

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-vettore, in ambiente di calcolo parallelo su
architettura MIMD a memoria condivisa**

CANDIDATI:

CAROFANO Mario Gabriele - N97000437

NOVIELLO Francesco - N97000436

Anno Accademico 2023-2024

Indice

1	Definizione e analisi del problema	1
2	Errori, costanti e librerie	3
2.1	Errori	3
2.2	Costanti	6
2.3	Librerie	10
2.3.1	Libreria omp.h	10
2.3.2	Libreria auxfunc.h	11
2.3.3	Libreria menufunc.h	13
2.3.4	Altre librerie	14
3	Creazione del file di esecuzione .pbs	17
3.1	Inizializzazione dell'ambiente di lavoro	18
3.2	Applicazione della suite di testing	19
3.3	Scelta delle dimensioni della matrice e del vettore	20
3.4	Scrittura del file .pbs	22
4	Implementazione dell'algoritmo	25
4.1	Inizializzazione dell'ambiente di lavoro	26
4.2	Lettura dei dati	29
4.3	Allocazione della matrice e dei vettori	30
4.4	Lettura e/o generazione del valore degli operandi	31
4.5	Calcolo del prodotto matrice-vettore	33
4.6	Calcolo dei tempi di esecuzione	34
4.7	Stampa dell'output e terminazione	35
5	Analisi dei tempi e delle prestazioni	37
6	Conclusioni	43
6.1	Futuri sviluppi	44

A Elenco dei listati	45
A.1 auxfunc.c	45
A.2 menufunc.c	54
A.3 menu.c	61
A.4 prova2.c	64
A.5 media-csv.c	70
Sitografia	73

Capitolo 1

Definizione e analisi del problema

L'algoritmo che viene presentato in questa relazione ha l'obiettivo di effettuare il calcolo del prodotto matrice-vettore in ambiente di calcolo parallelo su architettura MIMD *Multiple Instruction, Multiple Data (MIMD)* a memoria condivisa¹.

- L'algoritmo è stato implementato in linguaggio C.
- Il programma utilizza la libreria *Open MultiProcessing (OpenMP)* per la gestione dei processori / cores e, in particolare, dei thread generati dal processo in esecuzione².

¹Da questo punto in poi della relazione si farà uso anche del termine MIMD-SM

²In un'architettura MIMD-SM si può dividere il carico di lavoro dello stesso processo in più thread per ottenere maggior leggerezza ed efficienza, a discapito della sincronizzazione degli accessi alla memoria condivisa

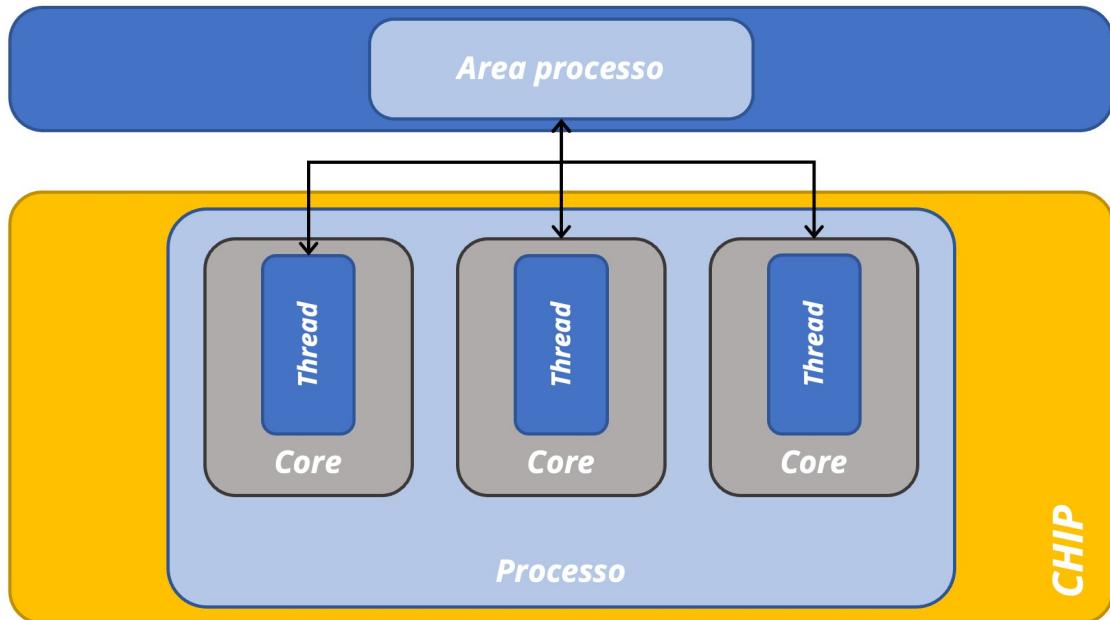


Figura 1.1: Esempio di architettura MIMD-SM multicore

- Dato che il programma è progettato per essere eseguito su un'architettura MIMD-SM, non è necessaria la fase di distribuzione degli operandi della matrice e del vettore, appunto perché sono già disponibili a tutti i processori.

In particolare, il prodotto fra una matrice A di dimensioni $m \times n$ e un vettore b di dimensione n restituisce come risultato un vettore c di dimensione m , la cui i -esima componente è la somma dei prodotti fra ogni componente della i -esima riga della matrice A e ogni componente del vettore b , prese a uno a uno. Per questo motivo, è necessario che il vettore b sia di dimensione n .

Si può riassumere questa descrizione con il seguente pseudocodice:

```

for  $i \in [1, m]$  do
   $c[i] \leftarrow 0$ .
  for  $j \in [1, n]$  do
     $c[i] \leftarrow c[i] + A[i][j] * b[j]$ .
  end for
end for

```

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.

In particolare, per la loro definizione si utilizza la direttiva del preprocessore `#define` (principalmente, per una miglior leggibilità del codice). In questo modo, prima dell'inizio della compilazione del programma, il preprocessore sostituisce nel codice tali definizioni con i valori ad essi associati.

- `#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 poichè 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 poichè 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 perchè non ha inserito un numero reale nel formato corretto.

- `#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 nel formato corretto.

- `#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 determinato file.

- `#define OVERFLOW_ERROR 10`

Si utilizza per segnalare all’utente che l’esecuzione del programma è stata terminata siccome il file .csv letto in input contiene dei valori float con una maggior precisione di quella descritta dalla costante *CSV_FIELD_PRECISION*.

- `#define MATRIX_DIMENSION_ERROR 11`

Si utilizza per segnalare all’utente che l’esecuzione del programma è stata terminata poichè ha inserito delle dimensione per la matrice più grandi di quelle del file .csv.

- `#define VECTOR_DIMENSION_ERROR 12`

Si utilizza per segnalare all’utente che l’esecuzione del programma è stata terminata siccome ha inserito una dimensione per il vettore più grande di quella del file .csv.

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 3.

- `#define MULTIPLICATION_INPUT_TEST 1`
- `#define MULTIPLICATION_CSV_TEST 2`
- `#define MULTIPLICATION_ONE_TEST 3`
- `#define MULTIPLICATION_SINGLE_NUMBER_TEST 4`
- `#define MULTIPLICATION_EIGENVECTOR_TEST 5`
- `#define EXIT_TEST 6`

Queste 6 costanti sono valori interi che rappresentano le possibili scelte che può inserire l'utente nel menù della suite di test dell'applicazione. In particolare, la costante *MULTIPLICATION_INPUT_TEST* si utilizza per eseguire l'algoritmo del calcolo del prodotto matrice-vettore utilizzando i valori degli operandi scelti dall'utente oppure generati in modo casuale.

- `#define OP_MAX_QUANTITY 25`

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

- `#define OP_MIN_EXP_TEST 2`

Si utilizza come esponente minimo per determinare la quantità di operandi nella creazione dei casi di test.

- `#define OP_MAX_EXP_TEST 4`

Si utilizza come esponente massimo per determinare la quantità di operandi 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 TIME_PRECISION 1000000.0`

Si utilizza per modificare la sensibilità dei tempi di esecuzione recuperati durante l'esecuzione del programma. In questo caso, il valore 1000000.0 indica che il tempo sarà misurato in secondi.

- `#define NOME_PROVA "prova2"`

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

- `#define NODE_NUMBER "1"`

Si utilizza nella funzione createPBS() per personalizzare il file .pbs con il numero di nodi nel cluster sui 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"`

Si utilizza per indicare alla funzione system() il percorso sul cluster dell'eseguibile 'mkdir'.

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

Si utilizza per indicare alla funzione system() il percorso sul cluster dell'eseguibile 'rm'.

- `#define CSV_TIME_PATH "../output"`

È 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:

1. *rows* : *int* (indica il numero di righe della matrice)
2. *cols* : *int* (indica il numero di colonne della matrice e, quindi, anche di elementi del vettore)
3. *threads* : *int* (indica il numero di threads parallelizzati)
4. *test* : *int* (indica il caso di test eseguito)
5. *t_tot* : *double* (indica il tempo di esecuzione impiegato)

- `#define OP_MAX_VALUE 100`

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

- `#define CSV_FIELD_PRECISION 12`

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

- `#define PATH_MAX_LENGTH 255`

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

2.3 Librerie

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

2.3.1 Libreria *omp.h*



Figura 2.1: Il logo di "Open MP Project"

OpenMP (Open Multi-Processing) è un'API che supporta la programmazione multipiattaforma a memoria condivisa in C, C++ e Fortran.

In particolare, la libreria *omp.h* del linguaggio C fornisce una serie di direttive, routine e variabili d'ambiente per consentire la programmazione parallela in modo semplice.

Nel dettaglio, il team di sviluppo ha fatto uso dell'istruzione `#pragma omp parallel for` per la parallelizzazione dell'operazione di calcolo del prodotto richiesto.

2.3.2 Libreria auxfunc.h

Questa è una delle due librerie implementate dal team di sviluppo e contiene alcune funzionalità aggiuntive per l'esecuzione del programma che calcola il prodotto matrice-vettore. Il codice è disponibile nella sezione A.1 dell'appendice A a pagina 45.

- `int argToInt(char *arg)`

La funzione riceve in input una stringa, verifica che sia non vuota e che sia diversa dal null-termination character, e la converte in un intero.

- `double argToDouble(char *arg)`

La funzione riceve in input una stringa, verifica che sia non vuota e che sia diversa dal null-termination character, e la converte in un double.

- `void writeTimeCSV(const char* path, int rows, int cols, int threads, int test, double t_tot)`

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

- `int getRowsFromCSV(const char* path)`

La funzione riceve in input il percorso di un file dal parametro in input `path`. Si utilizza per calcolarne il numero di righe, verificando prima di tutto che il file non sia vuoto.

- `int getColsFromCSV(const char* path)`

La funzione riceve in input il percorso di un file dal parametro in input `path`. Si utilizza per calcolarne il numero di campi separati da virgola, verificando prima di tutto che il file non sia vuoto. In particolare, questa funzione termina quando la lettura raggiunge la fine del file, nell'eventuale caso l'ultima riga abbia più campi delle altre righe.

- `void getMatrixFromCSV(const char* path, double** mat, int rows_mat, int cols_mat)`

Si utilizza per copiare gli elementi presenti nel file `.csv` in input, il cui percorso è specificato dal parametro in input `path`, in una matrice `mat` di numeri reali di dimensioni `rows_mat, cols_mat`.

- `void getVectorFromCSV(const char* path, double* vet, int cols_mat)`

Si utilizza per copiare gli elementi presenti nel file `.csv` in input, il cui percorso è specificato dal parametro in input `path`, in un vettore `vet` di numeri reali di dimensione `cols_mat`.

- `void freeMatrix(double** mat, int rows)`

Si utilizza per liberare correttamente la memoria allocata da una matrice `mat` il cui numero di righe è specificato dal parametro in input `rows`.

2.3.3 Libreria menufunc.h

Questa è l'altra delle due 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.2 dell'appendice A a pagina 54.

- `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 rows, int cols, int threads, int test, int time_calc, int pbs_count)`

La funzione riceve in input l'operazione da effettuare (calcolo del prodotto matrice-vettore o esecuzione della suite di testing), le dimensioni di righe e colonne della matrice e del vettore, e l'opzione di calcolo dei tempi di esecuzione per creare in modo dinamico il file *.pbs* necessario alla compilazione e all'esecuzione del programma sul cluster.

2.3.4 Altre librerie

- ***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.

- ***time.h***[8]

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).

- *sys/time.h*[9]

È una libreria che definisce la struttura `struct timeval` che rappresenta un intervallo di tempo, con campi per i secondi e i microsecondi.

Si utilizza per la gestione del tempo e delle informazioni temporali e, quindi, per calcolare il tempo di esecuzione del calcolo del prodotto matrice-vettore con la funzione `int gettimeofday(struct timeval *, void *)`.

- *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.

- *math.h*[11]

È una libreria che offre funzionalità per l'esecuzione di operazioni matematiche, dalle più generiche a quelle più specifiche e complesse. Si utilizza la funzione `double pow (double base, double exponent)` per calcolare le dimensioni di righe e colonne della matrice nell'esecuzione dei casi di test utilizzando dimensioni in potenza di 10.

Capitolo 3

Creazione del file di esecuzione .pbs

Questo capitolo è dedicato alla descrizione del menù di creazione dei file *.pbs*. Si esamineranno 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-vettore. Il codice è disponibile nella sezione A.3 dell'appendice A a pagina 61.

3.1 Inizializzazione dell'ambiente di lavoro

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

- `int rows = 0`

È una variabile di tipo intero per memorizzare la dimensione delle righe della matrice.

- `int cols = 0`

È una variabile di tipo intero per memorizzare la dimensione delle colonne della matrice e, di conseguenza, anche la dimensione del vettore da moltiplicare.

- `int test = MULTIPLICATION_INPUT_TEST`

È una variabile di tipo intero per memorizzare il caso di test da eseguire.

- `int i = 0`

È una variabile di tipo intero utilizzata genericamente nei cicli *for*.

- `int pbs_count = 1`

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

3.2 Applicazione della suite di testing

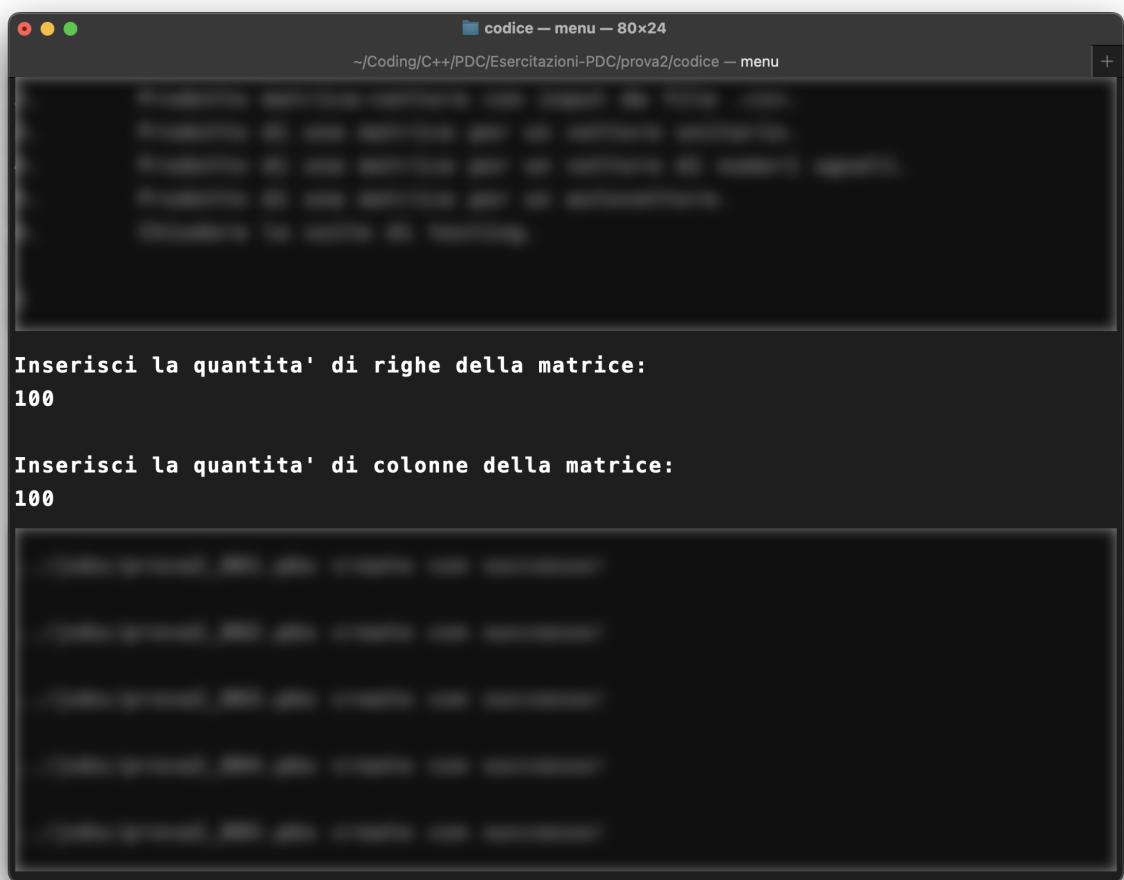
Come si vede nella figura 3.1, dal menù l’utente può procedere alla creazione del file `.pbs` selezionando l’esecuzione di un caso di test. Nella suite di testing implementata sono disponibili i seguenti casi di test:

- Prodotto matrice-vettore con scelta degli operandi.
 - Prodotto matrice-vettore con input da file *.csv*.
 - Prodotto di una matrice per un vettore unitario.
 - Prodotto di una matrice per un vettore di numeri uguali.
 - Prodotto di una matrice per un autovettore.

Figura 3.1: Avvio del menù e suite di testing.

3.3 Scelta delle dimensioni della matrice e del vettore

Nel caso in cui l'utente sceglie di eseguire uno dei primi due casi di test (cioè il prodotto matrice-vettore con scelta degli operandi o con input da file *.csv*), gli sarà chiesto la dimensione delle righe della matrice e la dimensione delle colonne della matrice e del vettore¹ (che devono essere almeno pari a 1).



The screenshot shows a terminal window with the following text:

```
codice — menu — 80x24
~/Coding/C++/PDC/Esercitazioni-PDC/prova2/codice — menu

Inserisci la quantita' di righe della matrice:
100

Inserisci la quantita' di colonne della matrice:
100
```

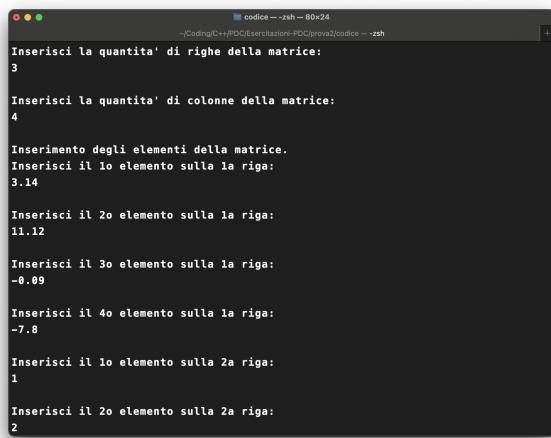
Figura 3.2: Inserimento delle dimensioni di righe e colonne.

¹La dimensione del vettore da moltiplicare viene impostata con il numero di colonne della matrice altrimenti l'operazione del prodotto tra matrice e vettore non sarebbe ben definita.

Nei rimanenti tre casi di test, la dimensione delle righe e delle colonne è prefissata:

- Matrice quadrata di dimensione 100×100 da moltiplicare ad un vettore riga di 100 elementi. Il risultato finale sarà un vettore riga di dimensione 100×1 .
- Matrice quadrata di dimensione 1000×1000 da moltiplicare ad un vettore riga di 1000 elementi. Il risultato finale sarà un vettore riga di dimensione 1000×1 .
- Matrice quadrata di dimensione 10000×10000 da moltiplicare ad un vettore colonna di 10000 elementi. Il risultato finale sarà un vettore riga di dimensione 10000×1 .

La funzione *createPBS*, infine, permette l'inserimento dei valori degli elementi della matrice e del vettore se tale quantità è minore o uguale a *OP_MAX_QUANTITY*, come mostrato nella figura 3.3a, altrimenti sono assegnati dei valori generati in modo pseudo-casuale.



```
codice -- zsh - 80x24
~/Coding/C++/POC/seriazioni-POC/prova2/codice -- zsh

Inserisci la quantita' di righe della matrice:
3

Inserisci la quantita' di colonne della matrice:
4

Inserimento degli elementi della matrice.
Inserisci il 1o elemento sulla la riga:
3.14

Inserisci il 2o elemento sulla la riga:
11.12

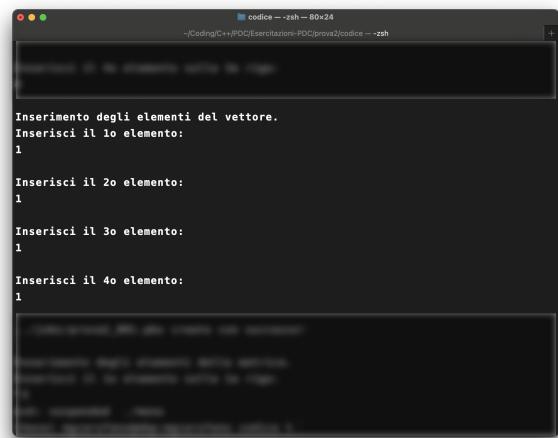
Inserisci il 3o elemento sulla la riga:
-0.09

Inserisci il 4o elemento sulla la riga:
-7.8

Inserisci il 1o elemento sulla 2a riga:
1

Inserisci il 2o elemento sulla 2a riga:
2
```

(a) Matrice



```
codice -- zsh - 80x24
~/Coding/C++/POC/seriazioni-POC/prova2/codice -- zsh

Inserimento degli elementi del vettore.
Inserisci il 1o elemento:
1

Inserisci il 2o elemento:
1

Inserisci il 3o elemento:
1

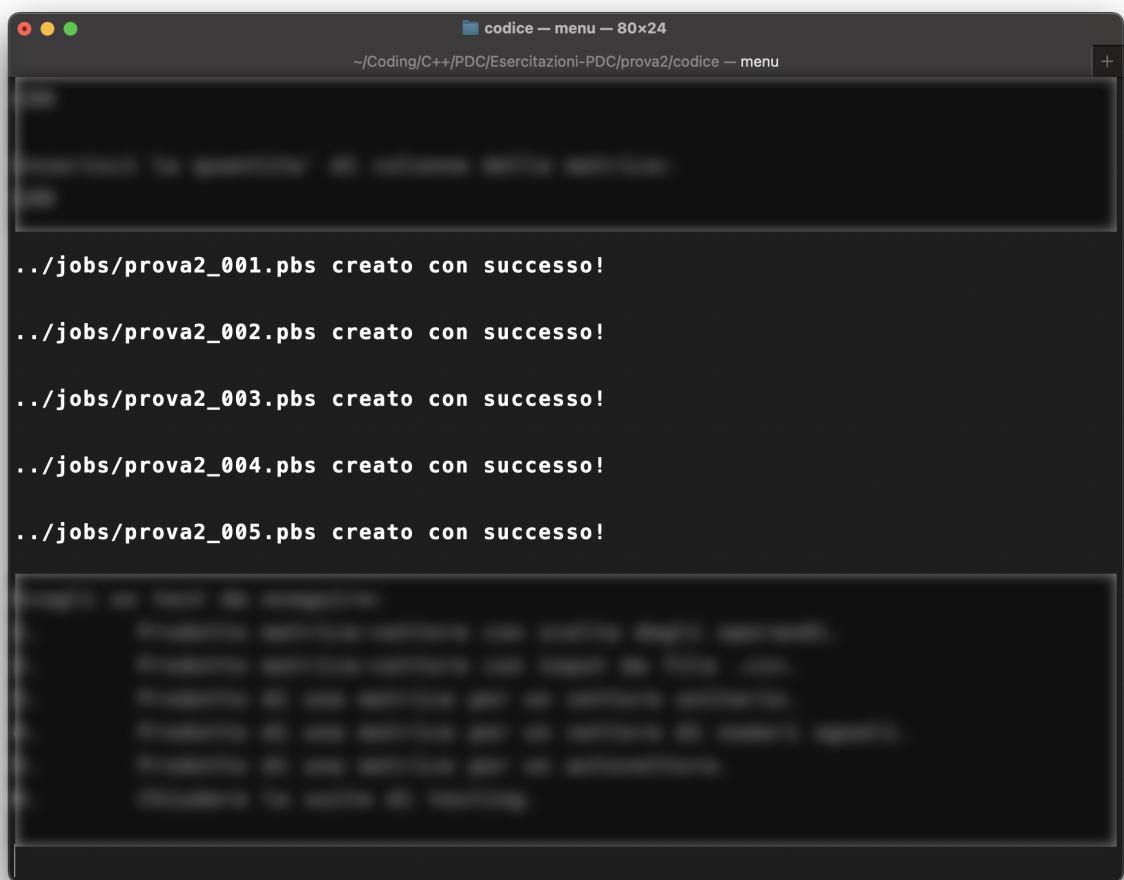
Inserisci il 4o elemento:
1
```

(b) Vettore

Figura 3.3: Inserimento manuale degli operandi.

3.4 Scrrittura del file .pbs

Al termine dell'esecuzione del programma del menù, come si evince dalla figura 3.4, se il processo di creazione del file *.pbs* termina con successo, allora si mostra il percorso relativo dove è collocato il file appena creato. In particolare, per ogni esecuzione del menù, si creano 5 diversi file *.pbs* per l'esecuzione su 1, 2, 4, 7 e 8 threads, rispettivamente.



The screenshot shows a terminal window titled "codice — menu — 80x24" with the path "~/Coding/C++/PDC/Esercitazioni-PDC/prova2/codice — menu". The window contains the following text output:

```
../jobs/prova2_001.pbs creato con successo!
../jobs/prova2_002.pbs creato con successo!
../jobs/prova2_003.pbs creato con successo!
../jobs/prova2_004.pbs creato con successo!
../jobs/prova2_005.pbs creato con successo!
```

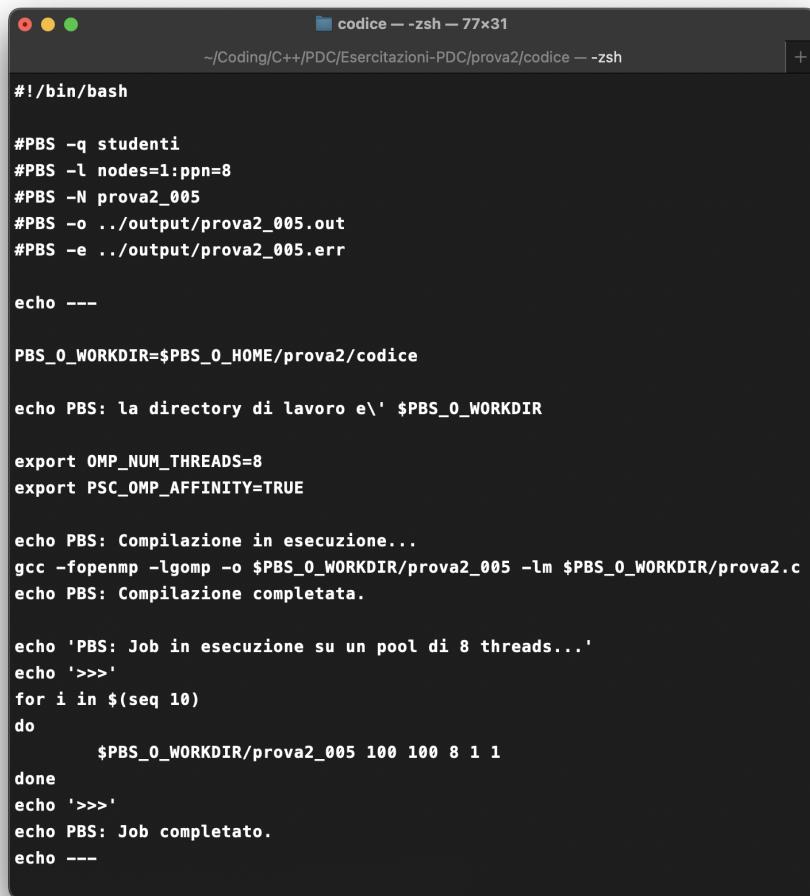
The terminal window has a dark background and light-colored text. The window title and path are at the top, and the output text is in the main body of the window.

Figura 3.4: Salvataggio dei file *.pbs*

Per evitare un eccessivo tempo di risposta da parte del cluster che comporterebbe un blocco delle esecuzioni in coda, attraverso il menù si generano in modo automatizzato differenti file *.pbs* per i differenti test da eseguire.

In particolare, ogni file *.pbs* generato è personalizzato in base alle scelte effettuate dall'utente come descritto nelle sezioni precedenti di questo capitolo.

La figura 3.5 mostra un esempio di file *.pbs* per il prodotto matrice-vettore in esecuzione su 8 thread, con matrice quadrata di dimensione 100×100 da moltiplicare ad un vettore riga di 100 elementi.



The image shows a terminal window titled "codice -- zsh -- 77x31" with the following content:

```
#!/bin/bash

#PBS -q studenti
#PBS -l nodes=1:ppn=8
#PBS -N prova2_005
#PBS -o ../output/prova2_005.out
#PBS -e ../output/prova2_005.err

echo --- 

PBS_0_WORKDIR=$PBS_0_HOME/prova2/codice

echo PBS: la directory di lavoro e\' $PBS_0_WORKDIR

export OMP_NUM_THREADS=8
export PSC_OMP_AFFINITY=TRUE

echo PBS: Compilazione in esecuzione...
gcc -fopenmp -lgomp -o $PBS_0_WORKDIR/prova2_005 -lm $PBS_0_WORKDIR/prova2.c
echo PBS: Compilazione completata.

echo 'PBS: Job in esecuzione su un pool di 8 threads...'
echo '>>>'
for i in $(seq 10)
do
    $PBS_0_WORKDIR/prova2_005 100 100 8 1 1
done
echo '>>>'
echo PBS: Job completato.
echo ---
```

Figura 3.5: Esempio di file *.pbs*

Capitolo 4

Implementazione dell'algoritmo

Questo capitolo è dedicato alla descrizione dell'algoritmo vero e proprio sviluppato per il calcolo del prodotto matrice-vettore in un ambiente di calcolo parallelo su architettura MIMD-SM. Si esamineranno gli input passati al programma e gli output attesi al termine dell'esecuzione dello stesso. Il codice è disponibile nella sezione A.4 dell'appendice A a pagina 64.

4.1 Inizializzazione dell'ambiente di lavoro

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

- `int test = MULTIPLICATION_INPUT_TEST`
- `int time_calc = NO_TIME_CALC`
- `int rows = 0`
- `int cols = 0`

Si veda la descrizione nella sezione 3.1 a pagina 18.

- `int threads = 0`

È una variabile di tipo intero che memorizza la quantità di thread che saranno parallelizzati dalle routine di OpenMP per l'esecuzione del calcolo del prodotto matrice-vettore.

- `int q_num = 0`

È una variabile di tipo intero che memorizza la quantità totale di elementi della matrice.

- `int i = 0`

- `int j = 0`

- `int k = 0`

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

- `double **mat`

È una matrice di elementi *double* da moltiplicare al vettore `vet`.

- `double *vet`

È un vettore di elementi *double* da moltiplicare alla matrice `mat`.

- `double *multiplication`

È un vettore di elementi *double* che ha il compito di memorizzare il risultato del prodotto tra la matrice `mat` ed il vettore `vet`.

- `struct timeval t`

È una struttura dati implementata nella libreria `sys/time.h`[9] utilizzata per memorizzare gli istanti di tempo con una sensibilità nella scala dei microsecondi.

- `double t_start = 0.0`

- `double t_end = 0.0`

Sono variabili di tipo *double* che memorizzano gli istanti di tempo in cui inizia e finisce il calcolo del prodotto matrice-vettore, rispettivamente.

- `double t_tot = 0.0`

È una variabile di tipo *double* che memorizza il tempo totale di esecuzione impiegato dal programma per calcolare il prodotto matrice-vettore.

- `int int_rand = 0`
- `double double_rand = 0.0`

Sono una variabile di tipo intero e una variabile di tipo *double* utilizzate per memorizzare valori generati in modo pseudo-casuale dalla funzione *rand()*.

4.2 Lettura dei dati

Dopo la definizione e l'inizializzazione delle variabili, si procede con la lettura e la verifica dei dati forniti in input al programma tramite il vettore (*argv*).

1. `rows = argToInt(argv[1]);`

È il numero di righe di cui sarà composta la matrice.

2. `cols = argToInt(argv[2]);`

È il numero di colonne di cui sarà composta la matrice, ma anche il numero di elementi di cui sarà composto il vettore.

3. `threads = argToInt(argv[3]);`

È il numero di thread sulla quale sarà diviso il carico di lavoro del processo padre, cioè quello sulla quale è in esecuzione il prodotto matrice-vettore. Inoltre, grazie alle routine messe a disposizione da OpenMP, l'esecuzione dei thread è parallelizzata quando possibile.

4. `test = argToInt(argv[4]);`

Indica il test che sarà eseguito. In particolare, tutti i valori che può assumere questo input sono specificati nella sezione 2.2 a pagina 6.

5. `time_calc = argToInt(argv[5]);`

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 questo input sono specificati nella sezione 2.2 a pagina 6.

4.3 Allocazione della matrice e dei vettori

Dato che questo programma è progettato per essere eseguito su architettura MIMD-SM, non si devono distribuire gli elementi della matrice e del vettore tra i diversi processori. Il prossimo passaggio, quindi, è quello di allocare la memoria necessaria per la matrice, il vettore da moltiplicare ed il vettore che conterrà il risultato finale, in modo da procedere con l'assegnazione degli elementi.

Per l'allocazione della memoria necessaria si utilizza la funzione `void* calloc()` per:

- Associare al puntatore `mat` l'indirizzo di memoria relativo allo spazio dedicato ai `rows` vettori di dimensione `cols` di elementi `double` della matrice.
- Associare al puntatore `vet` l'indirizzo di memoria dedicato ai `cols` operandi di tipo `double` del vettore da moltiplicare.
- Associare al puntatore `multiplication` l'indirizzo di memoria dedicato ai `rows` operandi di tipo `double` del vettore risultato.

4.4 Lettura e/o generazione del valore degli operandi

Una volta allocato lo spazio in memoria necessario, si passa alla lettura e/o generazione del valore degli operandi. Questa sezione tratta in dettaglio l'implementazione dello switch-case del codice A.29 a pagina 65.

- Il caso *MULTIPLICATION_INPUT_TEST*

In questo caso, l'utente ha scelto dal menù di creazione del *.pbs* di scegliere manualmente il valore degli elementi della matrice e del vettore. Se questi sono meno di *OP_MAX_QUANTITY*, allora essi sono letti dagli argomenti *argv* in input al programma; in caso contrario, allora ad ogni operando si assegna un numero reale generato in modo pseudo-casuale compreso nell'intervallo $[-100, 100] \in R$.

- Il caso *MULTIPLICATION_CSV_TEST*

In quest'altro caso, l'utente ha scelto dal menù di creazione del *.pbs* di inserire manualmente il valore degli elementi della matrice e del vettore utilizzando due file *.csv*. Se le dimensioni passate in input sono maggiori delle dimensioni dei file *.csv*, allora il programma termina restituendo l'errore *OVERFLOW_ERROR*, *MATRIX_DIMENSION_ERROR* oppure *VECTOR_DIMENSION_ERROR*.

- Il caso *MULTIPLICATION_ONE_TEST*

Questo è il primo vero e proprio caso di test della suite di testing: verifica se l'algoritmo del prodotto matrice-vettore calcola correttamente il prodotto di una matrice di elementi generati in modo casuale con il vettore unitario, cioè ad ogni operando del vettore è assegnato il valore 1. L'output atteso è un vettore costituito in ogni posizione dal valore della somma dei valori di ogni singola riga della matrice.

- Il caso *MULTIPLICATION_SINGLE_NUMBER_TEST*

Questo secondo caso di test è molto simile al precedente, ma concede all’utente la possibilità di scegliere l’unico valore da inserire nel vettore da moltiplicare, passato come argomento in input al programma. Ad ogni operando del vettore è, quindi, assegnato il valore scelto. L’output atteso è un vettore costituito dal valore della somma dei valori di ogni singola riga moltiplicato proprio per il valore scelto dall’utente.

- Il caso *MULTIPLICATION_EIGENVECTOR_TEST*

Questo caso di test è da implementare, si veda la sezione 6.1 a pagina 44.

4.5 Calcolo del prodotto matrice-vettore

Una volta aver caricato in memoria tutti gli elementi della matrice e del vettore, ed aver verificato che la matrice ed i vettori siano stati allocati correttamente, si può effettivamente procedere con il calcolo del prodotto matrice-vettore.

Questa è la parte del programma che viene parallelizzata utilizzando la direttiva di OpenMP `#pragma omp parallel for`. In particolare, a questa direttiva si aggiungono anche le seguenti clausule:

- `default (none)`

Consente di lasciare allo sviluppatore il compito di decidere quali variabili dovranno essere condivise tra i thread e quali variabili dovranno essere private per ogni thread.

- `shared (rows, cols, mat, vet, multiplication)`

Dato che è presente la clausola `default (none)`, è necessario specificare in questa clausola quali variabili dovranno essere condivise tra i thread del team.

- `private (i, j)`

Dato che è presente la clausola `default (none)`, è necessario specificare in questa clausola quali variabili dovranno essere private per ogni thread del team¹.

¹Cioè, ogni thread avrà la propria copia di queste variabili

4.6 Calcolo dei tempi di esecuzione

Nel caso in cui si sia scelto di calcolare i tempi di esecuzione, si assegna alla variabile ***t_start*** il valore dell’istante di tempo subito precedente all’inizio del calcolo del prodotto matrice-vettore utilizzando la funzione

`int gettimeofday(struct timeval *, void *).`

Al termine del calcolo del prodotto matrice-vettore, l’algoritmo assegna alla variabile ***t_end*** l’istante di tempo finale dell’operazione in questione, quindi calcola la distanza dall’istante di tempo ***t_start*** per stabilire l’intervallo temporale di esecuzione, cioè:

$$t_{tot} = t_{end} - t_{start} \quad (4.1)$$

Infine:

1. Si stampa il tempo impiegato dall’esecuzione utilizzando la notazione scientifica.
2. Si salvano all’interno di un file *.csv* tutte le informazioni riguardanti l’esecuzione, cioè: la dimensione delle righe della matrice, la dimensione delle colonne della matrice e di elementi del vettore da moltiplicare, il numero di thread del team, l’identificativo del test eseguito ed il tempo impiegato (in formato decimale).

4.7 Stampa dell'output e terminazione

Al termine dell'esecuzione del programma, quindi, si stampano i risultati ottenuti iterando su tutto il vettore *multiplication* che contiene il risultato. Per terminare con successo l'esecuzione, si libera lo spazio allocato in memoria per la matrice *mat*, il vettore da moltiplicare *vet* ed il vettore risultato *multiplication* utilizzando, rispettivamente, le funzioni `void freeMatrix(double** mat, int rows)` e `void free (void* ptr)`.

In particolare, l'esecuzione del comando *qsub* su tutti i file *.pbs* creati tramite l'apposito menù descritto nel capitolo 3 a pagina 17 produce 4 file:

1. L'eseguibile per il programma, la cui esecuzione è stata descritta in questo capitolo.
2. Un file con estensione *.out* contenente tutte le stampe ed i risultati ottenuti dall'esecuzione, cioè gli elementi del vettore risultato.
3. Un file con estensione *.err* contenente il log d'errore, per il salvataggio delle informazioni di errore.
4. Un file con estensione *.csv* contenente tutti i tempi di esecuzione e altri parametri già descritti nella sezione 4.6 a pagina 34.

Capitolo 5

Analisi dei tempi e delle prestazioni

La valutazione delle performance del software avvengono mediante i seguenti parametri:

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

Per ogni esperimento sono state effettuate 10 esecuzioni del test

MULTIPLICATION_ONE_TEST ed è stata, successivamente, considerata la media aritmetica dei tempi $T(p)$ di ciascuna prova.

$$T_m(p) = \frac{\sum_{n=1}^{10} T(p)}{10} \quad (5.1)$$

In particolare, è stato implementato un terzo algoritmo, sempre in linguaggio C, capace di calcolare la media dei tempi di esecuzione collezionati nel file *.csv* descritto nella sezione 4.7 a pagina 35. Il codice è disponibile nella sezione A.5 dell'appendice A a pagina 70.

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

Misura la riduzione del tempo $T(p)$ impiegato per l'esecuzione con p processori 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.

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

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

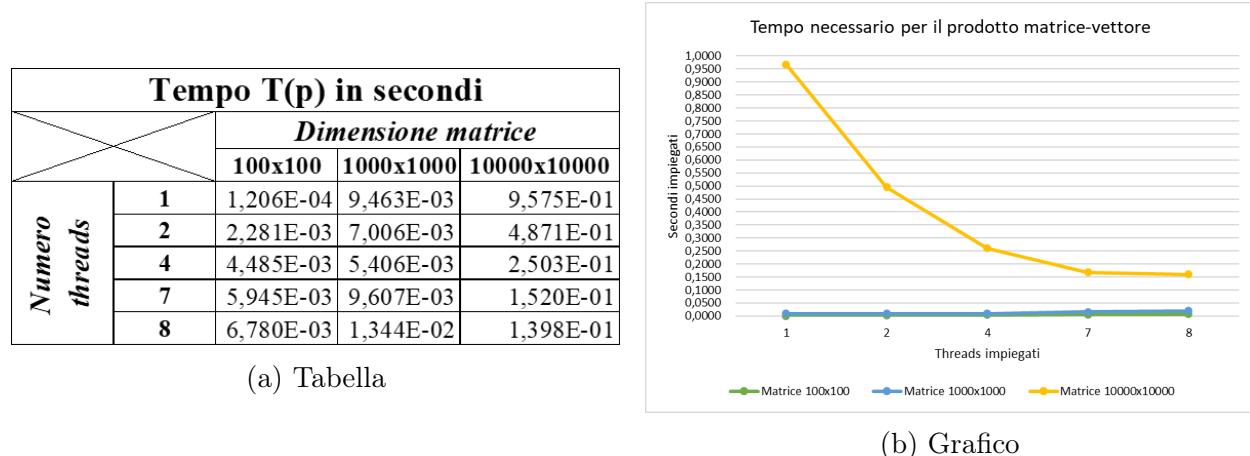


Figura 5.1: Confronti dei tempi di esecuzione.

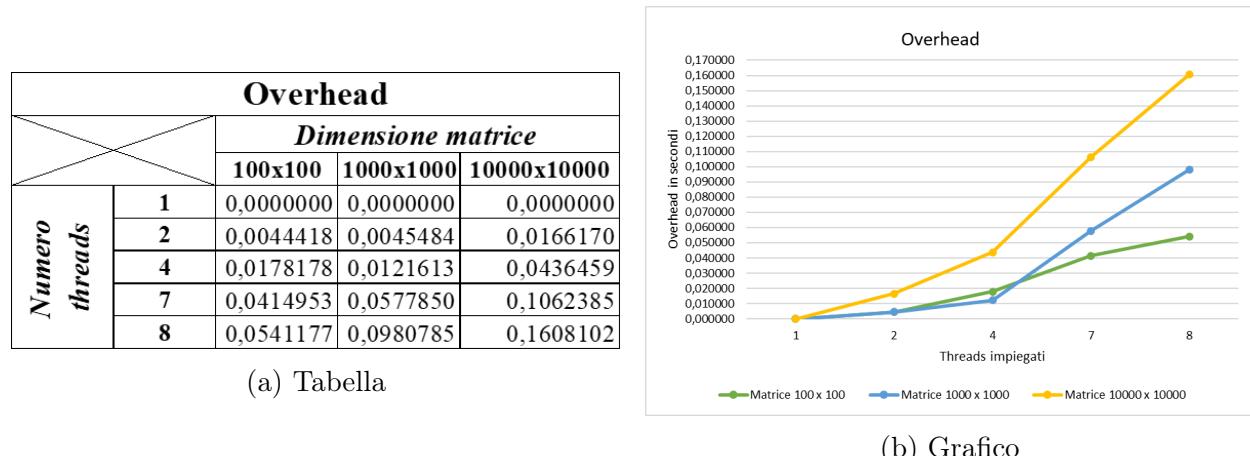
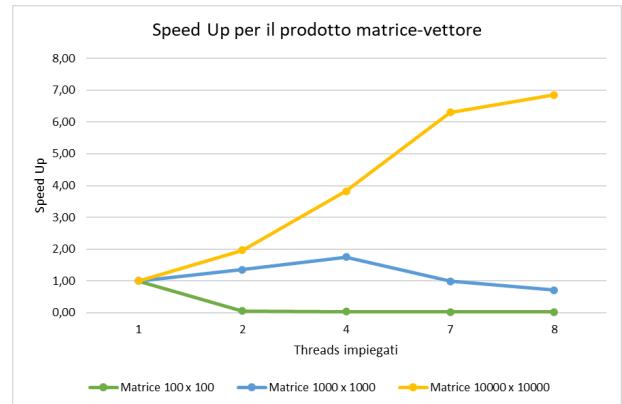


Figura 5.2: Confronti dei valori di overhead.

		Speed Up		
		Dimensione matrice		
Numero threads		100x100	1000x1000	10000x10000
	1	1,000000	1,000000	1,000000
	2	0,052863	1,350759	1,965884
	4	0,026890	1,750440	3,825622
	7	0,020284	0,985025	6,300912
	8	0,017787	0,703952	6,849654

(a) Tabella

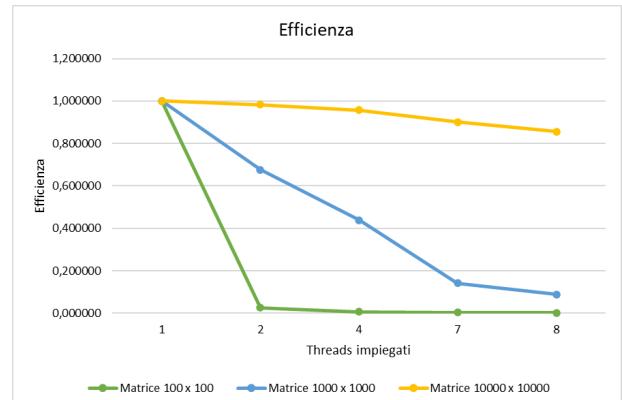


(b) Grafico

Figura 5.3: Confronti dei valori di speed up.

		Efficienza		
		Dimensione matrice		
Numero threads		100x100	1000x1000	10000x10000
	1	1,000000	1,000000	1,000000
	2	0,026432	0,675379	0,982942
	4	0,006723	0,437610	0,956405
	7	0,002898	0,140718	0,900130
	8	0,002223	0,087994	0,856207

(a) Tabella



(b) Grafico

Figura 5.4: Confronti dei valori dell'efficienza

I grafici mostrano chiaramente che per dimensioni piccole del problema, in questo caso per le dimensioni 100×100 e 1000×1000 , si hanno tempi di esecuzione eccellenti. Tuttavia, i valori di speed up ed efficienza che confrontano l'esecuzione sequenziale con l'esecuzione parallela su più thread, sono peggiori¹.

¹Sostanzialmente, i tempi di esecuzione sono più o meno costanti, anzi tendono addirittura leggermente ad aumentare.

Questa condizione può essere dovuta a diversi fattori, tra cui:

- **Overhead per la creazione e sincronizzazione dei thread.** Se le dimensioni della matrice sono piccole, il carico di lavoro potrebbe essere insufficiente per bilanciare l'overhead introdotto dalla creazione e sincronizzazione dei thread. In questo caso, l'allocazione delle risorse ai thread può richiedere più tempo rispetto all'effettivo calcolo.
- **Conflitti di memoria.** Con dimensioni ridotte della matrice, più thread potrebbero competere per l'accesso alla stessa porzione di memoria, causando conflitti e, quindi, rallentando l'esecuzione. Questo problema diventa sempre più evidente al crescere del numero di thread rispetto alle dimensioni della matrice.

La soluzione a questi problemi si presenta con l'aumento del carico di lavoro. Infatti, nell'ultima coppia di dimensioni della matrice, cioè 10000×10000 , si ottengono i migliori risultati sia per quanto riguarda la metrica del tempo di esecuzione che per le altre metriche di speed up ed efficienza.

Capitolo 6

Conclusioni

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

- Nel capitolo 1 è stato delineato il problema affrontato, e le caratteristiche dell'algoritmo implementato in architettura MIMD-SM.
- 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 o sviluppate dal team di sviluppo e dalle librerie di linguaggio disponibili.
- 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 un 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 test eseguito restituisce il risultato desiderato (al momento, è compito dell'utente e/o dello sviluppatore verificare la correttezza dei dati).
- Inoltre, come già è stato segnalato nella sezione 4.4 a pagina 31, si è pensato di introdurre un ulteriore caso di test nella suite di testing, cioè il caso *MULTIPLICATION_EIGENVECTOR_TEST*.

L'input è una matrice di dimensione $m \times n$ generati in modo pseudo-casuale. Si calcola uno degli autovalori della matrice¹ ed il corrispondente autovettore² e, infine, si esegue il calcolo del prodotto matrice-vettore tra la matrice in input e l'autovettore generato. L'output atteso è un vettore i cui elementi sono gli elementi del vettore da moltiplicare scalati per l'autovalore corrispondente, cioè:

$$Av = \lambda v \quad (6.1)$$

¹Si risolve l'equazione caratteristica $\det(A - \lambda I) = 0$.

²Si sostituisce l'autovalore trovato all'equazione caratteristica.

Appendice A

Elenco dei listati

A.1 auxfunc.c

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

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

```

1 int argToInt(char *arg) {
2
3     char *p;
4     long out_long = 0;
5
6     // Si inizializza il valore di 'errno' nella libreria <errno.h>
7     errno = 0;
8
9     /*
10      Si controlla che l'argomento passato al programma non sia una stringa
11      vuota. La funzione 'strlen', infatti, ne misura la dimensione.
12  */
13
14     if (strlen(arg) == 0) {
15         printf("Errore nella lettura degli argomenti di input!\n\n");
16         printf("Esecuzione terminata.\n");
17         exit(EMPTY_ARG_ERROR);
18     }
19
20     /*
21      Si preferisce l'utilizzo della funzione 'strtol' invece della
22      funzione 'atoi' perche' consente un maggior numero di controlli
23  */
24
25     out_long = strtol(arg, &p, 10);
26
27     /*
28      Si controlla che l'argomento passato al programma non sia una stringa
29      contenente il solo carattere '\0' (null-termination character) e non
30      ci siano stati altri errori durante la sua lettura.
31  */
32
33     if (*p != '\0' || errno != 0) {
34         printf("Errore nella lettura degli argomenti di input!\n\n");
35         printf("Esecuzione terminata.\n");
36         exit(INPUT_ARG_ERROR);
37     }
38
39     /*
40      Per una corretta conversione da long a int, si controlla
41      che il valore long sia rappresentabile come intero.
42  */
43
44     if (out_long < INT_MIN || out_long > INT_MAX) {
45         printf("Errore nella lettura degli argomenti di input!\n\n");
46         printf("Esecuzione terminata.\n");

```

```

47     exit(NOT_INT_ARG_ERROR);
48 }
49
50 return (int)out_long;
51 }

```

Listing A.2: 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        exit(EMPTY_ARG_ERROR);
12    }
13
14    out_double = strtod(arg, &p);
15
16    if (*p != '\0' || errno != 0) {
17        printf("Errore nella lettura degli argomenti di input!\n\n");
18        printf("Esecuzione terminata.\n");
19        exit(INPUT_ARG_ERROR);
20    }
21
22    return out_double;
23 }

```

Listing A.3: La funzione *double argToDouble(char*)*

```

1 void writeTimeCSV(const char* out_path, int rows, int cols, int threads,
2                   int test, double t_tot) {
3
4     FILE *csv_file;
5     int size = 0;
6
7     system(MKDIR_PATH" -p "CSV_TIME_PATH");
8
9     if ((csv_file = fopen(out_path, "a")) == NULL) {
10        printf("Errore durante l'esecuzione!\n");
11        printf("Applicazione terminata.\n");
12        exit(FILE_OPENING_ERROR);
13    }

```

```

14 // Se il file e' vuoto, allora non inserire '\n'
15 fseek(csv_file, 0, SEEK_END);
16     size = ftell(csv_file);
17 if (size != 0) {
18     fprintf(csv_file, "\n");
19 }
20
21 fseek(csv_file, 0, SEEK_SET);
22 fprintf(csv_file, "%d,%d,%d,%d,%1.9f",
23     rows, cols, threads, test, t_tot);
24
25 fclose(csv_file);
26
27 printf("%s aggiornato con successo!\n\n", out_path);
28
29 }

```

Listing A.4: La funzione *void writeTimeCSV(const char*, int, int, int, int, double)*

```

1 void writeTimeCSV(const char* out_path, int rows, int cols, int threads,
2     int test, double t_tot) {
3
4     FILE *csv_file;
5     int size = 0;
6
7     system(MKDIR_PATH" -p "CSV_TIME_PATH);
8
9     if ((csv_file = fopen(out_path, "a")) == NULL) {
10         printf("Errore durante l'esecuzione!\n");
11         printf("Applicazione terminata.\n");
12         exit(FILE_OPENING_ERROR);
13     }
14
15     // Se il file e' vuoto, allora non inserire '\n'
16     fseek(csv_file, 0, SEEK_END);
17     size = ftell(csv_file);
18     if (size != 0) {
19         fprintf(csv_file, "\n");
20     }
21
22     fseek(csv_file, 0, SEEK_SET);
23     fprintf(csv_file, "%d,%d,%d,%d,%1.9f",
24         rows, cols, threads, test, t_tot);
25
26     fclose(csv_file);
27
28     printf("%s aggiornato con successo!\n\n", out_path);

```

```
28  
29 }
```

Listing A.5: La funzione *void writeTimeCSV(const char*, int, int, int, int, double)*

```
1 void freeMatrix(double** mat, int rows) {  
2     int i = 0;  
3     for (i = 0; i < rows; i++) {  
4         free(mat[i]);  
5     }  
6     free(mat);  
7 }
```

Listing A.6: La funzione *void freeMatrix(double**, int)*

```
1 int getRowsFromCSV(const char* path) {  
2  
3     FILE* csv_file;  
4     char c = 0;  
5     int rows_csv = 0, size = 0;  
6  
7     if ((csv_file = fopen(path, "r")) == NULL) {  
8         printf("Errore durante l'esecuzione!\n");  
9         printf("Applicazione terminata.\n");  
10        exit(FILE_OPENING_ERROR);  
11    }  
12  
13    fseek(csv_file, 0, SEEK_END);  
14    size = ftell(csv_file);  
15    if (size == 0) {  
16        return 0;  
17    }  
18  
19    fseek(csv_file, 0, SEEK_SET);  
20  
21    do {  
22        c = fgetc(csv_file);  
23        if (c == 10) {  
24            rows_csv++;  
25        }  
26    } while (c != EOF);  
27  
28    return rows_csv+1;  
29 }
```

Listing A.7: La funzione *int getRowsFromCSV(const char*)*

```

1 int getColsFromCSV(const char* path) {
2
3     FILE* csv_file;
4     char c = 0;
5     int cols_csv = 0, size = 0;
6
7     if ((csv_file = fopen(path, "r")) == NULL) {
8         printf("Errore durante l'esecuzione!\n");
9         printf("Applicazione terminata.\n");
10        exit(FILE_OPENING_ERROR);
11    }
12
13    fseek(csv_file, 0, SEEK_END);
14    size = ftell(csv_file);
15    if (size == 0) {
16        return 0;
17    }
18
19    fseek(csv_file, 0, SEEK_SET);
20
21    do {
22        c = fgetc(csv_file);
23        if (c == 44) {
24            cols_csv++;
25        }
26        if (c == 10) {
27            cols_csv = 0;
28        }
29    } while (c != EOF);
30
31    return cols_csv+1;
32 }
```

Listing A.8: La funzione *int getColsFromCSV(const char*)*

```

1 void getMatrixFromCSV(const char* path, double** mat, int rows_mat, int
2     cols_mat) {
3     FILE *file;
4     char c, char_val[CSV_FIELD_PRECISION] = {};
5     int rows_csv = 0, cols_csv = 0;
6
7     int i = -1;
8     int row_count = 0, comma_count = 0;
9
10    if ((file = fopen(path, "r")) == NULL) {
11        printf("Errore durante l'esecuzione!\n");
12        printf("Applicazione terminata.\n");
```

```

12     exit(FILE_OPENING_ERROR);
13 }
14
15 rows_csv = getRowsFromCSV(path);
16 cols_csv = getColsFromCSV(path);
17
18 if ((rows_mat > rows_csv) || (cols_mat > cols_csv)) {
19     printf("La dimensione della matrice non e' corretta!\n");
20     printf("Applicazione terminata.\n");
21     exit(MATRIX_DIMENSION_ERROR);
22 }
23
24 do {
25     ++i;
26     c = fgetc(file);
27     if (c != 44) {
28         if (i < CSV_FIELD_PRECISION) {
29             char_val[i] = c;
30         } else {
31             printf("Errore nella lettura dell'input!\n\n");
32             printf("Applicazione terminata.\n");
33             exit(OVERFLOW_ERROR);
34         }
35     }
36
37     if (c == 44 || c == 10 || c == EOF) {
38         char_val[i] = '\0';
39         mat[row_count][comma_count++] = arg.ToDouble(char_val);
40
41         if (c == EOF) {
42             break;
43         }
44
45         if (c == 10 || comma_count == cols_mat) {
46             comma_count = 0;
47             row_count++;
48             if (row_count == rows_mat) {
49                 break;
50             }
51         }
52
53         for (i = 0; i < CSV_FIELD_PRECISION; i++) {
54             char_val[i] = '0';
55         }
56         char_val[i] = '\0';
57     }
}

```

```

58     i = -1;
59 }
60 } while (1);
61 }

```

Listing A.9: La funzione *void getMatrixFromCSV(const char*, double**, int, int)*

```

1 void getVectorFromCSV(const char* path, double* vet, int cols_mat) {
2     FILE *file;
3     char c, char_val[CSV_FIELD_PRECISION] = {};
4     int rows_csv = 0;
5
6     int i = -1;
7     int row_count = 0;
8
9     if ((file = fopen(path, "r")) == NULL) {
10         printf("Errore durante l'esecuzione!\n");
11         printf("Applicazione terminata.\n");
12         exit(FILE_OPENING_ERROR);
13     }
14
15     rows_csv = getRowsFromCSV(path);
16
17     if (cols_mat > rows_csv) {
18         printf("La dimensione del vettore non e' corretta!\n");
19         printf("Applicazione terminata.\n");
20         exit(VECTOR_DIMENSION_ERROR);
21     }
22
23     do {
24         ++i;
25         c = fgetc(file);
26         if (c != 10) {
27             if (i < CSV_FIELD_PRECISION) {
28                 char_val[i] = c;
29             } else {
30                 printf("Errore nella lettura dell'input!\n\n");
31                 printf("Applicazione terminata.\n");
32                 exit(OVERFLOW_ERROR);
33             }
34         }
35
36         if (c == 10 || c == EOF) {
37             char_val[i] = '\0';
38             vet[row_count] = argToDouble(char_val);
39
40             if (c == EOF) {

```

```

41         break;
42     }
43
44     if (c == 10) {
45         row_count++;
46         if (row_count == cols_mat) {
47             break;
48         }
49     }
50
51     for (i = 0; i < CSV_FIELD_PRECISION; i++) {
52         char_val[i] = '0';
53     }
54     char_val[i] = '\0';
55
56     i = -1;
57 }
58 } while (1);
59 }
```

Listing A.10: La funzione *void getVectorFromCSV(const char*, double*, int)*

A.2 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.11: Intestazione del file *menufunc.c*

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

Listing A.12: 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
7     // Inizializzazione del buffer con caratteri estratti dallo stream di
8     // input
9     chars_read = getline(&buffer, &bufsize, stdin);
10    printf("\n");
11
12    if (chars_read < 0) {
13
14        printf("Errore nella lettura dell'input!");
15        printf("Applicazione terminata.\n");
16        free(buffer);
17        exit(INPUT_LINE_ERROR);
18    } else {
19
20        double digit_val = 0.0;
21        int i = 0;
22        int exp_whole = 0;
```

```

23     int exp_fract = 0;
24     int pos_point = -1;
25
26     /*
27      La lettura dei caratteri inizia dall'indice chars_read-2 perche':
28      - la numerazione del buffer e' da 0 a N-1.
29      - l'ultimo carattere e' \0.
30  */
31
32     for (i = chars_read-2; i >= 0; i--) {
33         if (buffer[i] != '.' && buffer[i] != ',') {
34             exp_fract--;
35         } else {
36             pos_point = chars_read - (exp_fract * (-1)) - 2;
37             break;
38         }
39     }
40
41     i = chars_read-2;
42     while(i >= 0) {
43         if (i > pos_point && pos_point != -1) {
44
45             if (isdigit(buffer[i])) {
46                 // Per ottenere il valore intero a partire dal carattere ASCII
47                 digit_val = pow(10, exp_fract) * (digit_val + (buffer[i] - '0'))
48                         ;
49                 out = out + digit_val;
50             } else {
51                 out = 0.0;
52                 printf("Puoi inserire solo valori numerici reali!\n");
53                 printf("Applicazione terminata.\n");
54                 free(buffer);
55                 exit(NOT_REAL_NUMBER_ERROR);
56             }
57
58             exp_fract++;
59         } else if (i < pos_point || pos_point == -1) {
60
61             if (isdigit(buffer[i])) {
62                 digit_val = pow(10, exp_whole) * (digit_val + (buffer[i] - '0'))
63                         ;
64                 out = out + digit_val;
65             } else {
66                 if (i == 0 && buffer[0] == '-') {
67                     out = out * (-1);
68
69                 }
70             }
71         }
72     }
73
74     if (out == 0.0) {
75         printf("Puoi inserire solo valori numerici reali!\n");
76         printf("Applicazione terminata.\n");
77         free(buffer);
78         exit(NOT_REAL_NUMBER_ERROR);
79     }
80
81     printf("Il risultato e' %f\n", out);
82
83     free(buffer);
84
85     return 0;
86 }

```

```

67     } else {
68         out = 0.0;
69         printf("Puoi inserire solo valori numerici reali!\n");
70         printf("Applicazione terminata.\n");
71         free(buffer);
72         exit(NOT_REAL_NUMBER_ERROR);
73     }
74 }
75
76     exp_whole++;
77 }
78
79     digit_val = 0.0;
80     i--;
81 }
82 }
83
84 free(buffer);
85 return out;
86 }

```

Listing A.13: 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.14: 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
4             %d!\n", lim_inf, lim_sup);
5         printf("Applicazione terminata.\n");
6         exit(NOT_IN_RANGE_ERROR);

```

```
6  }
7 }
```

Listing A.15: La funzione *void checkScelta(int, int, int)*

```
1 void createPBS(int rows, int cols, int threads, int test, int time_calc,
2   int pbs_count) {
3
4   char pbs_path[PATH_MAX_LENGTH] = {};
5   FILE *pbs_file;
6
7   int i = 0, j = 0, q_num = 0;
8   int int_op = 0;
9   double double_op = 0.0;
10
11  char *buffer = NULL;
12  char mat_csv[PATH_MAX_LENGTH] = {};
13  char vet_csv[PATH_MAX_LENGTH] = {};
14  size_t bufsize = 0;
15  ssize_t chars_read;
16
17  if (pbs_count <= 1) {
18    system(RM_PATH" -rf ../jobs");
19    system(MKDIR_PATH" -p ../jobs");
20  }
21
22  sprintf(pbs_path, "../jobs/"NOME_PROVA"_%03d.pbs", pbs_count);
23
24  if ((pbs_file = fopen(pbs_path, "a")) == NULL) {
25    printf("Errore durante l'esecuzione!\n");
26    printf("Applicazione terminata.\n");
27    exit(FILE_OPENING_ERROR);
28  }
29
30  fprintf(pbs_file,
31    "#!/bin/bash\n"
32    "\n"
33    "#PBS -q studenti\n"
34    "#PBS -l nodes=" NODE_NUMBER ":ppn=" NODE_PROCESS
35    "\n#PBS -N " NOME_PROVA "_%03d\n"
36    "#PBS -o ../output/" NOME_PROVA "_%03d.out\n"
37    "#PBS -e ../output/" NOME_PROVA "_%03d.err\n"
38    "\n",
39    pbs_count, pbs_count, pbs_count);
40
41  if (pbs_count <= 1) {
42    fprintf(pbs_file,
```

```

42         "rm -fr $PBS_O_HOME/" NOME_PROVA "/output\n"
43         "mkdir -p $PBS_O_HOME/" NOME_PROVA "/output\n\n"
44     );
45 }
46
47 fprintf(pbs_file,
48     "echo --- \n\n"
49     "PBS_O_WORKDIR=$PBS_O_HOME/" NOME_PROVA "/codice\n"
50     "\n"
51     "echo PBS: la directory di lavoro e' $PBS_O_WORKDIR\n\n"
52     "export OMP_NUM_THREADS=%d\n"
53     "export PSC_OMP_AFFINITY=TRUE\n\n"
54     "echo PBS: Compilazione in esecuzione...\n"
55     "gcc -fopenmp -lgomp -o $PBS_O_WORKDIR/" NOME_PROVA "_%03d -lm
56         $PBS_O_WORKDIR/" NOME_PROVA ".c\n"
57     "echo PBS: Compilazione completata.\n"
58     "\n"
59     "echo 'PBS: Job in esecuzione su un pool di %d threads...'\n"
60     "echo '>>>'\n",
61 threads, pbs_count, threads);
62
63 if (time_calc == OK_TIME_CALC) {
64     fprintf(pbs_file,
65         "for i in $(seq 10)\n"
66         "do\n"
67         "\t"
68     );
69 }
70
71 fprintf(pbs_file, "$PBS_O_WORKDIR/" NOME_PROVA "_%03d %d %d %d %d %d",
72     pbs_count, rows, cols, threads, test, time_calc);
73
74 /*
75  Nella suite di testing progettata, se il numero di elementi della
76  matrice e' minore o uguale a OP_MAX_QUANTITY, allora il valore di
77  ogni elemento deve essere specificato dall'utente.
78 */
79
80 q_num = rows * cols;
81
82 switch(test) {
83     case MULTIPLICATION_INPUT_TEST: {
84         if (q_num <= OP_MAX_QUANTITY) {
85             printf("Inserimento degli elementi della matrice.\n");
86             for (i = 1; i <= rows; i++) {
87                 for (j = 1; j <= cols; j++) {

```

```

87         printf("Inserisci il %do elemento sulla %da riga:\n", j, i);
88         double_op = getNumberFromInput();
89         fprintf(pbs_file, " %f", double_op);
90     }
91 }
92
93 printf("Inserimento degli elementi del vettore.\n");
94 for (j = 1; j <= cols; j++) {
95     printf("Inserisci il %do elemento:\n", j);
96     double_op = getNumberFromInput();
97     fprintf(pbs_file, " %f", double_op);
98 }
99 }
100 break;
101 }
102 case MULTIPLICATION_CSV_TEST: {
103     printf("Inserimento degli elementi della matrice.\n");
104     printf("Inserisci il percorso del file .csv:\n");
105     chars_read = getline(&buffer, &bufsize, stdin);
106     printf("\n");
107
108     if (chars_read < 0 || chars_read > PATH_MAX_LENGTH) {
109         printf("Errore nella lettura dell'input!");
110         printf("Applicazione terminata.\n");
111         free(buffer);
112         exit(INPUT_LINE_ERROR);
113     }
114
115     for (i = 0; i < chars_read; i++) {
116         if (buffer[i] != 10) {
117             mat_csv[i] = buffer[i];
118         }
119     }
120
121     printf("Inserimento degli elementi del vettore.\n");
122     printf("Inserisci il percorso del file .csv:\n");
123     chars_read = getline(&buffer, &bufsize, stdin);
124     printf("\n");
125
126     if (chars_read < 0 || chars_read > PATH_MAX_LENGTH) {
127         printf("Errore nella lettura dell'input!");
128         printf("Applicazione terminata.\n");
129         free(buffer);
130         exit(INPUT_LINE_ERROR);
131     }
132 }
```

```

133     for (i = 0; i < chars_read; i++) {
134         if (buffer[i] != 10) {
135             vet_csv[i] = buffer[i];
136         }
137     }
138
139     fprintf(pbs_file, " %s %s", mat_csv, vet_csv);
140
141     free(buffer);
142     break;
143 }
144 case MULTIPLICATION_SINGLE_NUMBER_TEST:
145 {
146     printf("Inserimento degli elementi del vettore.\n");
147     printf("Inserisci l'unico valore per tutti gli elementi:\n");
148     int_op = getIntegerFromInput();
149     fprintf(pbs_file, " %d", int_op);
150     break;
151 }
152 default:
153     break;
154 }
155
156 if (time_calc == OK_TIME_CALC) {
157     fprintf(pbs_file, "\ndone\n");
158 } else {
159     fprintf(pbs_file, "\n");
160 }
161
162 fprintf(pbs_file,
163         "echo >>>'\n"
164         "echo PBS: Job completato.\n"
165         "echo --- \n"
166 );
167
168 fclose(pbs_file);
169 printf("%s creato con successo!\n\n", pbs_path);
170 }

```

Listing A.16: La funzione *void createPBS(int, int, int, int, int, int)*

A.3 menu.c

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

Listing A.17: Intestazione del file *menu.c*

```
1 int rows = 0, cols = 0;
2 int test = MULTIPLICATION_INPUT_TEST;
3 int i = 0, pbs_count = 1;
```

Listing A.18: 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.19: Introduzione

```
1 /*
2   Nell'eseguire la suite di testing, il programma fornira':
3   - il risultato prodotto matrice-vettore;
4   - il tempo di esecuzione relativo all'esecuzione del prodotto.
5 */
6
7 do {
8
9   printf("Scegli un test da eseguire: \n");
10  printf("%d. \t Prodotto matrice-vettore con scelta degli operandi.\n",
11        MULTIPLICATION_INPUT_TEST);
12  printf("%d. \t Prodotto matrice-vettore con input da file .csv.\n",
13        MULTIPLICATION_CSV_TEST);
14  printf("%d. \t Prodotto di una matrice per un vettore unitario.\n",
15        MULTIPLICATION_ONE_TEST);
16  printf("%d. \t Prodotto di una matrice per un vettore di numeri uguali
17        .\n", MULTIPLICATION_SINGLE_NUMBER_TEST);
18  printf("%d. \t Prodotto di una matrice per un autovettore.\n",
19        MULTIPLICATION_EIGENVECTOR_TEST);
```

```

15     printf("%d. \t Chiudere la suite di testing.\n\n", EXIT_TEST);
16     test = getIntegerFromInput();
17     checkScelta(test, MULTIPLICATION_INPUT_TEST, EXIT_TEST);

```

Listing A.20: Applicazione della suite di testing e scelta del caso di test

```

1     printf("Inserisci la quantita' di righe della matrice:\n");
2     rows = getIntegerFromInput();
3
4     if (rows < 1) {
5         printf("Devi inserire almeno una riga!\n");
6         printf("Applicazione terminata.\n");
7         exit(MATRIX_DIMENSION_ERROR);
8     }
9
10    printf("Inserisci la quantita' di colonne della matrice:\n");
11    cols = getIntegerFromInput();
12
13    if (cols < 1) {
14        printf("Devi inserire almeno una colonna!\n");
15        printf("Applicazione terminata.\n");
16        exit(MATRIX_DIMENSION_ERROR);
17    }

```

Listing A.21: Scelta delle dimensioni della matrice e del vettore

```

1     case MULTIPLICATION_INPUT_TEST:
2     case MULTIPLICATION_CSV_TEST:
3     {
4         createPBS(rows, cols, 1, test, OK_TIME_CALC, pbs_count++);
5         createPBS(rows, cols, 2, test, OK_TIME_CALC, pbs_count++);
6         createPBS(rows, cols, 4, test, OK_TIME_CALC, pbs_count++);
7         createPBS(rows, cols, 7, test, OK_TIME_CALC, pbs_count++);
8         createPBS(rows, cols, 8, test, OK_TIME_CALC, pbs_count++);
9
10        break;
11    }

```

Listing A.22: Applicazione dei casi di test

```

1     case MULTIPLICATION_ONE_TEST:
2     case MULTIPLICATION_SINGLE_NUMBER_TEST:
3     case MULTIPLICATION_EIGENVECTOR_TEST:
4     {
5
6         for (i = OP_MIN_EXP_TEST; i <= OP_MAX_EXP_TEST; i++) {
7
8             rows = cols = pow(10, i);

```

```
9
10    createPBS(rows, cols, 1, test, OK_TIME_CALC, pbs_count++);
11    createPBS(rows, cols, 2, test, OK_TIME_CALC, pbs_count++);
12    createPBS(rows, cols, 4, test, OK_TIME_CALC, pbs_count++);
13    createPBS(rows, cols, 7, test, OK_TIME_CALC, pbs_count++);
14    createPBS(rows, cols, 8, test, OK_TIME_CALC, pbs_count++);
15}
16
17    break;
18}
```

Listing A.23: Applicazione dei casi di test (cont.)

```
1 printf("Applicazione terminata.\n");
2 return 0;
```

Listing A.24: Uscita dall'applicativo

A.4 prova2.c

```
1 /*
2
3  prova2.c
4  di Mario Gabriele Carofano
5  e Francesco Noviello
6
7 */
8 // LIBRERIE
9
10 #include "./libraries/auxfunc.h"
11 #include <omp.h>
```

Listing A.25: Intestazione del file *prova2.c*

```
1 int test = MULTIPLICATION_INPUT_TEST, time_calc = NO_TIME_CALC;
2 int rows = 0, cols = 0, threads = 0;
3 int q_num = 0;
4
5 int i = 0, j = 0, k = 0;
6 double **mat, *vet;
7 double *multiplication;
8
9 struct timeval t;
10 double t_start = 0.0, t_end = 0.0, t_tot = 0.0;
11
12 int int_rand = 0;
13 double double_rand = 0.0;
14
15 srand(time(NULL));
```

Listing A.26: Inizializzazione dell'ambiente di lavoro

```
1 if (argc < 5) {
2     printf("Errore nella lettura degli argomenti di input!\n\n");
3     printf("Esecuzione terminata.\n");
4     exit(NOT_ENOUGH_ARGS_ERROR);
5 }
6
7 /*
8  --- int argToInt(char *arg) ---
9  Si utilizza la funzione 'argToInt' definita in 'auxfunc.h'
10 per leggere gli argomenti contenuti nel vettore di
11 stringhe 'argv[]' e convertirli in valori interi.
12 */
13
```

```

14 rows = argToInt(argv[1]);
15 cols = argToInt(argv[2]);
16
17 threads = argToInt(argv[3]);
18 if (threads > rows)
19     threads = rows;
20 omp_set_num_threads(threads);
21
22 test = argToInt(argv[4]);
23 time_calc = argToInt(argv[5]);

```

Listing A.27: Lettura dei dati

```

1 mat = (double**) calloc(rows, sizeof(double*));
2 for (i = 0; i < rows; i++) {
3     mat[i] = (double*) calloc(cols, sizeof(double));
4 }
5
6 vet = (double*) calloc(cols, sizeof(double));
7
8 multiplication = (double*) calloc(rows, sizeof(double));

```

Listing A.28: Allocazione della memoria

```

1 case MULTIPLICATION_INPUT_TEST:
2 {
3     q_num = rows * cols;
4     k = 1;
5     if (q_num <= OP_MAX_QUANTITY) {
6
7         for (i = 0; i < rows; i++) {
8             for (j = 0; j < cols; j++) {
9                 mat[i][j] = argToDouble(argv[k+5]);
10                k++;
11            }
12        }
13
14        for (j = 0; j < cols; j++) {
15            vet[j] = argToDouble(argv[k+5]);
16            k++;
17        }
18
19    } else {
20
21        for (i = 0; i < rows; i++) {
22            for (j = 0; j < cols; j++) {
23                double_rand = (double)rand();
24                int_rand = (int)rand();

```

```

25
26     // Si genera un numero casuale reale compreso tra 0 e 100
27     mat[i][j] = (double_rand / RAND_MAX) * OP_MAX_VALUE;
28
29     // Si ha il 33% di possibilità che mat[i][j] < 0
30     if (int_rand % 3 == 0) {
31         mat[i][j] = mat[i][j] * (-1);
32     }
33 }
34 }
35
36 for (j = 0; j < cols; j++) {
37     double_rand = (double)rand();
38     int_rand = (int)rand();
39
40     vet[j] = (double_rand / RAND_MAX) * OP_MAX_VALUE;
41
42     if (int_rand % 3 == 0) {
43         vet[j] = vet[j] * (-1);
44     }
45 }
46
47 }
48
49     break;
50 }
```

Listing A.29: Applicazione del caso di test *MULTIPLICATION_INPUT_TEST*

```

1 case MULTIPLICATION_CSV_TEST:
2 {
3
4     /*
5      In questo caso di test, i valori per la matrice ed il vettore
6      sono caricati da un file .csv i cui percorsi sono specificati
7      alle posizioni 'argv[6]' e 'argv[7]'.
8     */
9
10    getMatrixFromCSV(argv[6], mat, rows, cols);
11    getVectorFromCSV(argv[7], vet, cols);
12    break;
13 }
```

Listing A.30: Applicazione del caso di test *MULTIPLICATION_CSV_TEST*

```

1 case MULTIPLICATION_ONE_TEST:
2 {
3 }
```

```

4  /*
5   Il vettore da moltiplicare e' costituito di soli 1.
6
7   Il test termina con successo se il vettore finale
8   e' costituito dal valore della somma dei valori di
9   ogni singola riga.
10 */
11
12 for (i = 0; i < rows; i++) {
13     for (j = 0; j < cols; j++) {
14         double_rand = (double)rand();
15         int_rand = (int)rand();
16
17         mat[i][j] = (double_rand / RAND_MAX) * OP_MAX_VALUE;
18
19         if (int_rand % 3 == 0) {
20             mat[i][j] = mat[i][j] * (-1);
21         }
22     }
23 }
24
25 for (j = 0; j < cols; j++) {
26     vet[j] = 1;
27 }
28
29 break;
30 }

```

Listing A.31: Applicazione del caso di test *MULTIPLICATION_ONE_TEST*

```

1 case MULTIPLICATION_SINGLE_NUMBER_TEST:
2 {
3
4 /*
5   Ad ogni posizione del vettore degli operandi e'
6   assegnato lo stesso valore reale, passato come
7   argomento 'argv[6]' al programma.
8
9   Il test termina con successo se il vettore finale
10  e' costituito dal valore della somma dei valori di
11  ogni singola riga moltiplicato proprio per il
12  valore 'argv[6]'.
13 */
14
15 for (i = 0; i < rows; i++) {
16     for (j = 0; j < cols; j++) {
17         double_rand = (double)rand();

```

```

18     int_rand = (int)rand();
19
20     mat[i][j] = (double_rand / RAND_MAX) * OP_MAX_VALUE;
21
22     if (int_rand % 3 == 0) {
23         mat[i][j] = mat[i][j] * (-1);
24     }
25 }
26 }
27
28 for (j = 0; j < cols; j++) {
29     vet[j] = argToDouble(argv[6]);
30 }
31
32 break;
33 }
```

Listing A.32: Applicazione del caso di test *MULTIPLICATION_SINGLE_NUMBER_TEST*

```

1 case MULTIPLICATION_EIGENVECTOR_TEST:
2 {
3     // Da implementare.
4     // Si veda la sezione "Futuri sviluppi" nella documentazione.
5     break;
6 }
```

Listing A.33: Applicazione del caso di test *MULTIPLICATION_EIGENVECTOR_TEST*

```

1 if (time_calc == OK_TIME_CALC) {
2     gettimeofday(&t, NULL);
3     t_start = t.tv_sec + (t.tv_usec / TIME_PRECISION);
4 }
```

Listing A.34: Inizio del calcolo dei tempi di esecuzione

```

1 if (mat && vet && multiplication) {
2     #pragma omp parallel for default(none) shared(rows, cols, mat, vet,
3         multiplication) private (i,j)
4     for (i = 0; i < rows; i++) {
5         for (j = 0; j < cols; j++) {
6             multiplication[i] = multiplication[i] + mat[i][j] * vet[j];
7         }
8     }
9 }
```

Listing A.35: Calcolo del prodotto matrice-vettore

```

1 if (time_calc == OK_TIME_CALC) {
2     gettimeofday(&t, NULL);
```

```

3  t_end = t.tv_sec + (t.tv_usec / TIME_PRECISION);
4
5  // Si calcola la distanza di tempo tra l'istante iniziale e quello
6  // finale.
7  t_tot = t_end - t_start;
8
9  printf("\nCalcolo del prodotto matrice-vettore terminato in %e sec\n",
10   t_tot);
11  writeTimeCSV(CSV_TIME_PATH"/"NOME_PROVA"_time.csv", rows, cols,
12   threads, test, t_tot);
13 }
```

Listing A.36: Salvataggio del calcolo dei tempi di esecuzione

```

1  printf("Risultato:\n");
2  for (i = 0; i < rows; i++) {
3      printf("Riga %d -> %f\n", i, multiplication[i]);
4 }
```

Listing A.37: Stampa dell'output

```

1  freeMatrix(mat, rows);
2
3  free(vet);
4  free(multiplication);
5  printf("\nEsecuzione terminata.\n");
6
7  return 0;
```

Listing A.38: Terminazione dell'esecuzione

A.5 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 ROWS_FIELD 0  
11 #define COLS_FIELD 1  
12 #define THREADS_FIELD 2  
13 #define TEST_FIELD 3  
14 #define TIME_FIELD 4  
15  
16 #define BURST_SIZE 10  
17 //  LIBRERIE E ALTRE FUNZIONI  
18  
19 #include "./libraries/auxfunc.h"
```

Listing A.39: Intestazione del file *media-csv.c*

```
1 void calcola_media(double **mat, int rows) {  
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][TIME_FIELD];  
11         }  
12  
13         avg /= BURST_SIZE;  
14         writeTimeCSV(  
15             CSV_TIME_PATH"/" NOME_PROVA"_time_avg.csv",  
16             (int) mat[k][ROWS_FIELD],  
17             (int) mat[k][COLS_FIELD],  
18             (int) mat[k][THREADS_FIELD],  
19             (int) mat[k][TEST_FIELD],  
20             avg  
21         );  
22     }
```

23 }

Listing A.40: La funzione *void calcola_media(double **, int)*

```
1 int main(int argc, char **argv) {
2
3     int rows = 0, cols = 0, i = 0;
4     char *csv_path = NULL;
5     double **mat = NULL;
6
7     csv_path = argv[1];
8     rows = getRowsFromCSV(csv_path);
9     cols = getColsFromCSV(csv_path);
10
11    mat = (double**) calloc(rows, sizeof(double*));
12    for (i = 0; i < rows; i++) {
13        mat[i] = (double*) calloc(cols, sizeof(double));
14    }
15
16    getMatrixFromCSV(csv_path, mat, rows, cols);
17    calcola_media(mat, rows);
18    freeMatrix(mat, rows);
19
20    return 0;
21
22 }
```

Listing A.41: Calcolo della media dei tempi di esecuzione

Sitografia

- [1] Open MP, *La libreria omp.h (v. 200505)*,
<https://www.openmp.org/wp-content/uploads/spec25.pdf>.
- [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 time.h*, <https://cplusplus.com/reference/ctime>.
- [9] The Open Group, *La libreria sys/time.h*,
<https://pubs.opengroup.org/onlinepubs/7908799/xsh/systime.h.html>.
- [10] C++ Reference, *La libreria ctype.h*, <https://cplusplus.com/reference/cctype>.
- [11] C++ Reference, *La libreria math.h*, <https://cplusplus.com/reference/cmath>.