B sono generati dalla funzione *getRandomMatrix()*.

Si sceglie per verificare le proprietà del prodotto matrice-matrice rispetto alle matrici trasposte (indicate con T). Infatti, se il prodotto è eseguito correttamente deve risultare alla fine che:

$$C_1 = (A \cdot B)^T \iff C_2 = B^T \cdot A^T \quad (4.2)$$

- Il caso *MULTIPLICATION_TRACE_TEST*

L'ultimo caso di test è molto simile al secondo, in quanto gli elementi delle matrici A, B sono comunque generati dalla funzione *getRandomMatrix()*.

Tuttavia, il senso di questo test è quello di verificare le proprietà del prodotto matrice-matrice rispetto alla traccia *tr()*, cioè la somma di tutti gli elementi sulla diagonale principale. Quindi, se il prodotto è eseguito correttamente deve risultare alla fine che:

$$tr(C_1) = tr(C_2), \quad \text{con } C_1 = A \cdot B, \quad C_2 = B \cdot A \quad (4.3)$$

Al termine della lettura e/o generazione degli elementi delle matrici globali A, B , il processo 0 procede alla loro stampa nel proprio file di output.

4.7 Distribuzione delle matrici globali

Il prossimo passaggio è quello di distribuire equamente sulla griglia di processi le matrici globali A, B generate dal processore 0: esso, infatti, è l'unico adibito alla lettura dei dati in input. Questa operazione è fondamentale per distribuire equamente il carico di lavoro tra tutti i processi.

La distribuzione dipende dal numero di processori utilizzati per l'esecuzione del programma:

- Se il programma è in esecuzione su un singolo processo, allora non vi è alcuna distribuzione in quanto l'unico processo (cioè 0) è già in possesso delle matrici necessarie al calcolo del prodotto matrice-matrice.
- In caso contrario, cioè se il programma è in esecuzione su un maggior numero di processi, allora si utilizza la funzione `scatterMatrixToGrid()` appositamente progettata dal team di sviluppo per realizzare questa fase dell'esecuzione.

Al termine della distribuzione delle matrici globali A, B nelle matrici locali `loc_A_mat`, `loc_B_mat`, tutti i processi in esecuzione nel contesto procedono alla loro stampa nel proprio file di output.

4.8 Applicazione del BMR per il calcolo del prodotto matrice-matrice

Questa fase dell'esecuzione riguardo il calcolo del prodotto matrice-matrice vero e proprio utilizzando, come richiesto dalle specifiche dell'algoritmo, la strategia di comunicazione BMR secondo quanto riportato alla sezione 1.2 a pagina 3.

Essenzialmente, ogni processo alloca il vettore `loc_C_mat` dove conservare il proprio risultato locale del prodotto matrice-matrice utilizzando come dimensioni `loc_A_rows` per le righe e `loc_B_cols` per le colonne.

Si utilizza, infine, il sottoprogramma `bmr()` appositamente progettato dal team di sviluppo per eseguire le tre fasi di questa strategia, cioè:

1. *Broadcast*. La distribuzione lungo le righe con `bmr_broadcast()`.
2. *Multiply*. Il prodotto matrice-matrice con `bmr_multiply()`.
3. *Rolling*. La distribuzione lungo le colonne con `bmr_rolling()`.

4.9 Calcolo dei tempi di esecuzione

Nel caso in cui si sia scelto di calcolare i tempi di esecuzione, si imposta la variabile `time_start` come tempo di inizio del calcolo del prodotto matrice-matrice per ogni processore utilizzando la funzione `MPI_Wtime()`.

Al termine dell'applicazione del BMR, il programma imposta la variabile `time_end` come tempo di fine, utilizzando sempre la funzione `MPI_Wtime()`, e, quindi, calcolando la distanza tra tempo di fine e di inizio per stabilire l'intervallo temporale di esecuzione, cioè: `time_loc = time_end - time_start`.

Si effettua, poi, tramite la funzione `MPI_Reduce()` un'operazione collettiva con lo scopo di calcolare il tempo massimo fra tutti i tempi `time_loc` calcolati localmente. Infine:

1. Si stampa il tempo impiegato per il solo calcolo del prodotto matrice-matrice nel file di output `std.out` utilizzando la notazione scientifica.
2. Si salvano all'interno di un file `.csv` tutte le informazioni riguardanti l'esecuzione, cioè: le dimensioni di riga e colonna delle due matrici A, B in input, il test eseguito, il numero di processori, il tipo di input, il caso di test eseguito ed il tempo impiegato per il solo calcolo del prodotto matrice-matrice (in formato decimale).

4.10 Stampa dell'output e terminazione

Al termine dell'esecuzione del programma, quindi, si stampano i risultati ottenuti. Ogni processore inserisce nel proprio file `.out` la matrice locale contenente il risultato dell'applicazione del BMR, preceduta anche dalle dimensioni di tale matrice, utilizzando le funzioni `fprintfMatrix()` e `fprintf()`, rispettivamente. Terminato il lavoro da parte di ogni processore:

- Si libera lo spazio allocato in memoria per le matrici locali `loc_A_mat`, `loc_B_mat` e `loc_C_mat`.
- Se il numero di processori è maggiore di 1, allora all'inizio dell'esecuzione questi sono stati equamente distribuiti in una griglia e, pertanto, è necessario libeare la memoria anche per tutti i vettori di supporto. Si legga la sezione A.6 nell'appendice A a pagina 65 per maggiori dettagli.
- Si utilizza la funzione `fclose()` per chiudere correttamente i file di output aperti dai processori durante l'esecuzione.
- Si termina l'esecuzione del programma con la funzione `MPI_Finalize()`.

Quindi, in ultima analisi, l'esecuzione del comando `qsub` su tutti i file `.pbs` creati tramite l'apposito menù descritto nel capitolo 3 a pagina 23 produce diversi file in output:

1. Nella directory `/bin` si salva l'eseguibile del programma vero e proprio, la cui esecuzione è stata descritta in questo capitolo.
2. Nella directory `/output` si genera una sottodirectory per ogni file `.pbs` eseguito contenente: il file `std.out`, per le stampe di controllo dell'esecuzione; il file `std.err`, per il salvataggio delle informazioni di errore; infine, un numero di file con estensione `.out` pari al numero di processori coinvolti nell'esecuzione, ognuno contenente le matrici locali di input e output del prodotto matrice-matrice (solo il processore radice contiene anche le matrici globali).
3. Un file con estensione `.csv`, contenente tutti i tempi di esecuzione e altri parametri già descritti nella sezione 4.9 a pagina 37.

Capitolo 5

Analisi dei tempi e delle prestazioni

Per la valutazione delle performance del software sono stati scelti i seguenti parametri:

- **Tempo medio impiegato** $T_m(p)$

È la media aritmetica dei tempi $T(p)$ di ciascuna esecuzione di un singolo esperimento. Per ognuno di essi, sono state effettuate 10 esecuzioni sui 3 casi di test della suite di testing.

$$T_m(p) = \frac{\sum_{n=1}^{30} T(p)}{30} \quad (5.1)$$

- **Speed Up** $S(p)$

Misura la riduzione del tempo $T_m(p)$ impiegato per l'esecuzione con p processi rispetto al tempo $T(1)$ impiegato per l'esecuzione del programma su singolo processore.

$$S(p) = \frac{T(1)}{T(p)}, \text{ con } S(p)_{ideale} = p \quad (5.2)$$

- **Overhead** O_h

Misura quanto lo speed up effettivo differisca da quello ideale.

$$O_h = p \cdot T(p) - T(1), \text{ con } O_h > 0 \quad (5.3)$$

- **Efficienza** $E(p)$

Misura quanto l'esecuzione del programma sfrutta il parallelismo del calcolatore.

$$E(p) = S(p)/p, \text{ con } E(p) < E(p)_{ideale} = 1 \quad (5.4)$$

Per raccogliere queste misure sono stati eseguiti tutti i casi di test (nella suite di testing progettata dal team di sviluppo) su dimensioni delle matrici pari a 36×36 , 72×72 e 108×108 su un numero di processi in esecuzione pari a 1, 4 e 9 per un totale di 27 esperimenti.

Le seguenti figure mostrano i valori di tempo ottenuti ed i valori di speed up, overhead ed efficienza calcolati, sia in formato tabellare che tramite dei grafici (disegnati utilizzando MS Excel).

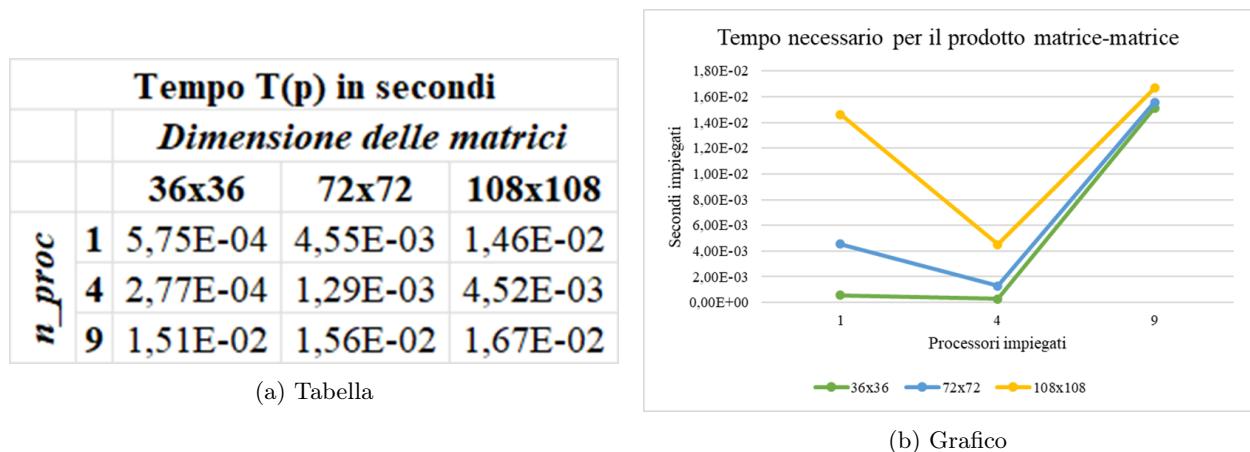


Figura 5.1: Confronto del tempo medio di esecuzione.

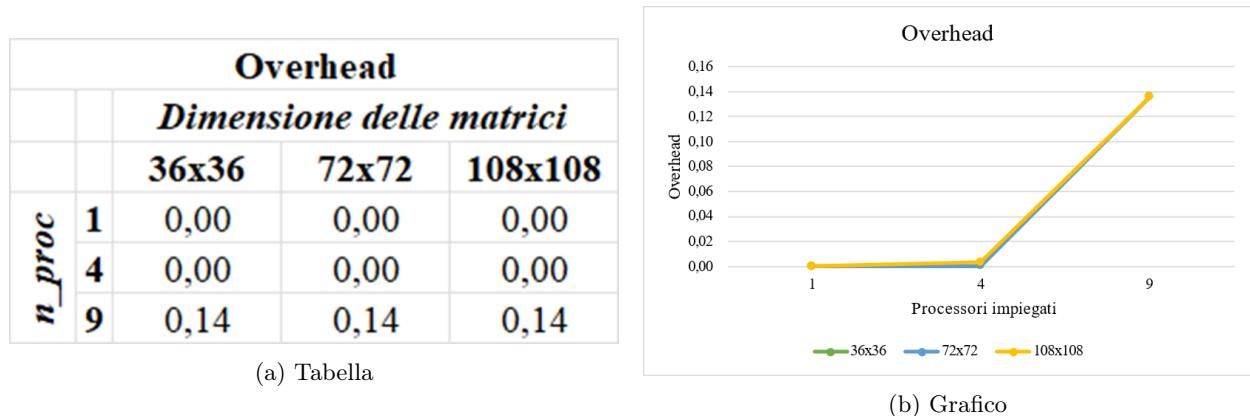
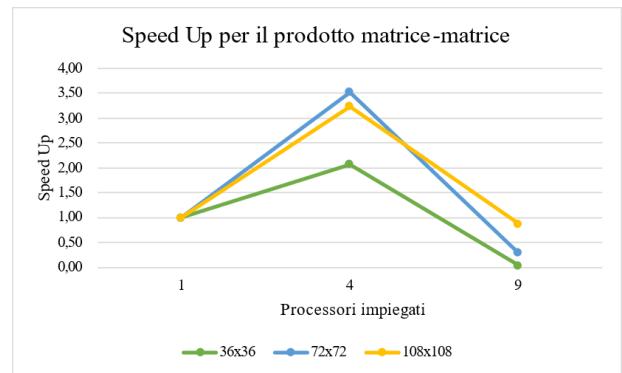


Figura 5.2: Confronto dei valori di overhead totale.

Speed Up			
Dimensione delle matrici			
	36x36	72x72	108x108
1	1,00	1,00	1,00
4	2,07	3,53	3,24
9	0,04	0,29	0,88

(a) Tabella

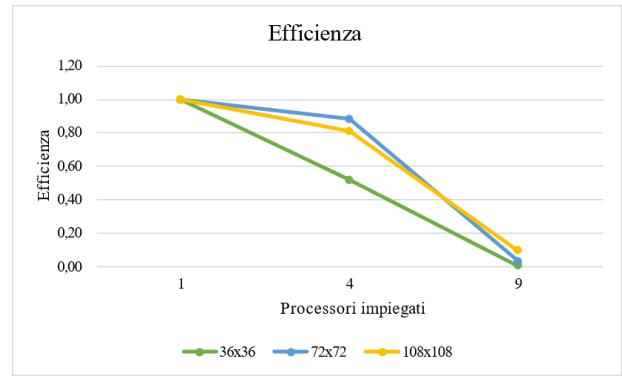


(b) Grafico

Figura 5.3: Confronto dei valori di speed up.

Efficienza			
Dimensione delle matrici			
	36x36	72x72	108x108
1	1,00	1,00	1,00
4	0,52	0,88	0,81
9	0,00	0,03	0,10

(a) Tabella



(b) Grafico

Figura 5.4: Confronto dei valori di efficienza.

I risultati emersi durante le fasi di test hanno chiaramente indicato che la parallelizzazione dell'algoritmo produce un miglioramento nel tempo medio di esecuzione, soprattutto per il caso di 4 processi in esecuzione dove si raggiunge un picco dell'88% di efficienza.

Tuttavia, è cruciale notare che per gli esperimenti analizzati si può osservare un peggioramento del tempo, dello speed up e dell'efficienza quando si esegue il programma su 9 processi.

Questo peggioramento è attribuibile all'architettura del cluster SCoPE, che comprende solo 8 nodi fisici. Infatti, per il calcolo dei tempi di esecuzione, si riserva un solo processore per ogni nodo fisico avendo, di conseguenza, un totale di 8 processori per l'esecuzione. Il 9° processo, quindi, è avviato su un processore che già ne ha avviato uno, generando un totale di due processi in concorrenza su quel nodo.

Quest'ultima configurazione, allora, compromette i risultati ottenuti e preclude la possibilità di condurre un'analisi esaustiva sulla scalabilità e sulla validità dell'algoritmo sviluppato.

Capitolo 6

Conclusioni

In conclusione, il presente progetto rappresenta un'importante esplorazione nell'implementazione di un algoritmo per il calcolo del prodotto matrice-matrice in ambiente di calcolo parallelo su architettura MIMD-DM.

- Nel capitolo 1 è stato delineato il problema affrontato descrivendo una possibile implementazione in pseudo-codice del prodotto matrice-matrice, e descrivendo la strategia di comunicazione BMR tra i processi.
- Nel capitolo 2 è stata fornita una descrizione dei codici di errori utili a rappresentare le cause di terminazione del programma, delle costanti utili per la definizione di alcuni parametri nel codice e delle funzioni adoperate dalle librerie di linguaggio disponibili e/o progettate dal team di sviluppo.
- Nel capitolo 3 è stato esaminato il processo di creazione del file *.pbs* necessario per l'esecuzione dell'algoritmo sul cluster, attraverso un apposito menù di creazione.
- Nel capitolo 4 è stata presentata l'implementazione dell'algoritmo vero e proprio, il cuore del nostro progetto, specificandone nel dettaglio le varie fasi dell'esecuzione.
- Nel capitolo 5, infine, è stata condotta un'analisi dettagliata delle prestazioni a seguito dell'esecuzione del programma sul cluster. I risultati ottenuti sono stati fondamentali per dimostrarne l'efficacia e la scalabilità in ambiente di calcolo parallelo.

La documentazione termina con l'appendice A che mostra tutto il codice sorgente del progetto.

6.1 Futuri sviluppi

In particolare, si è pensato anche ai possibili miglioramenti che si possono implementare nel programma.

- Primo tra tutti, la capacità del programma di salvare / stampare i tempi di esecuzione solo nel caso in cui il caso di test eseguito restituisca il risultato desiderato (al momento, è compito dell'utente e/o dello sviluppatore verificare la correttezza dei dati).
- Un altro possibile miglioramento potrebbe essere quello di rimettere insieme in unica matrice sul processo 0 (root) il risultato finale dell'esecuzione, utilizzando funzionalità messe a disposizione dalla libreria *mpi.h* come *MPI_Allgather()*, *MPI_Alltoall()* e/o *MPI_Gather()*.

Appendice A

Elenco dei listati

A.1 constants.c

```
1 /*  
2  
3     constants.c  
4     di Mario Gabriele Carofano  
5     e Francesco Noviello  
6  
7 */
```

Listing A.1: Intestazione del file *constants.c*

```
1 //  CODICI DI ERRORE  
2 #define NOT_ENOUGH_ARGS_ERROR 1  
3 #define EMPTY_ARG_ERROR 2  
4 #define INPUT_ARG_ERROR 3  
5 #define NOT_INT_ARG_ERROR 4  
6 #define INPUT_LINE_ERROR 5  
7 #define NOT_REAL_NUMBER_ERROR 6  
8 #define NOT_NATURAL_NUMBER_ERROR 7  
9 #define NOT_IN_RANGE_ERROR 8  
10 #define FILE_OPENING_ERROR 9  
11 #define FILE_CLOSING_ERROR 10  
12 #define MATRIX_DIMENSION_ERROR 11  
13 #define PROCESSOR_QUANTITY_ERROR 12  
14 #define ALLOCATION_ERROR 13
```

Listing A.2: Codici di errore

```
1 //  ENUMERAZIONI E COSTANTI  
2 #define DEBUG 1  
3 #define DEFAULT_SCELTA 0  
4 #define NO_TEST 1  
5 #define TESTING_SUITE 2  
6 #define EXIT_APPLICATION 3  
7 #define DEFAULT_TEST 0  
8 #define MULTIPLICATION_IDENTITY_TEST 1  
9 #define MULTIPLICATION_TRANSPOSE_TEST 2  
10 #define MULTIPLICATION_TRACE_TEST 3  
11 #define EXIT_TEST 4  
12 #define DEFAULT_INPUT 0  
13 #define VALUES_FROM_INPUT 1
```

```

14 #define VALUES_FROM_CSV 2
15 #define TAB_SPACE_PRINT 1
16 #define WOLFRAM_PRINT 2
17 #define OP_MAX_QUANTITY 32
18 #define MIN_COEFF_TEST 1
19 #define MAX_COEFF_TEST 3
20 #define NO_TIME_CALC 0
21 #define OK_TIME_CALC 1
22 #define NOME_PROVA "prova3"
23 #define NODE_NUMBER "8"
24 #define NODE_PROCESS "8"
25 #define MKDIR_PATH "/bin/mkdir"
26 #define RM_PATH "/bin/rm"
27 #define COMPILER_PATH "/usr/lib64/openmpi/1.4-gcc/bin/mpicc"
28 #define EXECUTE_PATH "/usr/lib64/openmpi/1.4-gcc/bin/mpiexec"
29 #define PBS_OUTPUT "$PBS_O_HOME/" NOME_PROVA "/output"
30 #define PBS_WORKDIR "$PBS_O_HOME/" NOME_PROVA "/codice"
31 #define PBS_EXECUTABLE "$PBS_O_HOME/" NOME_PROVA "/bin"
32 #define CSV_TIME_PATH "" NOME_PROVA "/output"
33 #define OP_MAX_VALUE 10
34 #define ARGS_QUANTITY 8
35 #define CSV_FIELD_PRECISION 10
36 #define CSV_FIELDS_SEPARATOR 44
37 #define CSV_ROWS_SEPARATOR 10
38 #define PATH_MAX_LENGTH 255
39 #define BROADCAST_TAG 101
40 #define ROLLING_TAG 103

```

Listing A.3: Enumerazioni e costanti

A.2 auxfunc.c

```

1 /*
2  auxfunc.c
3  di Mario Gabriele Carofano
4  e Francesco Noviello
5
6 */
7 // LIBRERIE
8
9
10 #include "constants.c"
11 #include <stdlib.h>
12 #include <errno.h>
13 #include <limits.h>
14 #include <string.h>
15 #include <math.h>
16 #include <time.h>

```

Listing A.4: Intestazione del file *auxfunc.c*

```

1 int argToInt(char *arg) {
2
3     char *p;
4     long out_long = 0;
5     errno = 0;
6     if (strlen(arg) == 0) {
7         printf("Errore nella lettura degli argomenti di input!\n\n");
8         printf("Esecuzione terminata.\n");
9         MPI_Finalize();
10        exit(EMPTY_ARG_ERROR);
11    }

```

```

12     out_long = strtol(arg, &p, 10);
13     if (*p != '\0' || errno != 0) {
14         printf("Errore nella lettura degli argomenti di input!\n\n");
15         printf("Esecuzione terminata.\n");
16         MPI_Finalize();
17         exit(INPUT_ARG_ERROR);
18     }
19     if (out_long < INT_MIN || out_long > INT_MAX) {
20         printf("Errore nella lettura degli argomenti di input!\n\n");
21         printf("Esecuzione terminata.\n");
22         MPI_Finalize();
23         exit(NOT_INT_ARG_ERROR);
24     }
25
26     return (int)out_long;
27 }
```

Listing A.5: La funzione *int argToInt(char*)*

```

1 double argToDouble(char *arg) {
2
3     char *p;
4     double out_double = 0.0;
5
6     errno = 0;
7
8     if (strlen(arg) == 0) {
9         printf("Errore nella lettura degli argomenti di input!\n\n");
10        printf("Esecuzione terminata.\n");
11        MPI_Finalize();
12        exit(EMPTY_ARG_ERROR);
13    }
14
15    out_double = strtod(arg, &p);
16
17    if (*p != '\0' || errno != 0) {
18        printf("Errore nella lettura degli argomenti di input!\n\n");
19        printf("Esecuzione terminata.\n");
20        MPI_Finalize();
21        exit(INPUT_ARG_ERROR);
22    }
23
24    return out_double;
25 }
```

Listing A.6: La funzione *double argToDouble(char*)*

```

1 void getRandomMatrix(double *mat, int mat_rows, int mat_cols) {
2
3     int i = 0, j = 0;
4     double double_rand = 0.0;
5     int int_rand = 0;
6
7     for (i = 0; i < mat_rows; i++) {
8         for (j = 0; j < mat_cols; j++) {
9             double_rand = (double)rand();
10            int_rand = (int)rand();
11            mat[i*mat_cols + j] = (double_rand / RAND_MAX) * OP_MAX_VALUE;
12            if (int_rand % 3 == 0) {
13                mat[i*mat_cols + j] *= -1;
14            }
15        }
16    }
17 }
18 }
```

Listing A.7: La funzione *void getRandomMatrix(double, int, int)*

```

1 void getIdentityMatrix(double *mat, int mat_rows, int mat_cols) {
2
3     int i = 0, j = 0;
4
5     for (i = 0; i < mat_rows; i++) {
6         for (j = 0; j < mat_cols; j++) {
7             if (i == j) {
8                 mat[i*mat_cols + j] = 1;
9             }
10        }
11    }
12 }
13 }
```

Listing A.8: La funzione *void getIdentityMatrix(double*, int, int)*

```

1 void getTransposeMatrix(double *A_mat, int A_rows, int A_cols, double *B_mat, int
2 B_rows, int B_cols) {
3
4     int i = 0, j = 0;
5
6     if (A_rows != B_cols || A_cols != B_rows) {
7         printf("Errore nella costruzione della matrice trasposta!\n\n");
8         printf("Esecuzione terminata!\n");
9         MPI_Finalize();
10        exit(MATRIX_DIMENSION_ERROR);
11    }
12
13    for (j = 0; j < A_cols; j++) {
14        for (i = 0; i < A_rows; i++) {
15            B_mat[j*B_cols + i] = A_mat[i*A_cols + j];
16        }
17    }
18 }
```

Listing A.9: La funzione *void getTransposeMatrix(double*, int, int, double, int, int)*

```

1 double* scatterMatrixToGrid(
2     double* mat, int mat_rows, int mat_cols,
3     int* loc_rows, int* loc_cols,
4     int id_grid, int* grid_dim, int* grid_coords, MPI_Comm comm_grid
5 ) {
6
7     int i = 0, j = 0, k = 0;
8     int rest_rows = 0, rest_cols = 0;
9
10    double* ret = NULL;
11
12    int *send_counts = NULL, *displs = NULL;
13    MPI_Datatype type_block;
14    *loc_rows = mat_rows / grid_dim[0];
15    *loc_cols = mat_cols / grid_dim[1];
16    rest_rows = mat_rows % grid_dim[0];
17    if (grid_coords[0] < rest_rows) {
18        *loc_rows++;
19    }
20
21    rest_cols = mat_cols % grid_dim[1];
22    if (grid_coords[1] < rest_cols) {
23        *loc_cols++;
24    }
25    ret = (double*) calloc(*loc_rows * *loc_cols, sizeof(double));
26
27    if (!ret) {
28        printf("La matrice locale non e' stata allocata correttamente!\n");
29        printf("Esecuzione terminata.\n");
30        MPI_Abort(comm_grid, ALLOCATION_ERROR);
31    }
32
33    MPI_Type_vector(*loc_rows, *loc_cols, mat_cols, MPI_DOUBLE, &type_block);
34    MPI_Type_create_resized(type_block, 0, *loc_rows * sizeof(double), &type_block
35    );
36    MPI_Type_commit(&type_block);
37    send_counts = (int*) calloc(grid_dim[0] * grid_dim[1], sizeof(int));
38    displs = (int*) calloc(grid_dim[0] * grid_dim[1], sizeof(int));
39    for (i = 0; i < grid_dim[0]; i++) {
40        for (j = 0; j < grid_dim[1]; j++) {
41            send_counts[i*grid_dim[1] + j] = 1;
42            displs[i*grid_dim[1] + j] = ((i * mat_cols * *loc_rows) + (j * *
43                loc_cols)) / *loc_cols;
44        }
45    }
46    MPI_Scatterv(
47        mat, send_counts, displs, type_block,
48        ret, *loc_rows * *loc_cols,
49        MPI_DOUBLE, 0, comm_grid
50    );
51    free(send_counts);
52    free(displs);
53
54    return ret;
55}

```

Listing A.10: La funzione `double* scatterMatrixToGrid(double*, int, int, int*, int*, int, int*, int*, MPI_Comm)`

```

1 void fprintfMatrix(FILE* out_file, double* mat, int rows, int cols, const char*
2 format) {
3
4     int i = 0, j = 0;
5     int mode = TAB_SPACE_PRINT;
6
7     switch(mode) {
8         case TAB_SPACE_PRINT:
9         {
10             for (i = 0; i < rows; i++) {
11                 for (j = 0; j < cols; j++) {
12                     fprintf(out_file, format, mat[i*cols + j]);
13                     if (j != cols-1) {
14                         fprintf(out_file, "\t");
15                     }
16                     if (i != rows-1) {
17                         fprintf(out_file, "\n");
18                     }
19                 }
20                 fprintf(out_file, "\n");
21             }
22             break;
23         case WOLFRAM_PRINT:
24         {
25             fprintf(out_file, "{");
26             for (i = 0; i < rows; i++) {
27                 fprintf(out_file, "{");
28                 for (j = 0; j < cols; j++) {
29                     fprintf(out_file, format, mat[i*cols + j]);
30                     if (j != cols-1) {
31                         fprintf(out_file, ", ");
32                     }
33                 }
34                 fprintf(out_file, "}");
35                 if (i != rows-1) {
36                     fprintf(out_file, ", ");
37                 }
38             }
39             fprintf(out_file, "}");
40             fprintf(out_file, "\n");
41             break;
42         }
43         default:
44             break;
45     }
46 }
47 }
```

Listing A.11: La funzione *void fprintfMatrix(FILE*, double*, int, int, const char*)*

```

1 void bmr_broadcast(
2     double* mat, double* tmp,
3     int rows, int cols,
4     int step,
5     MPI_Comm comm_grid, MPI_Comm comm_row
6 ) {
7
8     int grid_rank = 0, grid_coord[2] = {};
9     int row_size = 0, row_rank = 0;
10    int dest = 0;
11    int k = 0;
12 }
```

```

13 MPI_Status status;
14
15 MPI_Comm_rank(comm_grid, &grid_rank);
16 MPI_Cart_coords(comm_grid, grid_rank, 2, grid_coord);
17 MPI_Comm_size(comm_row, &row_size);
18 MPI_Comm_rank(comm_row, &row_rank);
19
20 if ((grid_coord[0]+step) % row_size == grid_coord[1]) {
21     for (k = 0; k < row_size; k++) {
22         dest = (row_rank + k + 1) % row_size;
23         if (dest == row_rank) {
24             memcpy(tmp, mat, rows * cols * sizeof(double));
25         } else {
26             MPI_Send(&(*mat), rows * cols, MPI_DOUBLE, dest, step *
27                     BROADCAST_TAG, comm_row);
28         }
29     }
30 } else {
31     MPI_Recv(&(*tmp), rows * cols, MPI_DOUBLE, MPI_ANY_SOURCE, step *
32             BROADCAST_TAG, comm_row, &status);
33 }

```

Listing A.12: La funzione `void bmr_broadcast(double*, double*, int, int, int, MPI_Comm, MPI_Comm)`

```

1 void bmr_multiply(
2     double* A_mat, double* B_mat, double* C_mat,
3     int A_rows, int A_cols,
4     int B_rows, int B_cols,
5     int C_rows, int C_cols,
6     MPI_Comm comm
7 ) {
8
9     int i = 0, j = 0, k = 0;
10    if (B_rows != A_cols) {
11        printf("Le matrici A,B non sono compatibili!\n");
12        printf("Esecuzione terminata.\n");
13        MPI_Abort(comm, MATRIX_DIMENSION_ERROR);
14    }
15
16    if ((C_rows != A_rows) || (C_cols != B_cols)) {
17        printf("Le matrici A,C non sono compatibili!\n");
18        printf("Esecuzione terminata.\n");
19        MPI_Abort(comm, MATRIX_DIMENSION_ERROR);
20    }
21
22    for (i = 0; i < A_rows; i++) {
23        for (j = 0; j < B_rows; j++) {
24            for (k = 0; k < C_rows; k++) {
25                C_mat[i*C_cols + j] += (A_mat[i*A_cols + k] * B_mat[k*B_cols + j])
26                ;
27            }
28        }
29    }
30 }

```

Listing A.13: La funzione `void bmr_multiply(double*, double*, double*, int, int, int, int, int, int, MPI_Comm)`

```

1 void bmr_rolling(
2     double* mat, double* tmp,
3     int rows, int cols,
4     int step,
5     MPI_Comm comm_col
6 ) {
7     int col_size = 0, col_rank = 0;
8     int dest = 0, source = 0;
9     MPI_Status status;
10    MPI_Comm_size(comm_col, &col_size);
11    MPI_Comm_rank(comm_col, &col_rank);
12    source = (col_rank + step) % col_size;
13    dest = (col_rank - step) % col_size;
14    if (dest < 0) {
15        dest = (dest + col_size) % col_size;
16        MPI_Recv(&(*tmp), rows * cols, MPI_DOUBLE, source, ROLLING_TAG, comm_col,
17                  &status);
18        MPI_Send(&(*mat), rows * cols, MPI_DOUBLE, dest, ROLLING_TAG, comm_col);
19    } else {
20        MPI_Send(&(*mat), rows * cols, MPI_DOUBLE, dest, ROLLING_TAG, comm_col);
21        MPI_Recv(&(*tmp), rows * cols, MPI_DOUBLE, source, ROLLING_TAG, comm_col,
22                  &status);
23    }
24 }

```

Listing A.14: La funzione `void bmr_rolling(double*, double*, int, int, int, MPI_Comm)`

```

1 void bmr(
2     double* A_mat, double* B_mat, double* C_mat,
3     int A_rows, int A_cols,
4     int B_rows, int B_cols,
5     int C_rows, int C_cols,
6     MPI_Comm comm, MPI_Comm comm_row, MPI_Comm comm_col
7 ) {
8
9     int n_proc = 0, id_proc = 0, col_size = 0;
10    int step = 0;
11
12    double *tmp_A_mat = NULL, *tmp_B_mat = NULL;
13
14    MPI_Comm_size(comm, &n_proc);
15    MPI_Comm_rank(comm, &id_proc);
16
17    if (n_proc == 1) {
18
19        bmr_multiply(
20            A_mat, B_mat, C_mat,
21            A_rows, A_cols,
22            B_rows, B_cols,
23            C_rows, C_cols,
24            comm
25        );
26
27    } else {
28
29        MPI_Comm_size(comm_col, &col_size);
30
31        tmp_A_mat = (double*) calloc(A_rows * A_cols, sizeof(double));
32        tmp_B_mat = (double*) calloc(B_rows * B_cols, sizeof(double));
33
34        if (!tmp_A_mat && !tmp_B_mat) {
35            printf("Le matrici locali non sono state allocate correttamente!\n");
36            printf("Esecuzione terminata.\n");

```

```

37         MPI_Abort(comm, ALLOCATION_ERROR);
38     }
39
40     for (step = 0; step < col_size; step++) {
41
42         if (id_proc == 0 && DEBUG) printf("\n--- PASSO %d ---\n\n", step+1);
43
44         bmr_broadcast(
45             A_mat, tmp_A_mat,
46             A_rows, A_cols,
47             step,
48             comm, comm_row
49         );
50
51         if (id_proc == 0 && DEBUG) printf("\tBROADCAST\n");
52
53         if (step == 0) {
54
55             bmr_multiply(
56                 tmp_A_mat, B_mat, C_mat,
57                 A_rows, A_cols,
58                 B_rows, B_cols,
59                 C_rows, C_cols,
60                 comm
61             );
62
63         } else {
64             bmr_rolling(
65                 B_mat, tmp_B_mat,
66                 B_rows, B_cols,
67                 step,
68                 comm_col
69             );
70
71             if (id_proc == 0 && DEBUG) printf("\tROLLING\n");
72
73             bmr_multiply(
74                 tmp_A_mat, tmp_B_mat, C_mat,
75                 A_rows, A_cols,
76                 B_rows, B_cols,
77                 C_rows, C_cols,
78                 comm
79             );
80         }
81
82         if (id_proc == 0 && DEBUG) printf("\tMULTIPLY\n");
83         if (id_proc == 0 && DEBUG) printf("\n-----\n\n");
84     }
85
86     free(tmp_A_mat);
87     free(tmp_B_mat);
88 }
89 }
```

Listing A.15: La funzione *void bmr(double*, double*, double*, int, int, int, int, int, int, MPI_Comm, MPI_Comm, MPI_Comm)*

A.3 csvfunc.c

```
1 /*
2
3     csvfunc.c
4     di Mario Gabriele Carofano
5     e Francesco Noviello
6
7 */
8 // LIBRERIE
9
10 #include "constants.c"
11 #include "auxfunc.h"
12 #include <stdlib.h>
```

Listing A.16: Intestazione del file *csvfunc.c*

```
1 void getDimensionsFromCSV(FILE* csv_file, int* size, int* rows_csv, int* cols_csv)
2 {
3     char c = 0;
4     int file_pointer = -1;
5
6     *size = 0;
7     *rows_csv = 0;
8     *cols_csv = 0;
9
10    fseek(csv_file, 0, SEEK_END);
11    *size = ftell(csv_file);
12    if (*size == 0) {
13        *rows_csv = 0;
14        *cols_csv = 0;
15        return;
16    }
17
18    fseek(csv_file, 0, SEEK_SET);
19
20    do {
21        file_pointer++;
22        c = fgetc(csv_file);
23
24        if (c == CSV_FIELDS_SEPARATOR) {
25            *cols_csv += 1;
26        }
27
28        if (c == CSV_ROWS_SEPARATOR) {
29            *cols_csv = 0;
30            *rows_csv += 1;
31        }
32    } while (file_pointer < *size);
33
34    *cols_csv += 1;
35    *rows_csv += 1;
36
37    fseek(csv_file, 0, SEEK_SET);
38 }
```

Listing A.17: La funzione *void getDimensionsFromCSV(FILE* csv_file, int* size, int* rows_csv, int* cols_csv)*

```

1 void getMatrixFromCSV(FILE* csv_file, double* mat, int rows_mat, int cols_mat,
2 MPI_Comm comm) {
3
4     char c, char_val[CSV_FIELD_PRECISION] = {};
5     int size = 0, rows_csv = 0, cols_csv = 0;
6
7     int i = -1, file_pointer = -1;
8     int row_count = 0, comma_count = 0;
9
10    getDimensionsFromCSV(csv_file, &size, &rows_csv, &cols_csv);
11    if ((rows_mat > rows_csv) || (cols_mat > cols_csv)) {
12        printf("La dimensione della matrice non e' corretta!\n");
13        printf("Applicazione terminata.\n");
14        MPI_Abort(comm, MATRIX_DIMENSION_ERROR);
15    }
16
17    do {
18        ++file_pointer;
19        ++i;
20        c = fgetc(csv_file);
21        if (c != CSV_FIELDS_SEPARATOR) {
22            if (i < CSV_FIELD_PRECISION-1) {
23                char_val[i] = c;
24            }
25        }
26        if (file_pointer == size || c == CSV_FIELDS_SEPARATOR || c ==
27            CSV_ROWS_SEPARATOR) {
28            char_val[i] = '\0';
29            mat[row_count*cols_mat + comma_count] = argToDouble(char_val);
30            comma_count++;
31
32            if (c == CSV_ROWS_SEPARATOR || comma_count == cols_mat) {
33                comma_count = 0;
34                row_count++;
35                if (row_count == rows_mat) {
36                    break;
37                }
38            }
39            for (i = 0; i < CSV_FIELD_PRECISION; i++) {
40                char_val[i] = '0';
41            }
42            char_val[i] = '\0';
43
44            i = -1;
45        }
46    } while (file_pointer < size);
47
48 }

```

Listing A.18: La funzione `void getMatrixFromCSV(FILE* csv_file, double* mat, int rows_mat, int cols_mat, MPI_Comm comm)`

```

1 void writeTimeCSV(
2     const char* csv_path,
3     int A_rows, int A_cols,
4     int B_rows, int B_cols,
5     int n_proc, int input, int test,
6     double time_tot,
7     MPI_Comm comm
8 ) {
9
10    FILE *csv_file;
11    int size = 0;
12
13    system(MKDIR_PATH" -p "CSV_TIME_PATH);
14
15    if ((csv_file = fopen(csv_path, "a")) == NULL) {
16        printf("Nessun file o directory con questo nome: %s\n", csv_path);
17        printf("Esecuzione terminata.\n");
18        MPI_Abort(comm, FILE_OPENING_ERROR);
19    }
20    fseek(csv_file, 0, SEEK_END);
21    size = ftell(csv_file);
22    if (size != 0) {
23        fprintf(csv_file, "\n");
24    }
25
26    fseek(csv_file, 0, SEEK_SET);
27    fprintf(csv_file, "%d,%d,%d,%d,%d,%d,%d,%1.9f",
28            A_rows, A_cols, B_rows, B_cols, n_proc, input, test, time_tot
29    );
30
31    if (fclose(csv_file) != 0) {
32        printf("Nessun file o directory con questo nome: %s\n", csv_path);
33        printf("Esecuzione terminata.\n");
34        MPI_Abort(comm, FILE_CLOSING_ERROR);
35    }
36
37    printf("%s aggiornato con successo!\n\n", csv_path);
38
39 }

```

Listing A.19: La funzione `void writeTimeCSV(const char* csv_path, int A_rows, int A_cols, int B_rows, int B_cols, int n_proc, int input, int test, double time_tot, MPI_Comm comm)`

A.4 menufunc.c

```

1 /*
2
3     menufunc.c
4     di Mario Gabriele Carofano
5     e Francesco Noviello
6
7 */
8 // LIBRERIE
9
10 #include "constants.c"
11
12 #include <stdlib.h>
13 #include <ctype.h>
14 #include <math.h>

```

Listing A.20: Intestazione del file `menufunc.c`

Si omette il codice di questa funzione in quanto contiene solo le stampe del titolo dell'elaborato.

Listing A.21: La funzione *void printTitle()*

```
1 double getNumberFromInput() {
2     double out = 0;
3     char *buffer = NULL;
4     size_t bufsize = 0;
5     ssize_t chars_read;
6     chars_read = getline(&buffer, &bufsize, stdin);
7     printf("\n");
8
9     if (chars_read < 0) {
10
11         printf("Errore nella lettura dell'input!");
12         printf("Applicazione terminata.\n");
13         free(buffer);
14         exit(INPUT_LINE_ERROR);
15
16     } else {
17
18         double digit_val = 0.0;
19         int i = 0;
20         int exp_whole = 0;
21         int exp_fract = 0;
22         int pos_point = -1;
23         for (i = chars_read-2; i >= 0; i--) {
24             if (buffer[i] != '.' && buffer[i] != ',') {
25                 exp_fract--;
26             } else {
27                 pos_point = chars_read - (exp_fract * (-1)) - 2;
28                 break;
29             }
30         }
31
32         i = chars_read-2;
33         while(i >= 0) {
34             if (i > pos_point && pos_point != -1) {
35
36                 if (isdigit(buffer[i])) {
37                     digit_val = pow(10, exp_fract) * (digit_val + (buffer[i] - '0'));
38                     out = out + digit_val;
39                 } else {
40                     out = 0.0;
41                     printf("Puoi inserire solo valori numerici reali!\n");
42                     printf("Applicazione terminata.\n");
43                     free(buffer);
44                     exit(NOT_REAL_NUMBER_ERROR);
45                 }
46
47                 exp_fract++;
48             } else if (i < pos_point || pos_point == -1) {
49
50                 if (isdigit(buffer[i])) {
51                     digit_val = pow(10, exp_whole) * (digit_val + (buffer[i] - '0'));
52                     out = out + digit_val;
53                 } else {
54                     if (i == 0 && buffer[0] == '_') {
55                         out = out * (-1);
56                     }
57                 }
58             }
59         }
60     }
61
62     free(buffer);
63     exit(0);
64 }
```

```

57             } else {
58                 out = 0.0;
59                 printf("Puoi inserire solo valori numerici reali!\n");
60                 printf("Applicazione terminata.\n");
61                 free(buffer);
62                 exit(NOT_REAL_NUMBER_ERROR);
63             }
64         }
65
66         exp_whole++;
67     }
68
69     digit_val = 0.0;
70     i--;
71 }
72
73 free(buffer);
74 return out;
75 }
76 }
```

Listing A.22: La funzione *double getNumberFromInput()*

```

1 int getIntegerFromInput() {
2
3     double out_double = 0.0;
4     int out_integer = 0;
5
6     out_double = getNumberFromInput();
7     out_integer = (int)out_double;
8
9     if (out_integer < 0) {
10        printf("Puoi inserire solo valori numerici naturali!\n");
11        printf("Applicazione terminata.\n");
12        exit(NOT_NATURAL_NUMBER_ERROR);
13    }
14
15    return out_integer;
16 }
```

Listing A.23: La funzione *int getIntegerFromInput()*

```

1 void checkScelta(int scelta, int lim_inf, int lim_sup) {
2     if(!(scelta >= lim_inf && scelta <= lim_sup)) {
3         printf("Puoi inserire solo un valore numerico intero compreso tra %d e %d
4             !\n", lim_inf, lim_sup);
5         printf("Applicazione terminata.\n");
6         exit(NOT_IN_RANGE_ERROR);
7     }
7 }
```

Listing A.24: La funzione *void checkScelta(int scelta, int lim_inf, int lim_sup)*

```

1 void createPBS(
2     int A_rows, int A_cols, int B_rows, int B_cols,
3     int n_proc, int input, int test, int time_calc,
4     int pbs_count
5 ) {
6
7     char pbs_path[PATH_MAX_LENGTH] = {};
8     FILE *pbs_file;
9
10    int i = 0, j = 0;
```

```

11 int A_num = 0, B_num = 0;
12 int int_op = 0;
13 double double_op = 0.0;
14
15 char *buffer = NULL;
16 char A_mat_csv[PATH_MAX_LENGTH] = {};
17 char B_mat_csv[PATH_MAX_LENGTH] = {};
18 size_t bufsize = 0;
19 ssize_t chars_read;
20
21 if (pbs_count <= 1) {
22     system(RM_PATH " -rf " NOME_PROVA "/jobs");
23     system(MKDIR_PATH " -p " NOME_PROVA "/jobs");
24     system(MKDIR_PATH " -p " NOME_PROVA "/bin");
25 }
26
27 sprintf(pbs_path, NOME_PROVA "/jobs/" NOME_PROVA "_%03d.pbs", pbs_count);
28
29 if ((pbs_file = fopen(pbs_path, "a")) == NULL) {
30     printf("Nessun file o directory con questo nome: %s\n", pbs_path);
31     printf("Applicazione terminata.\n");
32     exit(FILE_OPENING_ERROR);
33 }
34
35 fprintf(pbs_file,
36         "#!/bin/bash\n"
37         "\n"
38         "#PBS -q studenti\n"
39         "#PBS -l nodes=" NODE_NUMBER
40 );
41
42 if (time_calc == OK_TIME_CALC) {
43     fprintf(pbs_file, ":ppn=" NODE_PROCESS);
44 }
45
46 fprintf(pbs_file,
47         "\n"
48         "#PBS -N " NOME_PROVA "_%03d\n"
49         "#PBS -o ../output/" NOME_PROVA "_%03d/std.out\n"
50         "#PBS -e ../output/" NOME_PROVA "_%03d/std.err\n",
51 pbs_count, pbs_count, pbs_count);
52
53 fprintf(pbs_file,
54         "\n"
55         "PBS_O_OUTPUT=$PBS_O_HOME/" NOME_PROVA "/output/" NOME_PROVA "_%03
56         d\n"
57         "PBS_O_WORKDIR=" PBS_WORKDIR "\n"
58         "PBS_O_EXECUTABLE=" PBS_EXECUTABLE "\n"
59         "\n",
60 pbs_count);
61
62 if (pbs_count <= 1) {
63     fprintf(pbs_file,
64             RM_PATH " -fr $PBS_O_EXECUTABLE\n"
65             MKDIR_PATH " -p $PBS_O_EXECUTABLE\n"
66             "\n");
67 }
68
69 fprintf(pbs_file,
70         RM_PATH " -fr $PBS_O_OUTPUT\n"
71         MKDIR_PATH " -p $PBS_O_OUTPUT\n"
72         "\n");

```

```

73
74     fprintf(pbs_file, "echo --- \n");
75
76     if (time_calc == OK_TIME_CALC) {
77         fprintf(pbs_file, "sort -u $PBS_NODEFILE > hostlist\n");
78     }
79
80     fprintf(pbs_file,
81             "\n"
82             "echo PBS: la directory degli output e\\\' $PBS_O_OUTPUT\n"
83             "echo PBS: la directory di lavoro e\\\' $PBS_O_WORKDIR\n"
84             "echo PBS: la directory degli eseguibili e\\\' $PBS_O_EXECUTABLE\n"
85             "echo PBS: Compilazione in esecuzione...\n"
86             COMPILER_PATH " -o $PBS_O_EXECUTABLE/" NOME_PROVA "_%03d -lm
87             $PBS_O_WORKDIR/" NOME_PROVA ".c\n"
88             "echo PBS: Compilazione completata.\n"
89             "\n"
90             "echo 'PBS: Job in esecuzione su %d cpu...'\n"
91             "echo '>>>'\n",
92             pbs_count, n_proc);
93
94     if (time_calc == OK_TIME_CALC) {
95         fprintf(pbs_file,
96                 "for i in $(seq 10)\n"
97                 "do\n"
98                 "\t" EXECUTE_PATH " -machinefile hostlist -np %d ",
99                 n_proc);
100 } else {
101     fprintf(pbs_file, EXECUTE_PATH " -machinefile $PBS_NODEFILE -n %d ",
102             n_proc);
103 }
104
105     fprintf(pbs_file, "$PBS_O_EXECUTABLE/" NOME_PROVA "_%03d %d %d %d %d %d %d
106             %d",
107             pbs_count, A_rows, A_cols, B_rows, B_cols, input, test, time_calc,
108             pbs_count);
109     A_num = A_rows * A_cols;
110     B_num = B_rows * B_cols;
111
112     switch(test) {
113         case DEFAULT_TEST: {
114             if (input == VALUES_FROM_INPUT) {
115                 if ((A_num+B_num) <= OP_MAX_QUANTITY) {
116
117                     printf("Inserimento di %d elementi nella matrice A.\n", A_num)
118                     ;
119                     for (i = 1; i <= A_rows; i++) {
120                         for (j = 1; j <= A_cols; j++) {
121                             printf("Inserisci il %do elemento sulla %da riga:\n",
122                                 j, i);
123                             double_op = getNumberFromInput();
124                             fprintf(pbs_file, " %f", double_op);
125                         }
126                     }
127
128                     printf("Inserimento di %d elementi nella matrice B.\n", B_num)
129                     ;
130                     for (i = 1; i <= B_rows; i++) {
131                         for (j = 1; j <= B_cols; j++) {
132                             printf("Inserisci il %do elemento sulla %da riga:\n",
133                                 j, i);
134                             double_op = getNumberFromInput();
135                             fprintf(pbs_file, " %f", double_op);
136                         }
137                     }
138
139                     if (input == VALUES_FROM_FILE) {
140                         if ((A_num+B_num) <= OP_MAX_QUANTITY) {
141
142                             printf("Inserimento di %d elementi nella matrice A.\n", A_num)
143                             ;
144                             for (i = 1; i <= A_rows; i++) {
145                                 for (j = 1; j <= A_cols; j++) {
146                                     printf("Inserisci il %do elemento sulla %da riga:\n",
147                                         j, i);
148                                     double_op = getNumberFromInput();
149                                     fprintf(pbs_file, " %f", double_op);
150                                 }
151                             }
152
153                             printf("Inserimento di %d elementi nella matrice B.\n", B_num)
154                             ;
155                             for (i = 1; i <= B_rows; i++) {
156                                 for (j = 1; j <= B_cols; j++) {
157                                     printf("Inserisci il %do elemento sulla %da riga:\n",
158                                         j, i);
159                                     double_op = getNumberFromInput();
160                                     fprintf(pbs_file, " %f", double_op);
161                                 }
162                             }
163
164                             if (input == VALUES_FROM_FILE) {
165
166                                 printf("Inserimento di %d elementi nella matrice A.\n", A_num)
167                                 ;
168                                 for (i = 1; i <= A_rows; i++) {
169                                     for (j = 1; j <= A_cols; j++) {
170                                         printf("Inserisci il %do elemento sulla %da riga:\n",
171                                             j, i);
172                                         double_op = getNumberFromInput();
173                                         fprintf(pbs_file, " %f", double_op);
174                                     }
175                                 }
176
177                                 printf("Inserimento di %d elementi nella matrice B.\n", B_num)
178                                 ;
179                                 for (i = 1; i <= B_rows; i++) {
180                                     for (j = 1; j <= B_cols; j++) {
181                                         printf("Inserisci il %do elemento sulla %da riga:\n",
182                                             j, i);
183                                         double_op = getNumberFromInput();
184                                         fprintf(pbs_file, " %f", double_op);
185                                     }
186                                 }
187
188                                 if (input == VALUES_FROM_FILE) {
189
190                                     printf("Inserimento di %d elementi nella matrice A.\n", A_num)
191                                     ;
192                                     for (i = 1; i <= A_rows; i++) {
193                                         for (j = 1; j <= A_cols; j++) {
194                                             printf("Inserisci il %do elemento sulla %da riga:\n",
195                                                 j, i);
196                                             double_op = getNumberFromInput();
197                                             fprintf(pbs_file, " %f", double_op);
198                                         }
199                                     }
200
201                                     printf("Inserimento di %d elementi nella matrice B.\n", B_num)
202                                     ;
203                                     for (i = 1; i <= B_rows; i++) {
204                                         for (j = 1; j <= B_cols; j++) {
205                                             printf("Inserisci il %do elemento sulla %da riga:\n",
206                                                 j, i);
207                                             double_op = getNumberFromInput();
208                                             fprintf(pbs_file, " %f", double_op);
209                                         }
210                                     }
211
212                                     if (input == VALUES_FROM_FILE) {
213
214                                         printf("Inserimento di %d elementi nella matrice A.\n", A_num)
215                                         ;
216                                         for (i = 1; i <= A_rows; i++) {
217                                             for (j = 1; j <= A_cols; j++) {
218                                                 printf("Inserisci il %do elemento sulla %da riga:\n",
219                                                     j, i);
220                                                 double_op = getNumberFromInput();
221                                                 fprintf(pbs_file, " %f", double_op);
222                                             }
223                                         }
224
225                                         printf("Inserimento di %d elementi nella matrice B.\n", B_num)
226                                         ;
227                                         for (i = 1; i <= B_rows; i++) {
228                                             for (j = 1; j <= B_cols; j++) {
229                                                 printf("Inserisci il %do elemento sulla %da riga:\n",
230                                                     j, i);
231                                                 double_op = getNumberFromInput();
232                                                 fprintf(pbs_file, " %f", double_op);
233                                             }
234                                         }
235
236                                         if (input == VALUES_FROM_FILE) {
237
238                                             printf("Inserimento di %d elementi nella matrice A.\n", A_num)
239                                             ;
240                                             for (i = 1; i <= A_rows; i++) {
241                                                 for (j = 1; j <= A_cols; j++) {
242                                                     printf("Inserisci il %do elemento sulla %da riga:\n",
243                                                         j, i);
244                                                     double_op = getNumberFromInput();
245                                                     fprintf(pbs_file, " %f", double_op);
246                                                 }
247                                             }
248
249                                             printf("Inserimento di %d elementi nella matrice B.\n", B_num)
250                                             ;
251                                             for (i = 1; i <= B_rows; i++) {
252                                                 for (j = 1; j <= B_cols; j++) {
253                                                     printf("Inserisci il %do elemento sulla %da riga:\n",
254                                                         j, i);
255                                                     double_op = getNumberFromInput();
256                                                     fprintf(pbs_file, " %f", double_op);
257                                                 }
258                                             }
259
260                                             if (input == VALUES_FROM_FILE) {
261
262                                                 printf("Inserimento di %d elementi nella matrice A.\n", A_num)
263                                                 ;
264                                                 for (i = 1; i <= A_rows; i++) {
265                                                     for (j = 1; j <= A_cols; j++) {
266                                                         printf("Inserisci il %do elemento sulla %da riga:\n",
267                                                             j, i);
268                                                         double_op = getNumberFromInput();
269                                                         fprintf(pbs_file, " %f", double_op);
270                                                     }
271                                                 }
272
273                                                 printf("Inserimento di %d elementi nella matrice B.\n", B_num)
274                                                 ;
275                                                 for (i = 1; i <= B_rows; i++) {
276                                                     for (j = 1; j <= B_cols; j++) {
277                                                         printf("Inserisci il %do elemento sulla %da riga:\n",
278                                                             j, i);
279                                                         double_op = getNumberFromInput();
280                                                         fprintf(pbs_file, " %f", double_op);
281                                                     }
282                                                 }
283
284                                                 if (input == VALUES_FROM_FILE) {
285
286                                                     printf("Inserimento di %d elementi nella matrice A.\n", A_num)
287                                                     ;
288                                                     for (i = 1; i <= A_rows; i++) {
289                                                         for (j = 1; j <= A_cols; j++) {
290                                                             printf("Inserisci il %do elemento sulla %da riga:\n",
291                                                                 j, i);
292                                                             double_op = getNumberFromInput();
293                                                             fprintf(pbs_file, " %f", double_op);
294                                                         }
295                                                     }
296
297                                                     printf("Inserimento di %d elementi nella matrice B.\n", B_num)
298                                                     ;
299                                                     for (i = 1; i <= B_rows; i++) {
300                                                         for (j = 1; j <= B_cols; j++) {
301                                                             printf("Inserisci il %do elemento sulla %da riga:\n",
302                                                                 j, i);
303                                                             double_op = getNumberFromInput();
304                                                             fprintf(pbs_file, " %f", double_op);
305                                                         }
306                                                     }
307
308                                                     if (input == VALUES_FROM_FILE) {
309
310                                                         printf("Inserimento di %d elementi nella matrice A.\n", A_num)
311                                                         ;
312                                                         for (i = 1; i <= A_rows; i++) {
313                                                             for (j = 1; j <= A_cols; j++) {
314                                                                 printf("Inserisci il %do elemento sulla %da riga:\n",
315                                                                     j, i);
316                                                                 double_op = getNumberFromInput();
317                                                                 fprintf(pbs_file, " %f", double_op);
318                                                             }
319                                                         }
320
321                                                         printf("Inserimento di %d elementi nella matrice B.\n", B_num)
322                                                         ;
323                                                         for (i = 1; i <= B_rows; i++) {
324                                                             for (j = 1; j <= B_cols; j++) {
325                                                                 printf("Inserisci il %do elemento sulla %da riga:\n",
326                                                                     j, i);
327                                                                 double_op = getNumberFromInput();
328                                                                 fprintf(pbs_file, " %f", double_op);
329                                                             }
330                                                         }
331
332                                                         if (input == VALUES_FROM_FILE) {
333
334                                                             printf("Inserimento di %d elementi nella matrice A.\n", A_num)
335                                                             ;
336                                                             for (i = 1; i <= A_rows; i++) {
337                                                                 for (j = 1; j <= A_cols; j++) {
338                                                                     printf("Inserisci il %do elemento sulla %da riga:\n",
339                                                                         j, i);
340                                                                     double_op = getNumberFromInput();
341                                                                     fprintf(pbs_file, " %f", double_op);
342                                                                 }
343                                                             }
344
345                                                             printf("Inserimento di %d elementi nella matrice B.\n", B_num)
346                                                             ;
347                                                             for (i = 1; i <= B_rows; i++) {
348                                                                 for (j = 1; j <= B_cols; j++) {
349                                                                     printf("Inserisci il %do elemento sulla %da riga:\n",
350                                                                         j, i);
351                                                                     double_op = getNumberFromInput();
352                                                                     fprintf(pbs_file, " %f", double_op);
353                                                                 }
354                                                             }
355
356                                                             if (input == VALUES_FROM_FILE) {
357
358                                                                 printf("Inserimento di %d elementi nella matrice A.\n", A_num)
359                                                                 ;
360                                                                 for (i = 1; i <= A_rows; i++) {
361                                                                 for (j = 1; j <= A_cols; j++) {
362                                                                     printf("Inserisci il %do elemento sulla %da riga:\n",
363                                                                         j, i);
364                                                                     double_op = getNumberFromInput();
365                                                                     fprintf(pbs_file, " %f", double_op);
366                                                                 }
367
368                                                                 printf("Inserimento di %d elementi nella matrice B.\n", B_num)
369                                                                 ;
370                                                                 for (i = 1; i <= B_rows; i++) {
371                                                                 for (j = 1; j <= B_cols; j++) {
372                                                                     printf("Inserisci il %do elemento sulla %da riga:\n",
373                                                                         j, i);
374                                                                     double_op = getNumberFromInput();
375                                                                     fprintf(pbs_file, " %f", double_op);
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
589
590
591
592
593
594
595
596
597
598
599
599
600
601
602
603
604
605
606
607
608
609
609
610
611
612
613
614
615
616
617
618
619
619
620
621
622
623
624
625
626
627
628
629
629
630
631
632
633
634
635
636
637
638
639
639
640
641
642
643
644
645
646
647
648
649
649
650
651
652
653
654
655
656
657
658
659
659
660
661
662
663
664
665
666
667
668
669
669
670
671
672
673
674
675
676
677
678
679
679
680
681
682
683
684
685
686
687
687
688
689
689
690
691
692
693
694
695
696
697
697
698
699
699
700
701
702
703
704
705
706
707
708
709
709
710
711
712
713
714
715
716
717
717
718
719
719
720
721
722
723
724
725
726
727
727
728
729
729
730
731
732
733
734
735
736
737
737
738
739
739
740
741
742
743
744
745
746
747
747
748
749
749
750
751
752
753
754
755
756
757
757
758
759
759
760
761
762
763
764
765
766
767
767
768
769
769
770
771
772
773
774
775
776
777
777
778
779
779
780
781
782
783
784
785
786
787
787
788
789
789
790
791
792
793
794
795
796
797
797
798
799
799
800
801
802
803
804
805
806
807
807
808
809
809
810
811
812
813
814
815
816
816
817
817
818
819
819
820
821
822
823
824
825
826
826
827
827
828
829
829
830
831
832
833
834
835
836
837
837
838
839
839
840
841
842
843
844
845
846
846
847
847
848
849
849
850
851
852
853
854
855
856
856
857
857
858
859
859
860
861
862
863
864
865
866
866
867
867
868
869
869
870
871
872
873
874
875
876
876
877
877
878
879
879
880
881
882
883
884
885
886
886
887
887
888
889
889
890
891
892
893
894
895
895
896
896
897
897
898
899
899
900
901
902
903
904
905
906
906
907
907
908
909
909
910
911
912
913
914
914
915
915
916
916
917
917
918
918
919
919
920
921
921
922
922
923
923
924
924
925
925
926
926
927
927
928
928
929
929
930
930
931
931
932
932
933
933
934
934
935
935
936
936
937
937
938
938
939
939
940
940
941
941
942
942
943
943
944
944
945
945
946
946
947
947
948
948
949
949
950
950
951
951
952
952
953
953
954
954
955
955
956
956
957
957
958
958
959
959
960
960
961
961
962
962
963
963
964
964
965
965
966
966
967
967
968
968
969
969
970
970
971
971
972
972
973
973
974
974
975
975
976
976
977
977
978
978
979
979
980
980
981
981
982
982
983
983
984
984
985
985
986
986
987
987
988
988
989
989
990
990
991
991
992
992
993
993
994
994
995
995
996
996
997
997
998
998
999
999
1000
1000
1001
1001
1002
1002
1003
1003
1004
1004
1005
1005
1006
1006
1007
1007
1008
1008
1009
1009
1010
1010
1011
1011
1012
1012
1013
1013
1014
1014
1015
1015
1016
1016
1017
1017
1018
1018
1019
1019
1020
1020
1021
1021
1022
1022
1023
1023
1024
1024
1025
1025
1026
1026
1027
1027
1028
1028
1029
1029
1030
1030
1031
1031
1032
1032
1033
1033
1034
1034
1035
1035
1036
1036
1037
1037
1038
1038
1039
1039
1040
1040
1041
1041
1042
1042
1043
1043
1044
1044
1045
1045
1046
1046
1047
1047
1048
1048
1049
1049
1050
1050
1051
1051
1052
1052
1053
1053
1054
1054
1055
1055
1056
1056
1057
1057
1058
1058
1059
1059
1060
1060
1061
1061
1062
1062
1063
1063
1064
1064
1065
1065
1066
1066
1067
1067
1068
1068
1069
1069
1070
1070
1071
1071
1072
1072
1073
1073
1074
1074
1075
1075
1076
1076
1077
1077
1078
1078
1079
1079
1080
1080
1081
1081
1082
1082
1083
1083
1084
1084
1085
1085
1086
1086
1087
1087
1088
1088
1089
1089
1090
1090
1091
1091
1092
1092
1093
1093
1094
1094
1095
1095
1096
1096
1097
1097
1098
1098
1099
1099
1100
1100
1101
1101
1102
1102
1103
1103
1104
1104
1105
1105
1106
1106
1107
1107
1108
1108
1109
1109
1110
1110
1111
1111
1112
1112
1113
1113
1114
1114
1115
1115
1116
1116
1117
1117
1118
1118
1119
1119
1120
1120
1121
1121
1122
1122
1123
1123
1124
1124
1125
1125
1126
1126
1127
1127
1128
1128
1129
1129
1130
1130
1131
1131
1132
1132
1133
1133
1134
1134
1135
1135
1136
1136
1137
1137
1138
1138
1139
1139
1140
1140
1141
1141
1142
1142
1143
1143
1144
1144
1145
1145
1146
1146
1147
1147
1148
1148
1149
1149
1150
1150
1151
1151
1152
1152
1153
1153
1154
1154
1155
1155
1156
1156
1157
1157
1158
1158
1159
1159
1160
1160
1161
1161
1162
1162
1163
1163
1164
1164
1165
1165
1166
1166
1167
1167
1168
1168
1169
1169
1170
1170
1171
1171
1172
1172
1173
1173
1174
1174
1175
1175
1176
1176
1177
1177
1178
1178
1179
1179
1180
1180
1181
1181
1182
1182
1183
1183
1184
1184
1185
1185
1186
1186
1187
1187
1188
1188
1189
1189
1190
1190
1191
1191
1192
1192
1193
1193
1194
1194
1195
1195
1196
1196
1197
1197
1198
1198
1199
1199
1200
1200
1201
1201
1202
1202
1203
1203
1204
1204
1205
1205
1206
1206
1207
1207
1208
1208
1209
1209
1210
1210
1211
1211
1212
1212
1213
1213
1214
1214
1215
1215
1216
1216
1217
1217
1218
1218
1219
1219
1220
1220
1221
1221
1222
1222
1223
1223
1224
1224
1225
1225
1226
1226
1227
1227
1228
1228
1229
1229
1230
1230
1231
1231
1232
1232
1233
1233
1234
1234
1235
1235
1236
1236
1237
1237
1238
1238
1239
1239
1240
1240
1241
1241
1242
1242
1243
1243
1244
1244
1245
1245
1246
1246
1247
1247
1248
1248
1249
1249
1250
1250
1251
1251
1252
1252
1253
1253
1254
1254
1255
1255
1256
1256
1257
1257
1258
1258
1259
1259
1260
1260
1261
1261
1262
1262
1263
1263
1264
1264
1265
1265
1266
1266
1267
1267
1268
1268
1269
1269
1270
1270
1271
1271
1272
1272
1273
1273
1274
1274
1275
1275
1276
1276
1277
1277
1278
1278
1279
1279
1280
1280
1281
1281
1282
1282
1283
1283
1284
1284
1285
1285
1286
1286
1287
1287
1288
1288
1289
1289
1290
1290
1291
1291
1292
1292
1293
1293
1294
1294
1295
1295
1296
1296
1297
1297
1298
1298
1299
1299
1300
1300
1301
1301
1302
1302
1303
1303
1304
1304
1305
1305
1306
1306
1307
1307
1308
1308
1309
1309
1310
1310
1311
1311
1312
1312
1313
1313
1314
1314
1315
1315
1316
1316
1317
1317
1318
1318
1319
1319
1320
1320
1321
1321
1322
1322
1323
1323
1324
1324
1325
1325
1326
1326
1327
1327
1328
1328
1329
1329
1330
1330
1331
1331
1332
1332
1333
1333
1334
1334
1335
1335
1336
1336
1337
1337
1338
1338
1339
1339
1340
1340
1341
1341
1342
1342
1343
1343
1344
1344
1345
1345
1346
1346
1347
1347
1348
1348
1349
1349
1350
1350
1351
1351
1352
1352
1353
1353
1354
1354
1355
1355
1356
1356
1357
1357
1358
1358
1359
1359
1360
1360
1361
1361
1362
1362
1363
1363
1364
1364
1365
1365
1366
1366
1367
1367
1368
1368
1369
1369
1370
1370
1371
1371
1372
1372
1373
1373
1374
1374
1375
1375
1376
1376
1377
1377
1378
1378
1379
1379
1380
1380
1381
1381
1382
1382
1383
1383
1384
1384
1385
1385
1386
1386
1387
1387
1388
1388
1389
1389
1390
1390
1391
1391
1392
1392
1393
1393
1394
1394
1395
1395
1396
1396
1397
1397
1398
1398
1399
1399
1400
1400
1401
1401
1402
1402
1403
1403
1404
1404
1405
1405
1406
1406
1407
1407
1408
1408
1409
1409
1410
1410
1411
1411
1412
1412
1413
1413
1414
1414
1415
1415
1416
```

```

128         }
129     }
130
131     }
132 } else if (input == VALUES_FROM_CSV) {
133
134     printf("Inserimento di %d elementi nella matrice A.\n", A_num);
135     printf("Inserisci il percorso del file .csv:\n");
136     chars_read = getline(&buffer, &bufsize, stdin);
137     printf("\n");
138
139     if (chars_read < 0 || chars_read > PATH_MAX_LENGTH) {
140         printf("Errore nella lettura dell'input!");
141         printf("Applicazione terminata.\n");
142         free(buffer);
143         exit(INPUT_LINE_ERROR);
144     }
145
146     for (i = 0; i < chars_read; i++) {
147         if (buffer[i] != 10) {
148             A_mat_csv[i] = buffer[i];
149         }
150     }
151
152     printf("Inserimento di %d elementi nella matrice B.\n", B_num);
153     printf("Inserisci il percorso del file .csv:\n");
154     chars_read = getline(&buffer, &bufsize, stdin);
155     printf("\n");
156
157     if (chars_read < 0 || chars_read > PATH_MAX_LENGTH) {
158         printf("Errore nella lettura dell'input!");
159         printf("Applicazione terminata.\n");
160         free(buffer);
161         exit(INPUT_LINE_ERROR);
162     }
163
164     for (i = 0; i < chars_read; i++) {
165         if (buffer[i] != 10) {
166             B_mat_csv[i] = buffer[i];
167         }
168     }
169
170     fprintf(pbs_file, " %s %s", A_mat_csv, B_mat_csv);
171
172     free(buffer);
173 }
174
175     break;
176 }
177 case MULTIPLICATION_IDENTITY_TEST:
178 case MULTIPLICATION_TRANSPOSE_TEST:
179 case MULTIPLICATION_TRACE_TEST:
180 default:
181     break;
182 }
183
184 if (time_calc == OK_TIME_CALC) {
185     fprintf(pbs_file, "\ndone");
186 }
187
188 fprintf(pbs_file, "\n");
189
190 fprintf(pbs_file,

```

```

191     "echo '">>>>'\n"
192     "echo PBS: Job completato.\n"
193     "echo --- \n"
194 );
195
196 if (fclose(pbs_file) != 0) {
197     printf("Nessun file o directory con questo nome: %s\n", pbs_path);
198     printf("Esecuzione terminata.\n");
199     exit(FILE_CLOSING_ERROR);
200 }
201
202 printf("%s creato con successo!\n\n", pbs_path);
203 }
```

Listing A.25: La funzione *void createPBS(int A_rows, int A_cols, int B_rows, int B_cols, int n_proc, int input, int test, int time_calc, int pbs_count)*

A.5 menu.c

```

1 /*
2
3  menu.c
4  di Mario Gabriele Carofano
5  e Francesco Noviello
6
7 */
8 #include "./libraries/menufunc.h"
```

Listing A.26: Intestazione del file *menufunc.c*

```

1 int mat_dim = 0;
2 int A_rows = 0, A_cols = 0;
3 int B_rows = 0, B_cols = 0;
4 int scelta = DEFAULT_SCELTA, input = DEFAULT_INPUT, test = DEFAULT_TEST;
5 int i = 0, pbs_count = 1;
```

Listing A.27: Inizializzazione dell'ambiente di lavoro

```

1 printf("\n");
2 printf("Benvenuto nell'applicazione di testing per le esercitazioni di\n");
3 printf("Parallel and Distributed Computing A.A. 2023-2024\n\n");
4 printTitle();
```

Listing A.28: Introduzione

```

1 do {
2     printf("Scegli un'operazione da effettuare: \n");
3     printf("%d. \t Calcolo del prodotto matrice-matrice senza test.\n",
4            NO_TEST);
4     printf("%d. \t Esecuzione della suite di testing.\n", TESTING_SUITE);
5     printf("%d. \t Chiudere l'applicazione.\n\n", EXIT_APPLICATION);
6     scelta = getIntegerFromInput();
7     checkScelta(scelta, NO_TEST, EXIT_APPLICATION);
```

Listing A.29: Scelta dell'operazione da effettuare

```

1     switch(scelta) {
2         case NO_TEST:
3         {
4             printf("Inserisci la dimensione delle matrici:\n");
5             mat_dim = getIntegerFromInput();
6
7             if (mat_dim < 1) {
8                 printf("Devi inserire una dimensione maggiore di 1!\n");
9                 printf("Applicazione terminata.\n");
10                exit(MATRIX_DIMENSION_ERROR);
11            }
12
13            A_rows = A_cols = B_rows = B_cols = mat_dim;

```

Listing A.30: Scelta della dimensione delle matrici

```

1     printf("Scegli il tipo di input: \n");
2     printf("%d. \t Input da linea di comando (fino a %d elementi).\n",
3            VALUES_FROM_INPUT, OP_MAX_QUANTITY);
4     printf("%d. \t Input da file .csv.\n", VALUES_FROM_CSV);
5     input = getIntegerFromInput();
6     checkScelta(scelta, VALUES_FROM_INPUT, VALUES_FROM_CSV);

```

Listing A.31: Scelta dell'input

```

1         createPBS(A_rows, A_cols, B_rows, B_cols, 4, input, DEFAULT_TEST,
2                         NO_TIME_CALC, pbs_count++);
3
4     break;
}

```

Listing A.32: Creazione del file di esecuzione *.pbs*

```

1     case TESTING_SUITE:
2     {
3         /*
4             Nell'eseguire la suite di testing, il programma fornira':
5             - il risultato del prodotto matrice-matrice;
6             - il tempo di esecuzione relativo all'esecuzione
7                 del solo prodotto.
8             - un valore booleano che indica se il risultato in
9                 output e' corretto, in base al test scelto.
10            */
11
12         printf("Scegli un test da eseguire: \n");
13         printf("%d. \t Calcolo del prodotto con la matrice identita'.\n",
14                         MULTIPLICATION_IDENTITY_TEST);
15         printf("%d. \t Confronto di T(AB) con T(B) x T(A), dove T(-) e' la
16                         matrice trasposta.\n", MULTIPLICATION_TRANSPOSE_TEST);
17         printf("%d. \t Confronto di tr(AB) e tr(BA), dove tr(-) e' la
18                         traccia della matrice.\n", MULTIPLICATION_TRACE_TEST);
19         printf("%d. \t Chiudere la suite di testing.\n\n", EXIT_TEST);
20         test = getIntegerFromInput();
21         checkScelta(test, MULTIPLICATION_IDENTITY_TEST, EXIT_TEST);
22
23         if (test != EXIT_TEST) {
24             for (i = MIN_COEFF_TEST; i <= MAX_COEFF_TEST; i++) {
25                 A_rows = A_cols = B_rows = B_cols = 4 * 9 * i;
26                 createPBS(A_rows, A_cols, B_rows, B_cols, 1, DEFAULT_INPUT
27                               , test, OK_TIME_CALC, pbs_count++);
28                 createPBS(A_rows, A_cols, B_rows, B_cols, 4, DEFAULT_INPUT
29                               , test, OK_TIME_CALC, pbs_count++);
30                 createPBS(A_rows, A_cols, B_rows, B_cols, 9, DEFAULT_INPUT
31                               , test, OK_TIME_CALC, pbs_count++);
32             }
33         }
34
35         break;
36     }
}

```

Listing A.33: Applicazione della suite di testing

```

1     case EXIT_APPLICATION:
2     default:
3         break;
4     }
5
6 } while (scelta != EXIT_APPLICATION);
7 printf("Applicazione terminata.\n");
8 return 0;

```

Listing A.34: Uscita dall'applicativo

A.6 prova3.c

```
1 /*
2
3     prova3.c
4     di Mario Gabriele Carofano
5     e Francesco Noviello
6
7 */
8
9 #include "libraries/auxfunc.h"
10 #include "libraries/csvfunc.h"
```

Listing A.35: Intestazione del file prova3.c

```
1     int input = DEFAULT_INPUT, test = DEFAULT_TEST, time_calc = NO_TIME_CALC;
2     int pbs_count = 0, check_test = 1;
3
4     int i = 0, j = 0, k = 0;
5
6     double *A_mat = NULL, *B_mat = NULL;
7     int A_rows = 0, A_cols = 0;
8     int B_rows = 0, B_cols = 0;
9
10    double *loc_A_mat = NULL, *loc_B_mat = NULL, *loc_C_mat = NULL;
11    int loc_A_rows = 0, loc_A_cols = 0;
12    int loc_B_rows = 0, loc_B_cols = 0;
13    int loc_C_rows = 0, loc_C_cols = 0;
14
15    double time_start = 0.0, time_end = 0.0;
16    double time_loc = 0.0, time_tot = 0.0;
17
18    double grid_tmp = 0;
19    int n_proc = 0, id_proc = 0;
20    int grid_rows = 0, grid_cols = 0;
21    int *grid_dim = NULL, *period = NULL, *grid_coords = NULL, *remain_dims = NULL
22        ;
23
24    FILE *csv_A_mat = NULL, *csv_B_mat = NULL, *out_file = NULL;
25    char out_path[PATH_MAX_LENGTH] = {};
26
27    MPI_Comm comm, comm_row, comm_col;
28
29    srand(time(NULL));
```

Listing A.36: Inizializzazione dell'ambiente di lavoro

```
1     comm = MPI_COMM_WORLD;
2     MPI_Init(&argc, &argv);
3     MPI_Comm_rank(MPI_COMM_WORLD, &id_proc);
4     MPI_Comm_size(MPI_COMM_WORLD, &n_proc);
```

Listing A.37: Inizializzazione dell'ambiente MPI

```
1     if (id_proc == 0 && DEBUG) printf("Inizio esecuzione.\n");
2
3     if (id_proc == 0) {
4
5         if (argc < ARGS_QUANTITY) {
6             printf("Errore nella lettura degli argomenti di input!\n");
7             printf("Sono richiesti almeno %d argomenti in input!\n", ARGS_QUANTITY
8                 );
8             printf("Esecuzione terminata.\n");
```

```

9         MPI_Abort(comm, NOT_ENOUGH_ARGS_ERROR);
10    }
11    A_rows = argToInt(argv[1]);
12    A_cols = argToInt(argv[2]);
13    B_rows = argToInt(argv[3]);
14    B_cols = argToInt(argv[4]);
15
16    if (B_rows != A_cols) {
17        printf("Le matrici A,B non sono compatibili!\n");
18        printf("Esecuzione terminata.\n");
19        MPI_Abort(comm, MATRIX_DIMENSION_ERROR);
20    }
21
22    grid_tmp = sqrt(n_proc);
23
24    if (n_proc != 0 && grid_tmp != floor(grid_tmp)) {
25        printf("Il numero di processi (%d) non e' un quadrato perfetto.\n",
26               n_proc);
27        printf("Impossibile costruire una griglia di processi bidimensionale
28              quadrata.\n");
29        printf("Esecuzione terminata.\n");
30        MPI_Abort(comm, PROCESSOR_QUANTITY_ERROR);
31    }
32
33    grid_rows = grid_cols = floor(grid_tmp);
34
35    if ((A_cols % grid_cols != 0) || (B_cols % grid_cols != 0)) {
36        printf("Le matrici non sono compatibili con la griglia di processori!\n");
37        printf("Esecuzione terminata.\n");
38        MPI_Abort(comm, MATRIX_DIMENSION_ERROR);
39    }
40
41    input = argToInt(argv[5]);
42    test = argToInt(argv[6]);
43    time_calc = argToInt(argv[7]);
44
45    pbs_count = argToInt(argv[8]);
46
47    if (id_proc == 0 && DEBUG) printf("Lettura degli argomenti completata.\n");
48
49    MPI_Bcast(&A_rows, 1, MPI_INT, 0, comm);
50    MPI_Bcast(&A_cols, 1, MPI_INT, 0, comm);
51    MPI_Bcast(&B_rows, 1, MPI_INT, 0, comm);
52    MPI_Bcast(&B_cols, 1, MPI_INT, 0, comm);
53
54    MPI_Bcast(&grid_rows, 1, MPI_INT, 0, comm);
55    MPI_Bcast(&grid_cols, 1, MPI_INT, 0, comm);
56
57    MPI_Bcast(&input, 1, MPI_INT, 0, comm);
58    MPI_Bcast(&test, 1, MPI_INT, 0, comm);
59    MPI_Bcast(&time_calc, 1, MPI_INT, 0, comm);
60
61    MPI_Bcast(&pbs_count, 1, MPI_INT, 0, comm);
62
63    if (id_proc == 0 && DEBUG) printf("Distribuzione degli argomenti completata.\n
64    ");

```

Listing A.38: Lettura e distribuzione dei dati

```

1  if (n_proc > 1) {
2
3      grid_coords      = (int*) calloc(2, sizeof(int));
4      grid_dim         = (int*) calloc(2, sizeof(int));
5      period           = (int*) calloc(2, sizeof(int));
6      remain_dims      = (int*) calloc(2, sizeof(int));
7
8      grid_dim[0] = grid_rows;
9      grid_dim[1] = grid_cols;
10
11     period[0] = 1;
12     period[1] = 1;
13
14     MPI_Cart_create(MPI_COMM_WORLD, 2, grid_dim, period, 0, &comm);
15     MPI_Cart_coords(comm, id_proc, 2, grid_coords);
16
17     remain_dims[0] = 0;
18     remain_dims[1] = 1;
19     MPI_Cart_sub(comm, remain_dims, &comm_row);
20
21     remain_dims[0] = 1;
22     remain_dims[1] = 0;
23     MPI_Cart_sub(comm, remain_dims, &comm_col);
24
25 }
26
27 if (id_proc == 0 && DEBUG) printf("Creazione della griglia completata.\n");

```

Listing A.39: Creazione della griglia bidimensionale

```

1  if (n_proc == 1) {
2      sprintf(out_path, NOME_PROVA "/output/" NOME_PROVA "_%03d/proc%d.out",
3              pbs_count, id_proc
4      );
5  } else {
6      sprintf(out_path, NOME_PROVA "/output/" NOME_PROVA "_%03d/proc%d_%02d_%02d
7          .out",
8          pbs_count, id_proc, grid_coords[0], grid_coords[1]
9      );
10
11     if ((out_file = fopen(out_path, "w")) == NULL) {
12         printf("Nessun file o directory con questo nome: %s\n", out_path);
13         printf("Esecuzione terminata.\n");
14         MPI_Abort(comm, FILE_OPENING_ERROR);
15     }
16
17     if (id_proc == 0 && DEBUG) printf("File di output creato.\n");

```

Listing A.40: Creazione dei file di output

```

1  if (id_proc == 0) {
2
3      A_mat = (double*) calloc(A_rows * A_cols, sizeof(double));
4      B_mat = (double*) calloc(B_rows * B_cols, sizeof(double));
5
6      if (!A_mat && !B_mat) {
7          printf("Le matrici globali non sono state allocate correttamente!\n");
8          printf("Esecuzione terminata.\n");
9          MPI_Abort(comm, ALLOCATION_ERROR);
10     }
11
12     switch (test) {

```

```

13     case DEFAULT_TEST:
14     {
15         switch (input) {
16             case VALUES_FROM_INPUT:
17             {
18                 if (((A_rows*A_cols)+(B_rows*B_cols)) <= OP_MAX_QUANTITY)
19                 {
20                     k = 1;
21                     for (i = 0; i < A_rows; i++) {
22                         for (j = 0; j < A_cols; j++) {
23                             A_mat[i*A_cols + j] = arg.ToDouble(argv[
24                                     ARGS_QUANTITY+k]);
25                             k++;
26                         }
27                     }
28                     for (i = 0; i < B_rows; i++) {
29                         for (j = 0; j < B_cols; j++) {
30                             B_mat[i*B_cols + j] = arg.ToDouble(argv[
31                                     ARGS_QUANTITY+k]);
32                             k++;
33                         }
34                     }
35                 } else {
36                     getRandomMatrix(A_mat, A_rows, A_cols);
37                     getRandomMatrix(B_mat, B_rows, B_cols);
38                 }
39                 break;
40             }
41             case VALUES_FROM_CSV:
42             {
43                 if ((csv_A_mat = fopen(argv[ARGS_QUANTITY+1], "r")) ==
44                     NULL) {
45                     printf("Nessun file o directory con questo nome: %s\n"
46                           , argv[ARGS_QUANTITY+1]);
47                     printf("Applicazione terminata.\n");
48                     MPI_Abort(comm, FILE_OPENING_ERROR);
49                 }
50
51                 if ((csv_B_mat = fopen(argv[ARGS_QUANTITY+2], "r")) ==
52                     NULL) {
53                     printf("Nessun file o directory con questo nome: %s\n"
54                           , argv[ARGS_QUANTITY+2]);
55                     printf("Applicazione terminata.\n");
56                     MPI_Abort(comm, FILE_OPENING_ERROR);
57                 }
58
59                 getMatrixFromCSV(csv_A_mat, A_mat, A_rows, A_cols, comm);
60                 getMatrixFromCSV(csv_B_mat, B_mat, B_rows, B_cols, comm);
61
62                 if (fclose(csv_A_mat) != 0) {
63                     printf("Nessun file o directory con questo nome: %s\n"
64                           , argv[ARGS_QUANTITY+1]);
65                     printf("Applicazione terminata.\n");
66                     MPI_Abort(comm, FILE_CLOSING_ERROR);
67                 }
68
69                 if (fclose(csv_B_mat) != 0) {
70                     printf("Nessun file o directory con questo nome: %s\n"
71                           , argv[ARGS_QUANTITY+2]);
72                     printf("Applicazione terminata.\n");
73                     MPI_Abort(comm, FILE_CLOSING_ERROR);
74                 }
75             }
76         }
77     }
78 }
```

```

67
68         break;
69     }
70     default:
71         break;
72 }
73 break;
74 }
75 case MULTIPLICATION_IDENTITY_TEST:
76 {
77     getRandomMatrix(A_mat, A_rows, A_cols);
78     getIdentityMatrix(B_mat, B_rows, B_cols);
79     break;
80 }
81 case MULTIPLICATION_TRANSPOSE_TEST:
82 case MULTIPLICATION_TRACE_TEST:
83 {
84     getRandomMatrix(A_mat, A_rows, A_cols);
85     getRandomMatrix(B_mat, B_rows, B_cols);
86     break;
87 }
88 default:
89     break;
90 }
91
92 if (n_proc > 1) fprintf(out_file, "\n--- MATRICI GLOBALI ---\n");
93 fprintf(out_file, "\nMatrice A di dimensione %d x %d:\n", A_rows, A_cols);
94 fprintfMatrix(out_file, A_mat, A_rows, A_cols, "%f");
95
96 fprintf(out_file, "\nMatrice B di dimensione %d x %d:\n", B_rows, B_cols);
97 if (test == MULTIPLICATION_IDENTITY_TEST) {
98     fprintfMatrix(out_file, B_mat, B_rows, B_cols, "%1.0f");
99 } else {
100     fprintfMatrix(out_file, B_mat, B_rows, B_cols, "%f");
101 }
102 if (n_proc > 1) fprintf(out_file, "\n-----\n");
103
104 }
105
106 if (id_proc == 0 && DEBUG) printf("Generazione delle matrici globali
completata.\n");

```

Listing A.41: Lettura e/o generazione degli elementi della matrice

```

1 if (n_proc == 1) {
2
3     loc_A_mat = A_mat;
4     loc_A_rows = A_rows;
5     loc_A_cols = A_cols;
6
7     loc_B_mat = B_mat;
8     loc_B_rows = B_rows;
9     loc_B_cols = B_cols;
10
11 } else {
12
13     loc_A_mat = scatterMatrixToGrid(
14         A_mat, A_rows, A_cols,
15         &loc_A_rows, &loc_A_cols,
16         id_proc, grid_dim, grid_coords, comm
17     );
18
19     loc_B_mat = scatterMatrixToGrid(

```

```

20         B_mat, B_rows, B_cols,
21         &loc_B_rows, &loc_B_cols,
22         id_proc, grid_dim, grid_coords, comm
23     );
24
25     if (id_proc == 0) fprintf(out_file, "\n--- MATRICI LOCALI ---\n");
26     fprintf(out_file, "\nMatrice loc_A di dimensione %d x %d:\n", loc_A_rows,
27             loc_A_cols);
28     fprintfMatrix(out_file, loc_A_mat, loc_A_rows, loc_A_cols, "%f");
29
30     fprintf(out_file, "\nMatrice loc_B di dimensione %d x %d:\n", loc_B_rows,
31             loc_B_cols);
32     if (test == MULTIPLICATION_IDENTITY_TEST) {
33         fprintfMatrix(out_file, loc_B_mat, loc_B_rows, loc_B_cols, "%1.0f");
34     } else {
35         fprintfMatrix(out_file, loc_B_mat, loc_B_rows, loc_B_cols, "%f");
36     }
37     if (id_proc == 0) fprintf(out_file, "\n-----\n");
38
39     if (id_proc == 0 && DEBUG) printf("Distribuzione delle matrici completata.\n")
40     ;
41     if (n_proc > 1 && id_proc == 0) {
42         free(A_mat);
43         free(B_mat);
44     }
45     if (id_proc == 0 && DEBUG) printf("Pulizia della memoria completata.\n");

```

Listing A.42: Distribuzione delle matrici

```

1 if (time_calc == OK_TIME_CALC) {
2     MPI_Barrier(comm);
3     time_start = MPI_Wtime();
4 }

```

Listing A.43: Inizio del calcolo dei tempi di esecuzione

```

1 loc_C_rows = loc_A_rows;
2 loc_C_cols = loc_B_cols;
3 loc_C_mat = (double*) calloc(loc_C_rows * loc_C_cols, sizeof(double));
4
5 if (!loc_C_mat) {
6     printf("La matrice del risultato non e' stata allocata correttamente!\n");
7     printf("Esecuzione terminata.\n");
8     MPI_Abort(comm, ALLOCATION_ERROR);
9 }
10
11 bmr(
12     loc_A_mat, loc_B_mat, loc_C_mat,
13     loc_A_rows, loc_A_cols,
14     loc_B_rows, loc_B_cols,
15     loc_C_rows, loc_C_cols,
16     comm, comm_row, comm_col
17 );
18
19 if (id_proc == 0 && DEBUG) printf("Applicazione dell'algoritmo BMR completato
. \n");

```

Listing A.44: Applicazione del BMR per il calcolo del prodotto matrice-matrice

```

1  if (time_calc == OK_TIME_CALC) {
2
3      MPI_Barrier(comm);
4      time_end = MPI_Wtime();
5      time_loc = time_end - time_start;
6      MPI_Reduce(&time_loc, &time_tot, 1, MPI_DOUBLE, MPI_MAX, 0, comm);
7
8      if (id_proc == 0) {
9
10         // Si veda il sezione "Futuri sviluppi" della documentazione.
11         // switch (test) {
12         //   case MULTIPLICATION_IDENTITY_TEST:
13         //   {
14         //       // ...
15         //       break;
16         //   }
17         //   case MULTIPLICATION_TRANSPOSE_TEST:
18         //   {
19         //       // ...
20         //       break;
21         //   }
22         //   case MULTIPLICATION_TRACE_TEST:
23         //   {
24         //       // ...
25         //       break;
26         //   }
27         //   default:
28         //       break;
29     }
30
31     if (check_test) {
32
33         if (id_proc == 0 && DEBUG)
34             printf("\nCalcolo del prodotto matrice-matrice terminato in %e
35                   sec\n", time_tot);
36
37         writeTimeCSV(
38             CSV_TIME_PATH"/"NOME_PROVA"_time.csv",
39             A_rows, A_cols, B_rows, B_cols,
40             n_proc, input, test,
41             time_tot,
42             comm
43         );
44     }
45 }

```

Listing A.45: Salvataggio del calcolo dei tempi di esecuzione

```

1 // Si veda il sezione "Futuri sviluppi" della documentazione.
2 // if (id_proc == 0) {
3
4     // if (n_proc == 1) {
5     //     C_mat = loc_C_mat;
6     //     C_rows = A_rows;
7     //     C_cols = B_cols;
8     // } else {
9     //     // ...
10    // }
11
12    // fprintf(out_file, "\nRisultato globale:");
13    // fprintf(out_file, "\nMatrice C di dimensione %d x %d:\n", C_rows, C_cols);
14    // fprintfMatrix(out_file, C_mat, C_rows, C_cols, "%f");

```

```

15 // free(C_mat);
16 // }
17
18
19 fprintf(out_file, "\nRisultato locale:");
20 fprintf(out_file, "\nMatrice loc_C di dimensione %d x %d:\n", loc_C_rows,
21 loc_C_cols);
22 fprintfMatrix(out_file, loc_C_mat, loc_C_rows, loc_C_cols, "%f");
23
24 if (id_proc == 0 && DEBUG) printf("Stampa dell'output completata.\n");

```

Listing A.46: Stampa dell'output

```

1 MPI_Barrier(comm);
2 free(loc_A_mat);
3 free(loc_B_mat);
4 free(loc_C_mat);
5
6 if (n_proc > 1) {
7     free(grid_coords);
8     free(grid_dim);
9     free(period);
10    free(remain_dims);
11 }
12
13 if (fclose(out_file) != 0) {
14     printf("Nessun file o directory con questo nome: %s\n", out_path);
15     printf("Esecuzione terminata.\n");
16     MPI_Abort(comm, FILE_CLOSING_ERROR);
17 }
18
19 if (id_proc == 0 && DEBUG) printf("\nEsecuzione terminata.\n");
20
21 MPI_Finalize();

```

Listing A.47: Terminazione dell'esecuzione

A.7 media-csv.c

```

1 /*
2
3  media-csv.c
4  di Mario Gabriele Carofano
5  e Francesco Noviello
6
7 */
8 // COSTANTI
9
10 #define A_ROWS 0
11 #define A_COLS 1
12 #define B_ROWS 2
13 #define B_COLS 3
14 #define N_PROC 4
15 #define INPUT 5
16 #define TEST 6
17 #define TIME 7
18
19 #define BURST_SIZE 10
20 // LIBRERIE E ALTRE FUNZIONI
21
22 #include "libraries/csvfunc.h"

```

Listing A.48: Intestazione del file media-csv.c

```

1 void calcola_media(double *mat, int rows, int cols, MPI_Comm comm) {
2
3     double avg = 0.0;
4     int i = 0, k = 0;
5
6     for (k = 0; k < rows; k += BURST_SIZE) {
7         avg = 0.0;
8
9         for (i = 0; i < BURST_SIZE; i++) {
10             avg += mat[(k+i)*cols + TIME];
11         }
12
13         avg /= BURST_SIZE;
14         writeTimeCSV(
15             CSV_TIME_PATH"/"NOME_PROVA"_time_avg.csv",
16             (int) mat[k*cols + A_ROWS],
17             (int) mat[k*cols + A_COLS],
18             (int) mat[k*cols + B_ROWS],
19             (int) mat[k*cols + B_COLS],
20             (int) mat[k*cols + N_PROC],
21             (int) mat[k*cols + INPUT],
22             (int) mat[k*cols + TEST],
23             avg,
24             comm
25         );
26     }
27 }

```

Listing A.49: La funzione `void calcola_media(double *mat, int rows, int cols, MPI_Comm comm)`

```

1 int main(int argc, char **argv) {
2
3     int csv_rows = 0, csv_cols = 0, csv_size = 0;
4     int i = 0;
5     double *mat = NULL;
6     FILE *csv_mat = NULL;
7
8     MPI_Init(&argc, &argv);
9
10    if ((csv_mat = fopen(argv[1], "r")) == NULL) {
11        printf("Nessun file o directory con questo nome: %s\n", argv[1]);
12        printf("Applicazione terminata.\n");
13        MPI_Abort(MPI_COMM_WORLD, FILE_OPENING_ERROR);
14    }
15
16    getDimensionsFromCSV(csv_mat, &csv_size, &csv_rows, &csv_cols);
17
18    mat = (double*) calloc(csv_rows * csv_cols, sizeof(double));
19
20    getMatrixFromCSV(csv_mat, mat, csv_rows, csv_cols, MPI_COMM_WORLD);
21    calcola_media(mat, csv_rows, csv_cols, MPI_COMM_WORLD);
22
23    if ((fclose(csv_mat)) != 0) {
24        printf("Nessun file o directory con questo nome: %s\n", argv[1]);
25        printf("Applicazione terminata.\n");
26        MPI_Abort(MPI_COMM_WORLD, FILE_CLOSING_ERROR);
27    }
28
29    free(mat);
30    MPI_Finalize();
31    return 0;
32 }

```

Listing A.50: Il main di media-csv.c

Sitografia

- [1] Open MPI, *La libreria mpi.h (v. 1.4.5)*, <https://www.open-mpi.org/doc/v1.4/>.
- [2] C++ Reference, *La libreria stdio.h*, <https://cplusplus.com/reference/cstdio>.
- [3] Carlos III University Of Madrid, *La funzione getline*,
https://www.it.uc3m.es/pbasanta/asng/course_notes/input_output_getline_en.html.
- [4] C++ Reference, *La libreria stdlib.h*, <https://cplusplus.com/reference/cstdlib>.
- [5] C++ Reference, *La libreria errno.h*, <https://cplusplus.com/reference/cerrno>.
- [6] C++ Reference, *La libreria limits.h*, <https://cplusplus.com/reference/climits>.
- [7] C++ Reference, *La libreria string.h*, <https://cplusplus.com/reference/cstring>.
- [8] C++ Reference, *La libreria tgmath.h*, <https://cplusplus.com/reference/ctgmath>.
- [9] C++ Reference, *La libreria time.h*, <https://cplusplus.com/reference/ctime>.
- [10] C++ Reference, *La libreria ctype.h*, <https://cplusplus.com/reference/cctype>.