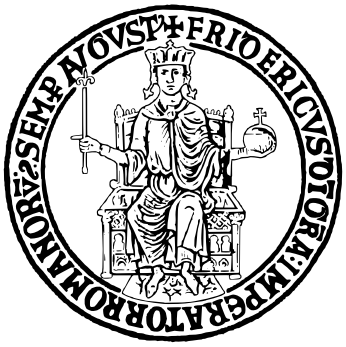
Università degli Studi di Napoli Federico II



Scuola Politecnica e delle Scienze di Base
Dipartimento di Ingegneria Elettrica e
Tecnologie dell'Informazione

Corso di Laurea Magistrale in Informatica

Parallel and Distributed Computing

Progettazione di un algoritmo per il calcolo del prodotto matrice-vettore in ambiente di calcolo parallelo su architettura MIMD a memoria condivisa

Docenti
Prof. Giuliano Laccetti
Prof.ssa Valeria Mele

Candidati
Marco Romano N97000395
Gianluca L'arco N97000393

Anno Accademico 2021 - 2022

Indice

Elenco delle figure

Elenco delle tabelle

1 Definizione e analisi del problema

L'obiettivo dell'algoritmo è eseguire il prodotto scalare tra una matrice $A \in \mathbb{R}^{n \times m}$ ed un vettore $\vec{x} \in \mathbb{R}^m$ in ambiente parallelo su un architettura MIMD a memoria condivisa. Il risultato di tale prodotto è un vettore $\vec{b} \in \mathbb{R}^n$, ovvero un vettore con dimensione pari al numero di righe n.

Condizione necessaria affinché l'algoritmo possa essere eseguito è che la dimensione del vettore x sia uguale al numero di colonne della matrice A.

L'algoritmo viene, successivamente, valutato attraverso i seguenti parametri: tempo impiegato, speed up ed efficienza.

Il linguaggio di programmazione utilizzato è il C. La gestione della parallelizzazione avviene mediante la libreria *OpenMP* (Open specifications for Multi Processing), la quale estende le capacità dei linguaggi di programmazione seriali consentendo la programmazione parallela a memoria condivisa.

Differentemente da quanto accade con la memoria distribuita, la memoria condivisa consente di utilizzare i dati senza la necessità di effettuare scambi tra i vari processori; bisogna, tuttavia, sincronizzare gli accessi in memoria.

2 Input e Output

Il programma prevede che vengano forniti i seguenti parametri in input:

1. n: numero di righe della matrice A,
2. m: numero di colonne della matrice A,
3. t: numero di thread da generare per l'esecuzione parallela.

Tali parametri vengono forniti al cluster mediante il file *input.pbs*. L'output è contenuto nel file *output.out* con il seguente formato:

```
> Generated Matrix
 [0.405286] [3.001648] [2.980895] [2.685781] [3.302408]
 [3.780480] [3.298898] [0.574053] [2.690817] [2.714469]
 [1.419263] [2.783746] [2.336344] [1.995605] [2.799463]
 [4.340953] [0.463424] [4.540696] [3.418531] [0.621967]
 [3.503983] [0.871303] [4.760125] [0.528123] [2.198096]

> Generated Vector
[3.934904]  [4.977569]  [1.114841]  [3.331706]  [2.018799]

> Product Vector
[35.474028]  [46.381277]  [34.345928]  [37.095276]  [29.628655]

> Overall time: 0.000000
```

Eventuali errori dell'esecuzione del programma sono presenti nel file *error.err*

3 Indicatori di errore

Gli indicatori di errore, illustrati anche nella funzione *help*, sono descritti nella Tabella 1:

Codice	**Errore**	**Descrizione**
400	ERR_ARGC	Invalid number of arguments
401	ERR_NO_ROWS	Mandatory argument [-r --rows] not provided
402	ERR_NO_COLUMNS	Mandatory argument [-c --columns] not provided
403	ERR_NO_THREADS	Mandatory argument [-t --threads] not provided
404	ERR_INVALD_ROWS	Invalid number of rows provided
405	ERR_INVALD_COLUMNS	Invalid number of columns provided
406	ERR_INVALD_THREADS	Invalid number of threads provided
407	ERR_MEMORY	Unable to allocate memory

Tabella 1: Indicatori di errori

4 Subroutine

In questa sezione vengono elencate e descritte tutte le funzioni adoperate nel software, incluse quelle di OpenMP.

4.1 Funzioni OpenMP

```
void omp_set_num_threads(int num_threads)
```

Descrizione: Imposta il numero di thread da impiegare.

Input: *num_threads* numero di thread.

4.2 Funzioni codificate

```
void help(char* program_name)
```

Descrizione: Stampa a video l'help del programma.

Input: *program_name* nome del programma.

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

Descrizione: Stampa a video l'help del programma.

Input: *argc* numero di argomenti passati in ingresso al programma, *argv* argomenti passati in ingresso al programma.

Output: Codice errore/successo.

```
double** generate_random_matrix(int rows, int columns, double lower, double upper)
```

Descrizione: Genera una matrice di $rows \times columns$ elementi reali pseudo randomici.

Input: *rows* numero di righe della matrice, *columns* numero di colonne della matrice, *lower* limite inferiore del valore degli elementi, *upper* limite superiore del valore degli elementi.

Output: Matrice generata.

```
double* generate_random_vector(int dimension, double lower, double upper)
```

Descrizione: Genera un vettore di *dimension* elementi reali pseudo-randomici.

Input: *dimension* dimensione del vettore, *lower* limite inferiore del valore degli elementi, *upper* limite superiore del valore degli elementi.

Output: Vettore generato.

```
void print_vector(double* vector, int dimension)
```

Descrizione: Effettua la stampa di un vettore.

Input: *vector* vettore da stampare, *dimension* dimensione del vettore.

```
void print_matrix(double** matrix, int rows, int columns)
```

Descrizione: Effettua la stampa di una matrice.

Input: *matrix* matrice da stampare, *rows* numero di righe della matrice.

```
double* product(int rows, int columns, double** matrix, double* restrict vector)
```

Descrizione: Effettua Il prodotto tra la matrice e il vettore.

Input: *rows* numero di righe della matrice, *columns* numero di colonne della matrice, *matrix* matrice da impiegare nel calcolo del prodotto, *vector* vettore da impiegare nel calcolo del prodotto.

Output: Vettore di dimensione *rows* risultante dal prodotto

5 Descrizione dell'algoritmo

In questo paragrafo viene fornita una panoramica generale dell'algoritmo.

5.1 Controllo dell'input

Nella prima fase, il programma effettua controlli sull'input al fine di verificare che i valori forniti siano corretti. In caso contrario, viene fornito un messaggio di errore e il programma termina l'esecuzione.

5.2 Inizializzazione delle strutture

Il programma alloca la matrice A e il vettore $\vec{x}$; dopodiché, tali strutture vengono inizializzate con valori random.

5.3 Prodotto Matrice - Vettore

Si vuole realizzare il seguente prodotto:

$$A\vec{x} = \vec{b} \tag{1}$$

dove $A \in \mathbb{R}^{n \times m}$, $\vec{x} \in \mathbb{R}^m$ e $\vec{b} \in \mathbb{R}^n$.

Tale operazione è stata implementata mediante la libreria OpenMP, la quale consente di eseguire, in modo parallelo, le istruzioni contenute all'interno di un blocco di codice, distribuendo equamente il carico delle operazioni ai vari thread.

Il blocco di codice parallellizzato è il seguente:

```
for(int i = 0; i < rows; i++)
    for(int j = 0; j < columns; j++)
      product[i] += matrix[i][j]*vector[j];
```

OpenMP utilizza delle direttive, contrassegnate dal simbolo #, per effettuare la parallelizzazione di blocchi di codice. Nel nostro caso specifico, la direttiva utilizzata è la seguente:

```
#pragma omp parallel for default(none) shared(rows, columns, matrix, vector,
    product)
```

Tale direttiva presenta le seguenti clausole:

1. *default*: determina in modo esplicito gli attributi di condivisione dei dati delle variabili a cui si fa riferimento in un costrutto e che altrimenti sarebbe implicitamente determinato;

2. *shared*: specifica che una o più variabili devono essere condivise tra tutti i thread;

3. *private*: specifica che ogni thread deve avere la propria istanza di una variabile.

6 Analisi dei tempi e delle prestazioni

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

1. Tempo medio impiegato: per ogni esperimento sono state effettuate 5 prove ed è stata, successivamente, considerata la media aritmetica dei tempi di ciascuna prova;

2. Speed Up: misura la riduzione del tempo di esecuzione rispetto all'algoritmo su 1 processore, $S(p) = \frac{T(1)}{T(p)}$ dove $T(1)$ rappresenta il tempo impiegato dall'algoritmo con singolo processore, mentre $T(p)$ rappresenta il tempo impiegato dall'algoritmo con p processori. Si osservi che $S(p)_{ideale} = p$;

3. Efficienza: misura quanto l'algoritmo sfrutta il parallelismo del calcolatore, $E(p) = \frac{S(p)}{p}$. Si osservi che $E(p)_{ideale} = 1$.

Per stimare il tempo di esecuzione dell'algoritmo è stata utilizzata la funzione della libreria *sys/time.h*:

```
int gettimeofday(struct timeval *tv, struct timezone *tz)
```

Tale funzione permette di calcolare il tempo trascorso in un blocco di codice in C recuperando l'ora corrente con una precisione di microsecondi. Il tempo di esecuzione

viene calcolato con la seguente formula:

$$time = t_1 - t_0 \tag{2}$$

6.1 Dati raccolti

Di seguito vengono riportati, in formato tabellare, i tempi ottenuti dai vari test effettuati. Il numero di test totali per ogni tupla (r, c, t) è di 5, dove r rappresenta il numero di righe della matrice, c il numero di colonne, oltre alla dimensione del vettore e, infine, t il numero di thread impiegati. Per ogni tupla (r, c, t) è riportato il tempo medio rilevato.

	Matrice $10^2 \times 10^2$ **- Vettore** 10^2			
Threads →	**T1**	**T2**	**T3**	**T4**
Test N°1	0,000120	0,002822	0,004946	0,003830
Test N°2	0,000120	0,002949	0,004097	0,004710
Test N°3	0,000118	0,001858	0,002621	0,005794
Test N°4	0,000118	0,002972	0,003780	0,004812
Test N°5	0,000119	0,001924	0,002710	0,003954
Media	**0,000119**	**0,002505**	**0,003631**	**0,004620**
Threads →	**T5**	**T6**	**T7**	**T8**
Test N°1	0,005733	0,005786	0,005693	0,007843
Test N°2	0,004833	0,004806	0,005785	0,006890
Test N°3	0,005804	0,005757	0,005915	0,006182
Test N°4	0,003854	0,005903	0,005018	0,008121
Test N°5	0,004019	0,004729	0,007044	0,005182
Media	**0,004849**	**0,005396**	**0,005891**	**0,006844**

Tabella 2: Test con Matrice $10^2 \times 10^2$ - Vettore 10^2

	Matrice $10^2 \times 10^3$ **- Vettore** 10^3			
Threads $\rightarrow$	**T1**	**T2**	**T3**	**T4**
Test N°1	0,001124	0,003617	0,003726	0,004576
Test N°2	0,001122	0,001222	0,002767	0,003695
Test N°3	0,001090	0,001255	0,003236	0,004217
Test N°4	0,000927	0,004075	0,002795	0,002836
Test N°5	0,001087	0,002745	0,004161	0,003733
Media	**0,001070**	**0,002583**	**0,003337**	**0,003811**
Threads $\rightarrow$	**T5**	**T6**	**T7**	**T8**
Test N°1	0,003724	0,004191	0,003697	0,009208
Test N°2	0,004896	0,005729	0,004898	0,007358
Test N°3	0,004039	0,004778	0,006710	0,011123
Test N°4	0,003975	0,004926	0,003700	0,006600
Test N°5	0,003656	0,003748	0,004641	0,006552
Media	**0,004058**	**0,004674**	**0,004729**	**0,008168**

Tabella 3: Test con Matrice $10^2 \times 10^3$ - Vettore 10^3

	Matrice $10^2 \times 10^4$ **- Vettore** 10^4			
Threads $\rightarrow$	**T1**	**T2**	**T3**	**T4**
Test N°1	0,009509	0,006391	0,007375	0,006103
Test N°2	0,011070	0,008544	0,005521	0,006579
Test N°3	0,011346	0,008632	0,005872	0,008569
Test N°4	0,011090	0,006406	0,005572	0,006483
Test N°5	0,010839	0,006015	0,006184	0,003970
Media	**0,010771**	**0,007198**	**0,006105**	**0,006341**
Threads $\rightarrow$	**T5**	**T6**	**T7**	**T8**
Test N°1	0,006157	0,005246	0,010357	0,010846
Test N°2	0,007716	0,004667	0,010484	0,010866
Test N°3	0,003585	0,007087	0,006873	0,006732
Test N°4	0,004880	0,007345	0,008699	0,008013
Test N°5	0,005386	0,008885	0,008649	0,006270
Media	**0,005545**	**0,006646**	**0,009012**	**0,008545**

Tabella 4: Test con Matrice $10^2 \times 10^4$ - Vettore 10^4

	Matrice $10^3 \times 10^2$ **- Vettore** 10^2			
Threads $\rightarrow$	**T1**	**T2**	**T3**	**T4**
Test N°1	0,001112	0,001416	0,001784	0,003592
Test N°2	0,001113	0,003245	0,003195	0,002419
Test N°3	0,001112	0,002440	0,002950	0,003672
Test N°4	0,001112	0,003169	0,002682	0,003727
Test N°5	0,001117	0,002778	0,004012	0,003328
Media	**0,001113**	**0,002610**	**0,002925**	**0,003348**
Threads $\rightarrow$	**T5**	**T6**	**T7**	**T8**
Test N°1	0,003630	0,003652	0,010040	0,005520
Test N°2	0,005892	0,003537	0,004494	0,006558
Test N°3	0,003390	0,004486	0,005629	0,009476
Test N°4	0,003987	0,007619	0,003604	0,008081
Test N°5	0,003526	0,003922	0,007459	0,006114
Media	**0,004085**	**0,004643**	**0,006245**	**0,007150**

Tabella 5: Test con Matrice $10^3 \times 10^2$ - Vettore 10^2

	Matrice $10^3 \times 10^3$ **- Vettore** 10^3			
Threads →	**T1**	**T2**	**T3**	**T4**
Test N°1	0,010957	0,008085	0,007860	0,006852
Test N°2	0,009454	0,008573	0,005923	0,007156
Test N°3	0,010230	0,007762	0,008247	0,007112
Test N°4	0,009504	0,006063	0,006541	0,008522
Test N°5	0,009428	0,008501	0,003863	0,005478
Media	**0,009915**	**0,007797**	**0,006487**	**0,007024**
Threads →	**T5**	**T6**	**T7**	**T8**
Test N°1	0,004472	0,006307	0,006599	0,011557
Test N°2	0,005899	0,005236	0,014602	0,009421
Test N°3	0,004232	0,007130	0,005896	0,008298
Test N°4	0,002894	0,005339	0,010878	0,011605
Test N°5	0,005926	0,008091	0,005800	0,016505
Media	**0,004685**	**0,006421**	**0,008755**	**0,011477**

Tabella 6: Test con Matrice $10^3 \times 10^3$ - Vettore 10^3

	Matrice $10^3 \times 10^4$ - Vettore 10^4			
Threads →	**T1**	**T2**	**T3**	**T4**
Test N°1	0,095687	0,054037	0,039577	0,032539
Test N°2	0,095960	0,051595	0,039769	0,029695
Test N°3	0,095723	0,054887	0,038083	0,030480
Test N°4	0,095638	0,055103	0,037746	0,031343
Test N°5	0,095916	0,052714	0,038372	0,031413
Media	**0,095785**	**0,053667**	**0,038709**	**0,031094**
Threads →	**T5**	**T6**	**T7**	**T8**
Test N°1	0,022614	0,024209	0,017606	0,022156
Test N°2	0,026769	0,021646	0,019368	0,020817
Test N°3	0,026789	0,023656	0,016584	0,021248
Test N°4	0,024288	0,025136	0,023037	0,020514
Test N°5	0,027154	0,025208	0,021749	0,023790
Media	**0,025523**	**0,023971**	**0,019669**	**0,021705**

Tabella 7: Test con Matrice $10^3 \times 10^4$ - Vettore 10^4

	Matrice $10^4 \times 10^2$ **- Vettore** 10^2			
Threads $\rightarrow$	**T1**	**T2**	**T3**	**T4**
Test N°1	0,009760	0,005275	0,008890	0,006742
Test N°2	0,011216	0,007846	0,007909	0,007154
Test N°3	0,009646	0,006449	0,006809	0,008969
Test N°4	0,011214	0,005968	0,006236	0,005770
Test N°5	0,010425	0,005759	0,006141	0,005509
Media	**0,010452**	**0,006259**	**0,007197**	**0,006829**
Threads $\rightarrow$	**T5**	**T6**	**T7**	**T8**
Test N°1	0,005088	0,005998	0,007280	0,013732
Test N°2	0,005124	0,006374	0,013789	0,012508
Test N°3	0,006245	0,005192	0,006981	0,010848
Test N°4	0,005492	0,007407	0,006352	0,006727
Test N°5	0,005113	0,005857	0,005979	0,024928
Media	**0,005412**	**0,006166**	**0,008076**	**0,013749**

Tabella 8: Test con Matrice $10^4 \times 10^2$ - Vettore 10^2

	Matrice $10^4 \times 10^3$ **- Vettore** 10^3			
Threads $\rightarrow$	**T1**	**T2**	**T3**	**T4**
Test N°1	0,094601	0,052609	0,036259	0,031674
Test N°2	0,094343	0,051234	0,035949	0,031177
Test N°3	0,094652	0,063828	0,036856	0,032540
Test N°4	0,094640	0,052218	0,034898	0,031402
Test N°5	0,094336	0,053815	0,035500	0,030815
Media	**0,094514**	**0,054741**	**0,035892**	**0,031522**
Threads $\rightarrow$	**T5**	**T6**	**T7**	**T8**
Test N°1	0,024949	0,025172	0,020231	0,023741
Test N°2	0,027565	0,022427	0,019374	0,021880
Test N°3	0,024702	0,021782	0,021837	0,020510
Test N°4	0,025090	0,023627	0,018426	0,023062
Test N°5	0,022495	0,021238	0,017643	0,019445
Media	**0,024960**	**0,022849**	**0,019502**	**0,021728**

Tabella 9: Test con Matrice $10^4 \times 10^3$ - Vettore 10^3

	Matrice $10^4 \times 10^4$ **- Vettore** 10^4			
Threads →	**T1**	**T2**	**T3**	**T4**
Test N°1	0,959134	0,484256	0,329048	0,248996
Test N°2	0,959424	0,488549	0,336333	0,246797
Test N°3	0,956847	0,484239	0,325488	0,248462
Test N°4	0,956505	0,484651	0,326078	0,251696
Test N°5	0,956367	0,489183	0,328692	0,249433
Media	**0,957655**	**0,486176**	**0,329128**	**0,249077**
Threads →	**T5**	**T6**	**T7**	**T8**
Test N°1	0,202266	0,172975	0,150216	0,144854
Test N°2	0,203415	0,172340	0,155288	0,139772
Test N°3	0,203118	0,174584	0,156248	0,139584
Test N°4	0,203637	0,168499	0,150977	0,143289
Test N°5	0,203899	0,172344	0,149533	0,139668
Media	**0,203267**	**0,172148**	**0,152452**	**0,141433**

Tabella 10: Test con Matrice $10^4 \times 10^4$ - Vettore 10^4

6.2 Analisi Matrice $10^2 \times 10^2$ - Vettore 10^2

	Matrice $10^2 \times 10^2$ - Vettore 10^2		
Threads	**Tempo medio**	**Speed Up**	**Efficienza**
T1	0,000119	1,000000	1,000000
T2	0,002505	0,047505	0,023752
T3	0,003631	0,032775	0,010925
T4	0,004620	0,025758	0,006439
T5	0,004849	0,024543	0,004909
T6	0,005396	0,022053	0,003675
T7	0,005891	0,020200	0,002886
T8	0,006844	0,017389	0,002174

Tabella 11: Dati rilevati con Matrice $10^2 \times 10^2$ - Vettore 10^2

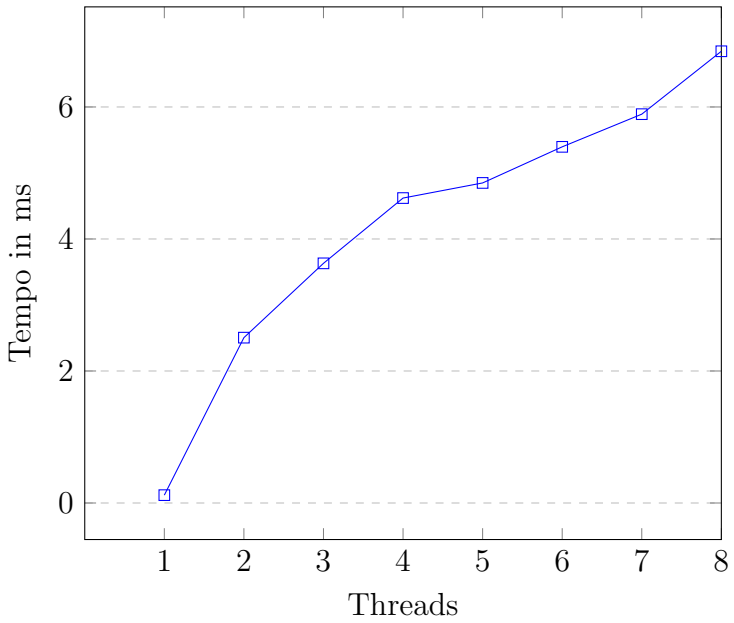


Figura 1: Grafico del tempo medio Matrice $10^2 \times 10^2$ - Vettore 10^2

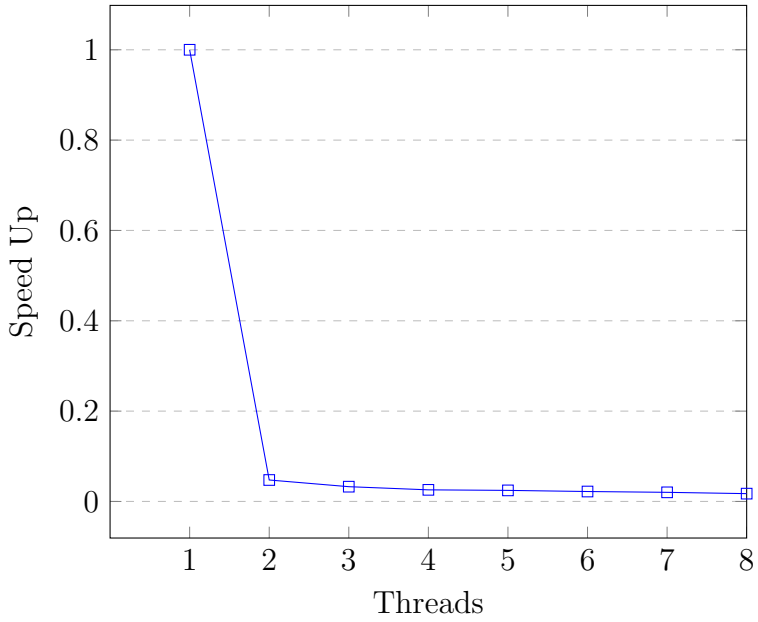


Figura 2: Grafico dello Speed Up Matrice $10^2 \times 10^2$ - Vettore 10^2

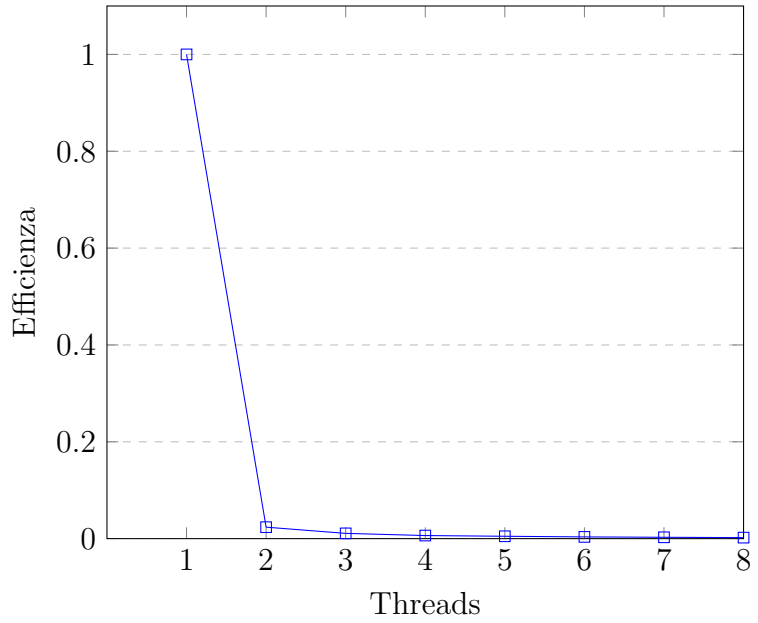


Figura 3: Grafico dell'Efficienza Matrice $10^2 \times 10^2$ - Vettore 10^2

6.3 Analisi Matrice $10^2 \times 10^3$ - Vettore 10^3

	Matrice $10^2 \times 10^3$ - Vettore 10^3		
Threads	**Tempo medio**	**Speed Up**	**Efficienza**
T1	0,001070	1,000000	1,000000
T2	0,002583	0,414279	0,207140
T3	0,003337	0,320647	0,106882
T4	0,003811	0,280737	0,070184
T5	0,004058	0,263677	0,052735
T6	0,004674	0,228906	0,038151
T7	0,004729	0,226254	0,032322
T8	0,008168	0,130996	0,016374

Tabella 12: Dati rilevati con Matrice $10^2 \times 10^3$ - Vettore 10^3

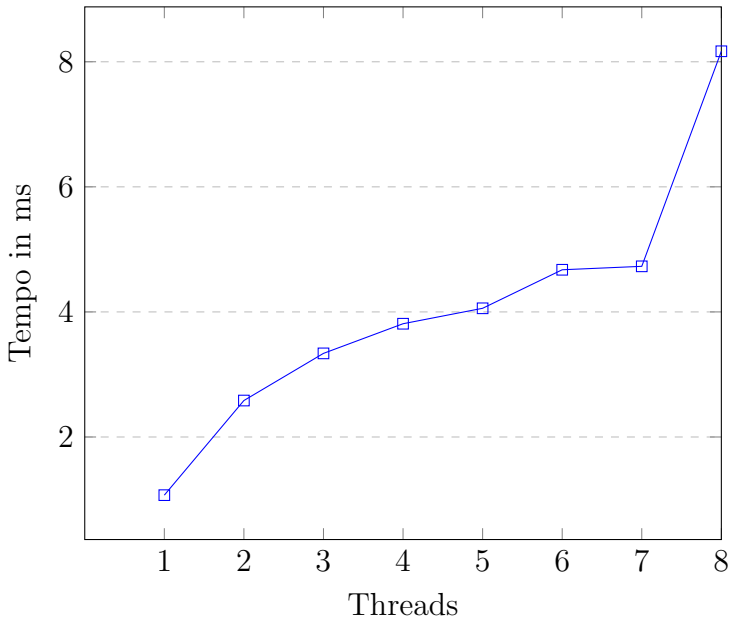


Figura 4: Grafico del tempo medio Matrice $10^2 \times 10^3$ - Vettore 10^3

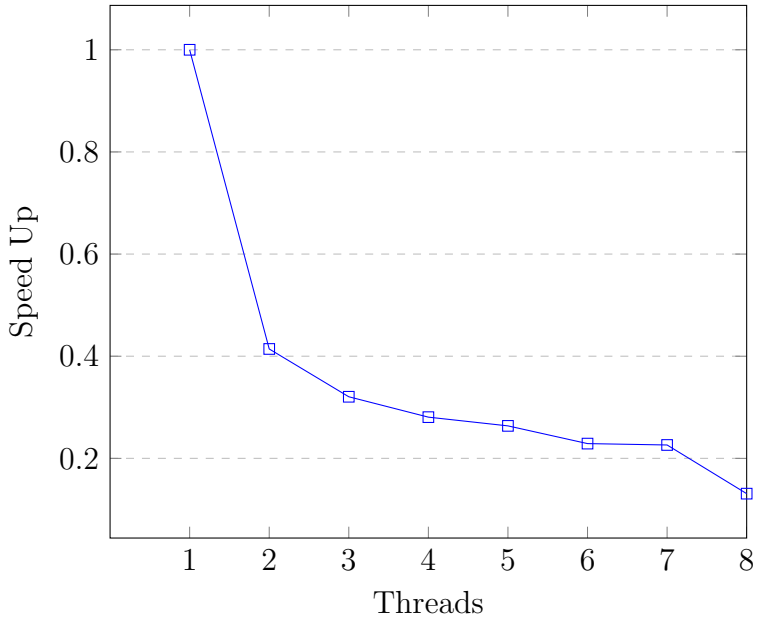


Figura 5: Grafico dello Speed Up Matrice $10^2 \times 10^2$ - Vettore 10^3

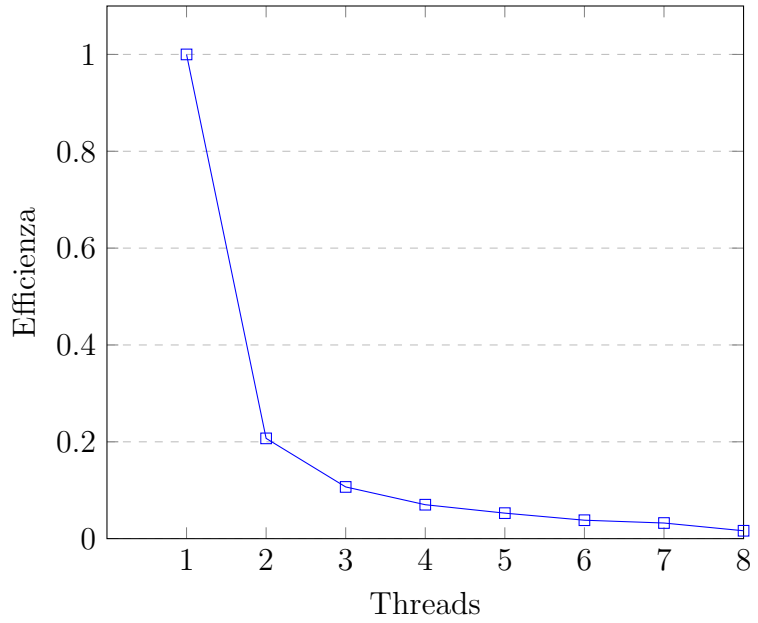


Figura 6: Grafico dell'Efficienza Matrice $10^2 \times 10^2$ - Vettore 10^3

6.4 Analisi Matrice $10^2 \times 10^4$ - Vettore 10^4

	Matrice $10^2 \times 10^4$ - Vettore 10^4		
Threads	**Tempo medio**	**Speed Up**	**Efficienza**
T1	0,010771	1,000000	1,000000
T2	0,007198	1,496443	0,748222
T3	0,006105	1,764317	0,588106
T4	0,006341	1,698650	0,424663
T5	0,005545	1,942505	0,388501
T6	0,006646	1,620644	0,270107
T7	0,009012	1,195109	0,170730
T8	0,008545	1,260421	0,157553

Tabella 13: Dati rilevati con Matrice $10^2 \times 10^4$ - Vettore 10^4

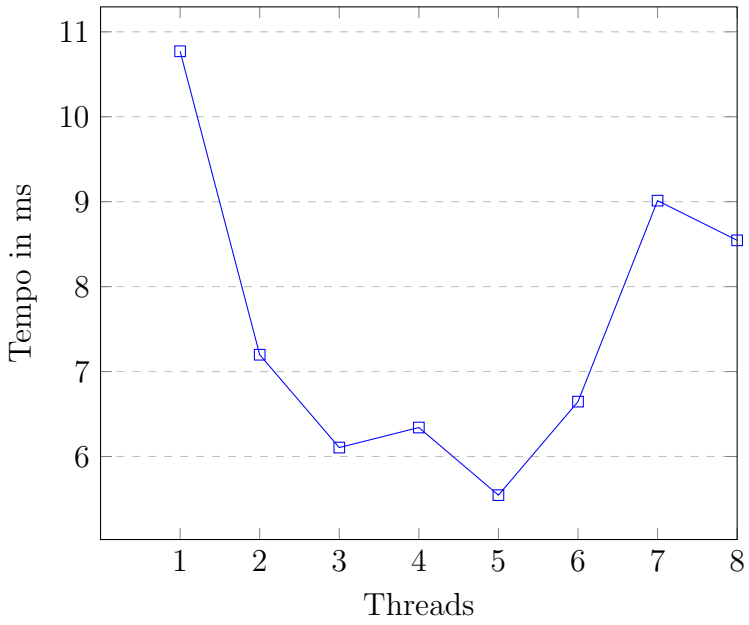


Figura 7: Grafico del tempo medio Matrice $10^2 \times 10^4$ - Vettore 10^4

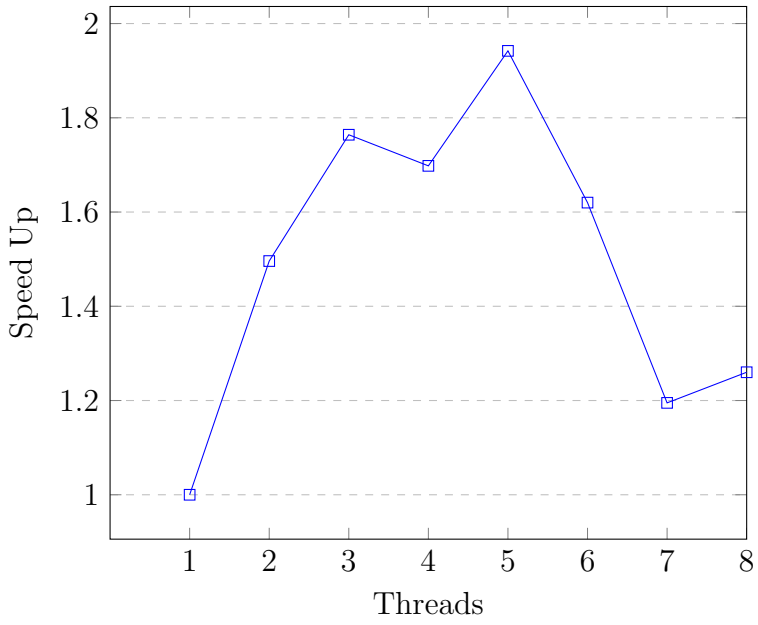


Figura 8: Grafico dello Speed Up Matrice $10^2 \times 10^4$ - Vettore 10^4

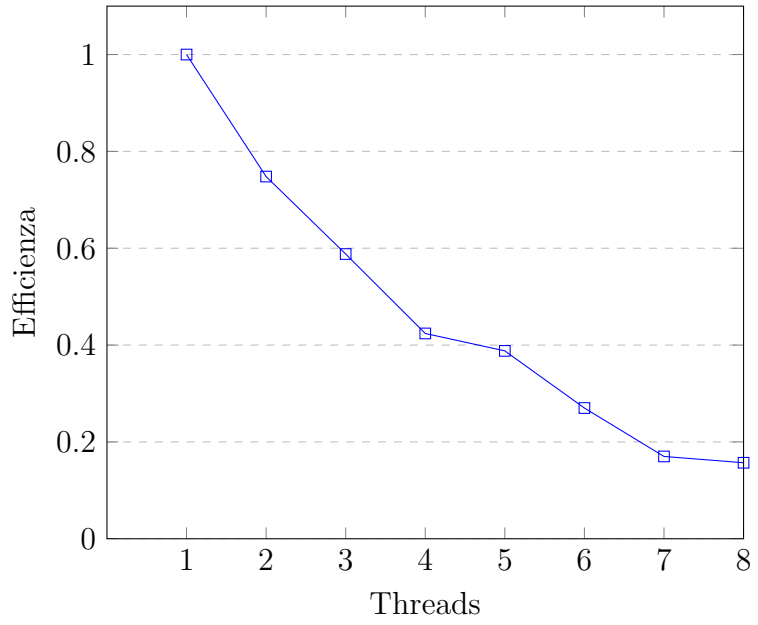


Figura 9: Grafico dell'Efficienza Matrice $10^4 \times 10^4$ - Vettore 10^4

6.5 Analisi Matrice $10^3 \times 10^2$ - Vettore 10^2

	Matrice $10^3 \times 10^2$ - Vettore 10^2		
Threads	**Tempo medio**	**Speed Up**	**Efficienza**
T1	0,001113	1,000000	1,000000
T2	0,002610	0,426579	0,213289
T3	0,002925	0,380633	0,126878
T4	0,003348	0,332537	0,083134
T5	0,004085	0,272509	0,054502
T6	0,004643	0,239748	0,039958
T7	0,006245	0,178249	0,025464
T8	0,007150	0,155697	0,019462

Tabella 14: Dati rilevati con Matrice $10^3 \times 10^2$ - Vettore 10^2

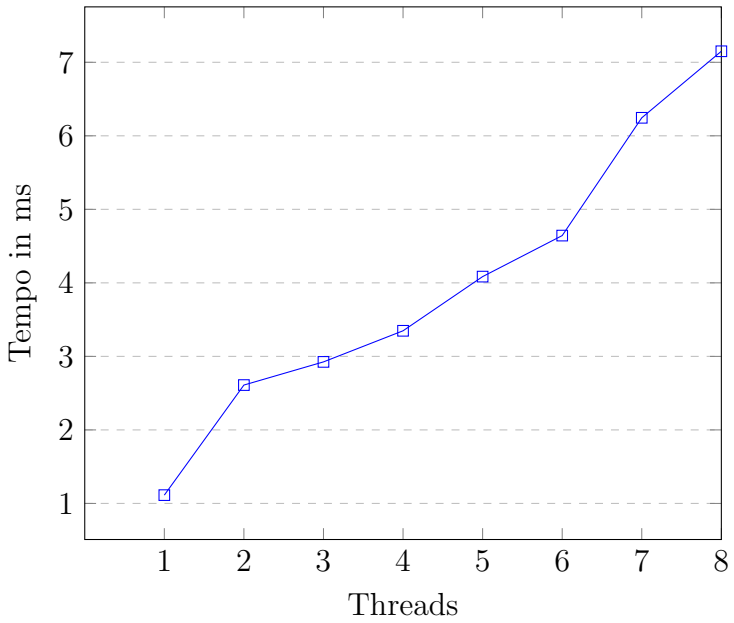


Figura 10: Grafico del tempo medio Matrice $10^3 \times 10^2$ - Vettore 10^2

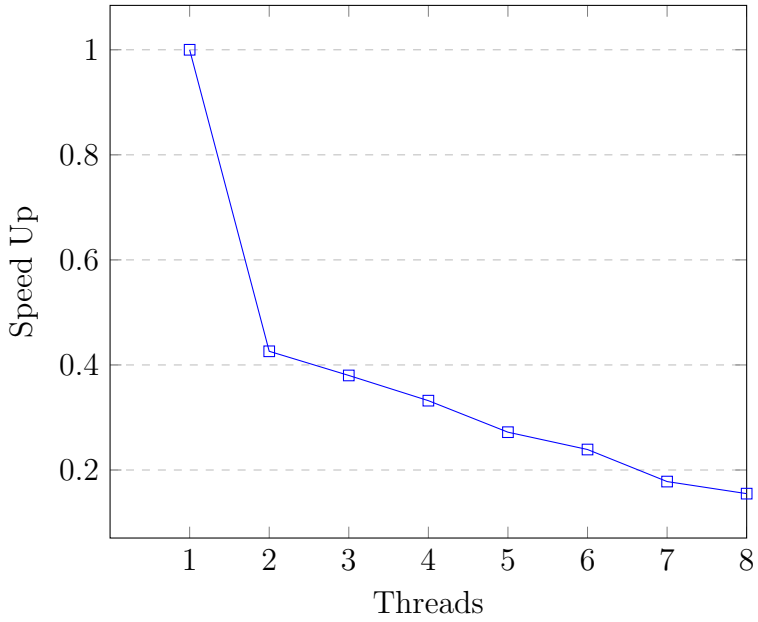


Figura 11: Grafico dello Speed Up Matrice $10^3 \times 10^2$ - Vettore 10^2

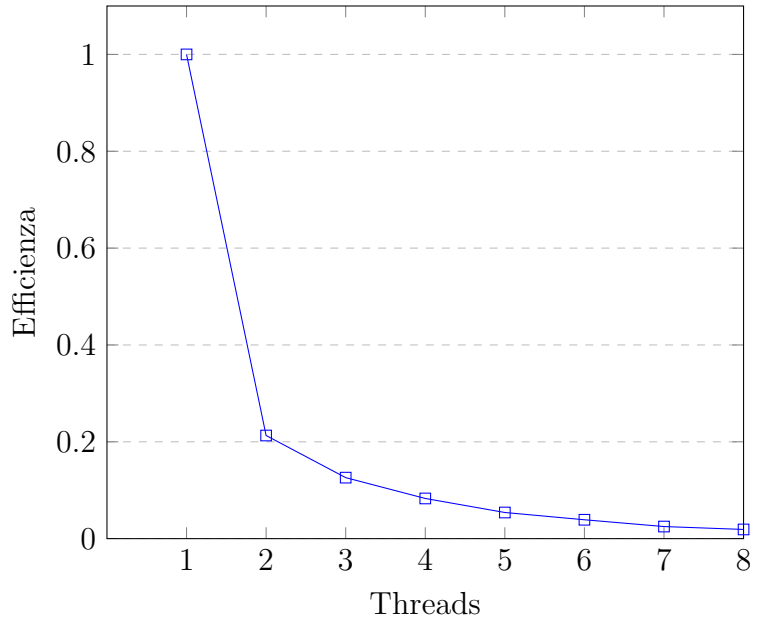


Figura 12: Grafico dell'Efficienza Matrice $10^3 \times 10^2$ - Vettore 10^2

6.6 Analisi Matrice $10^3 \times 10^3$ - Vettore 10^3

	Matrice $10^3 \times 10^3$ - Vettore 10^3		
Threads	**Tempo medio**	**Speed Up**	**Efficienza**
T1	0,009915	1,000000	1,000000
T2	0,007797	1,271624	0,635812
T3	0,006487	1,528427	0,509476
T4	0,007024	1,411532	0,352883
T5	0,004685	2,116424	0,423285
T6	0,006421	1,544186	0,257364
T7	0,008755	1,132450	0,161779
T8	0,011477	0,863852	0,107981

Tabella 15: Dati rilevati con Matrice $10^3 \times 10^3$ - Vettore 10^3

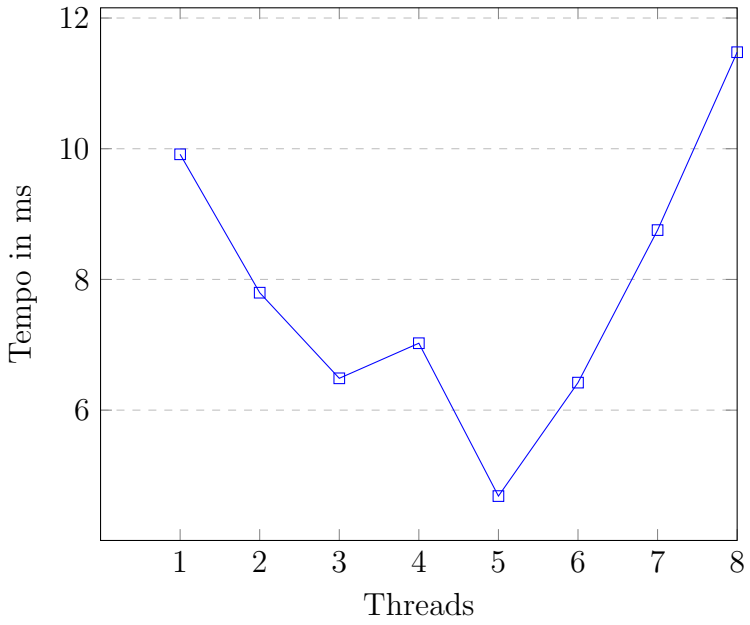


Figura 13: Grafico del tempo medio Matrice $10^3 \times 10^3$ - Vettore 10^3

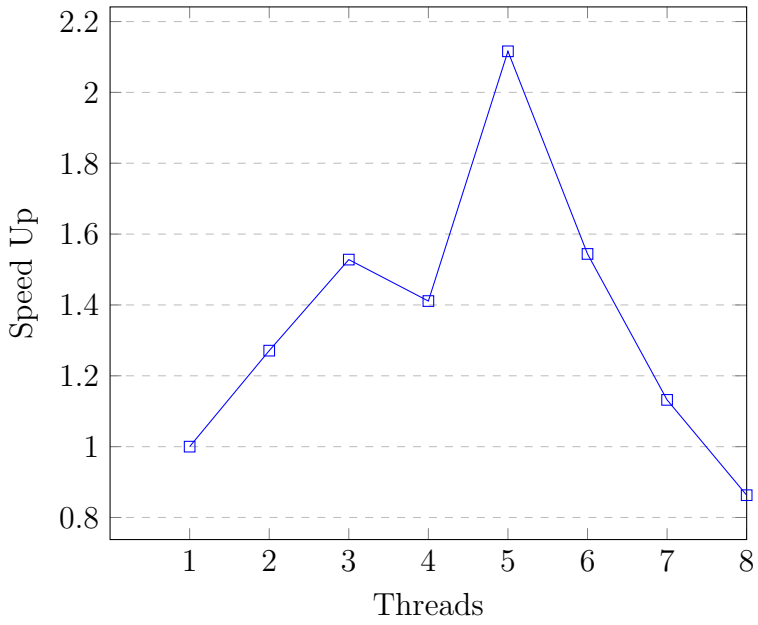


Figura 14: Grafico dello Speed Up Matrice $10^3 \times 10^3$ - Vettore 10^3

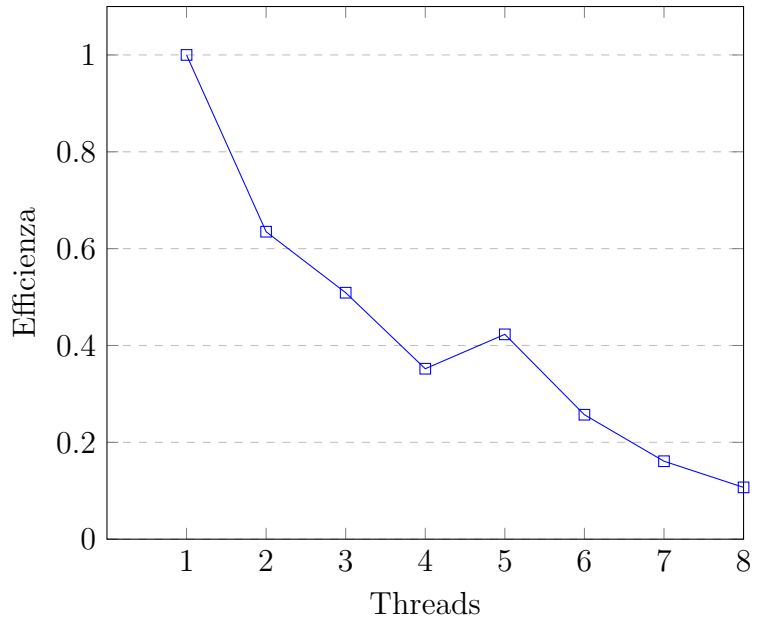


Figura 15: Grafico dell'Efficienza Matrice $10^3 \times 10^3$ - Vettore 10^3

6.7 Analisi Matrice $10^3 \times 10^4$ - Vettore 10^4

	Matrice $10^3 \times 10^4$ - Vettore 10^4		
Threads	**Tempo medio**	**Speed Up**	**Efficienza**
T1	0,095785	1,000000	1,000000
T2	0,053667	1,784792	0,892396
T3	0,038709	2,474458	0,824819
T4	0,031094	3,080491	0,770123
T5	0,025523	3,752911	0,750582
T6	0,023971	3,995862	0,665977
T7	0,019669	4,869885	0,695698
T8	0,021705	4,413029	0,551629

Tabella 16: Dati rilevati con Matrice $10^3 \times 10^4$ - Vettore 10^4

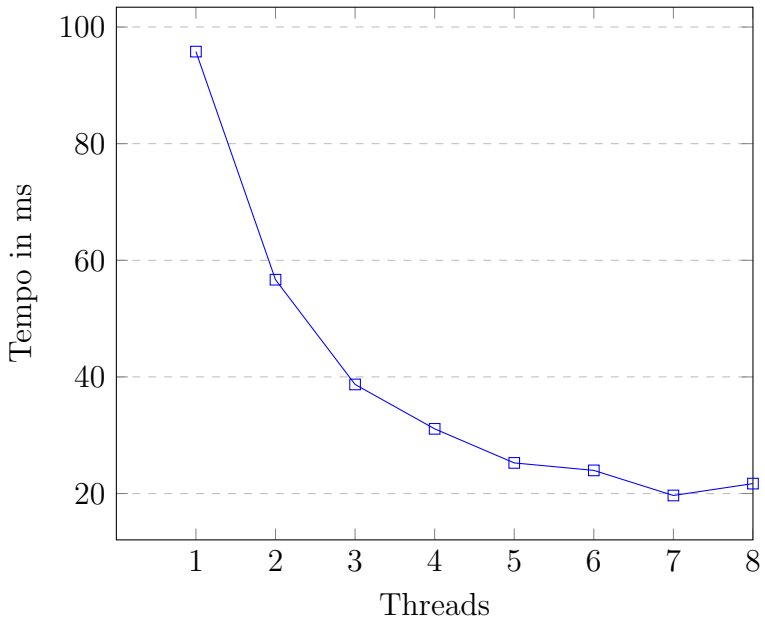


Figura 16: Grafico del tempo medio Matrice $10^3 \times 10^4$ - Vettore 10^4

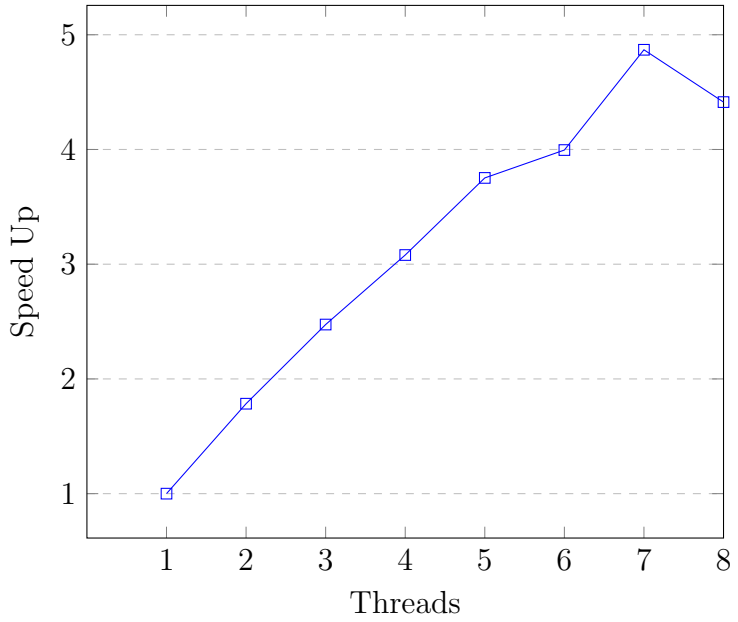


Figura 17: Grafico dello Speed Up Matrice $10^3 \times 10^4$ - Vettore 10^4

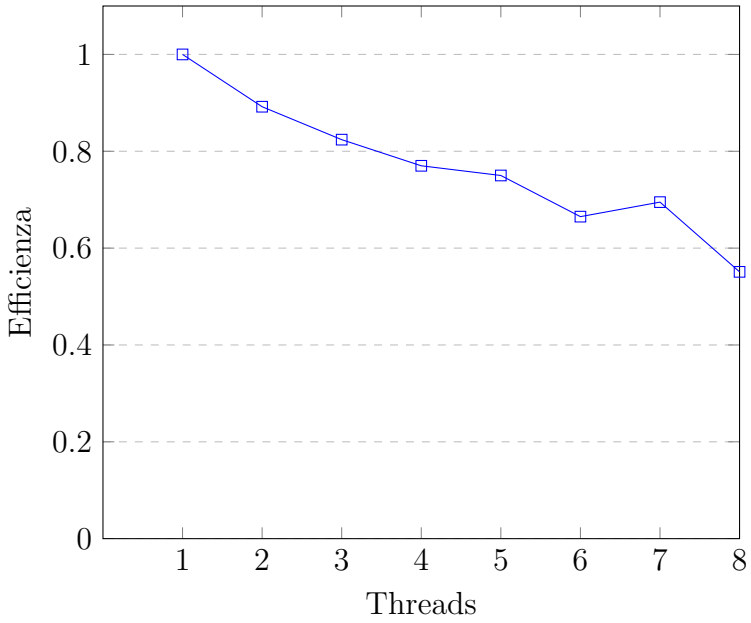


Figura 18: Grafico dell'Efficienza Matrice $10^3 \times 10^4$ - Vettore 10^4

6.8 Analisi Matrice $10^4 \times 10^2$ - Vettore 10^2

	Matrice $10^4 \times 10^2$ - Vettore 10^2		
Threads	**Tempo medio**	**Speed Up**	**Efficienza**
T1	0,010452	1,000000	1,000000
T2	0,006259	1,669841	0,834920
T3	0,007197	1,452300	0,484100
T4	0,006829	1,530606	0,382651
T5	0,005412	1,931158	0,386232
T6	0,006166	1,695245	0,282541
T7	0,008076	1,294198	0,184885
T8	0,013749	0,760237	0,095030

Tabella 17: Dati rilevati con Matrice $10^4 \times 10^2$ - Vettore 10^2

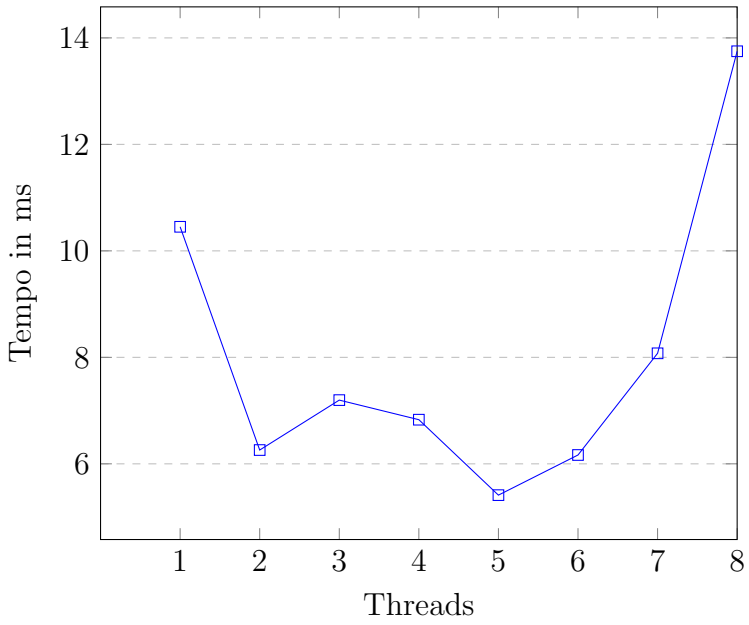


Figura 19: Grafico del tempo medio Matrice $10^4 \times 10^2$ - Vettore 10^2

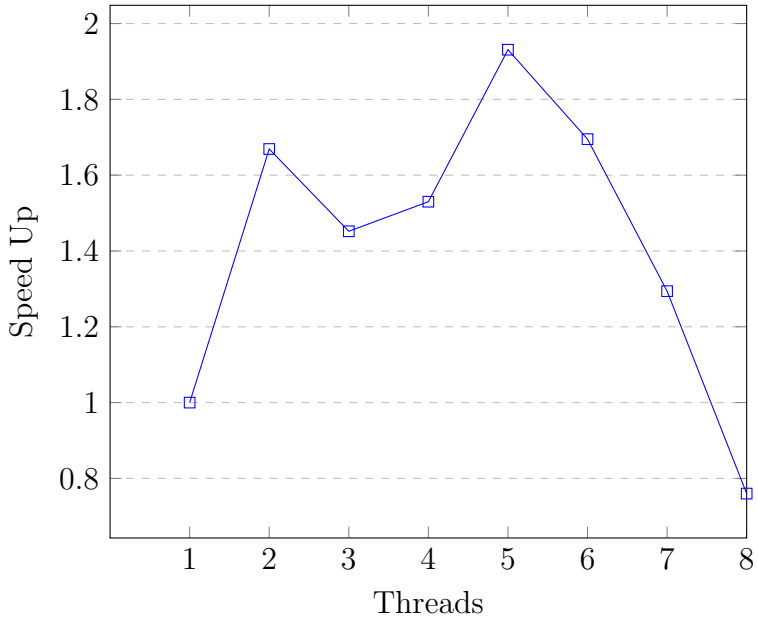


Figura 20: Grafico dello Speed Up Matrice $10^4 \times 10^2$ - Vettore 10^2

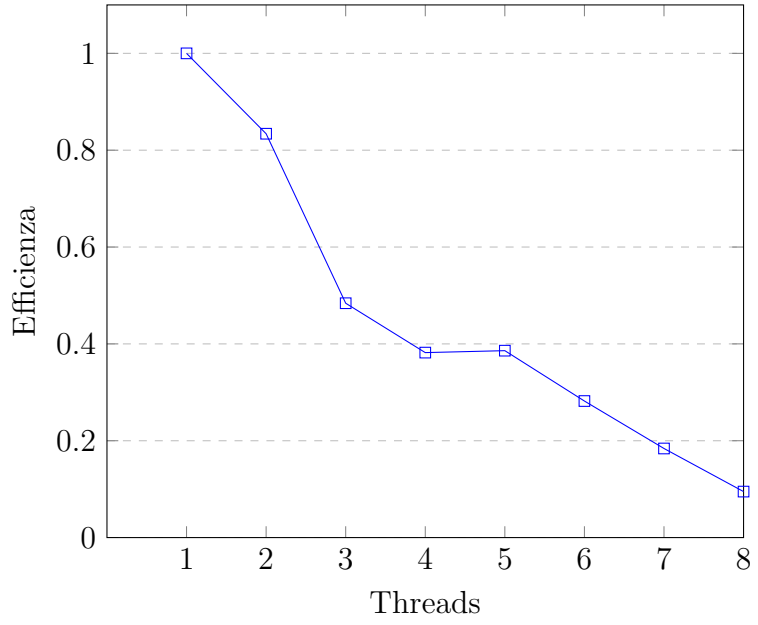


Figura 21: Grafico dell'Efficienza Matrice $10^4 \times 10^2$ - Vettore 10^2

6.9 Analisi Matrice $10^4 \times 10^3$ - Vettore 10^3

	Matrice $10^4 \times 10^3$ - Vettore 10^3		
Threads	**Tempo medio**	**Speed Up**	**Efficienza**
T1	0,094514	1,000000	1,000000
T2	0,054741	1,726581	0,863290
T3	0,035892	2,633271	0,877757
T4	0,031522	2,998401	0,749600
T5	0,024960	3,786604	0,757321
T6	0,022849	4,136442	0,689407
T7	0,019502	4,846346	0,692335
T8	0,021728	4,349970	0,543746

Tabella 18: Dati rilevati con Matrice $10^4 \times 10^3$ - Vettore 10^3

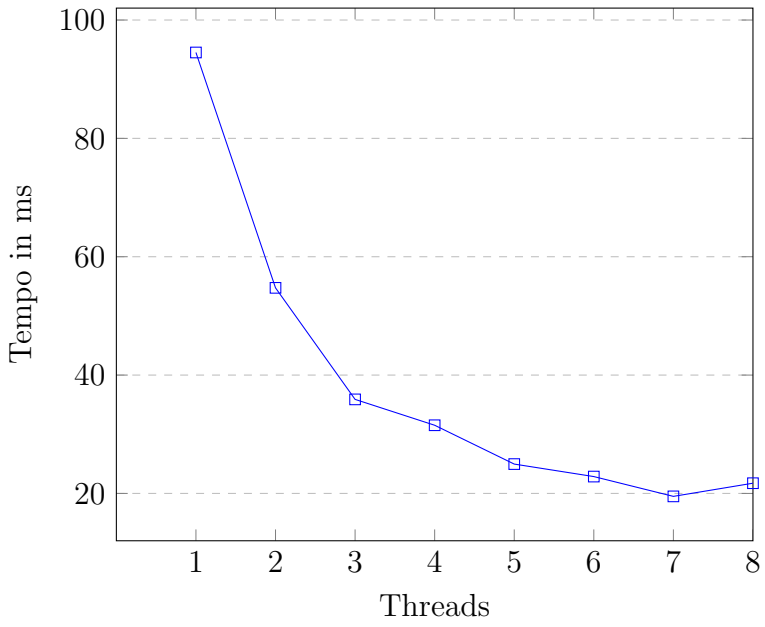


Figura 22: Grafico del tempo medio Matrice $10^4 \times 10^3$ - Vettore 10^3

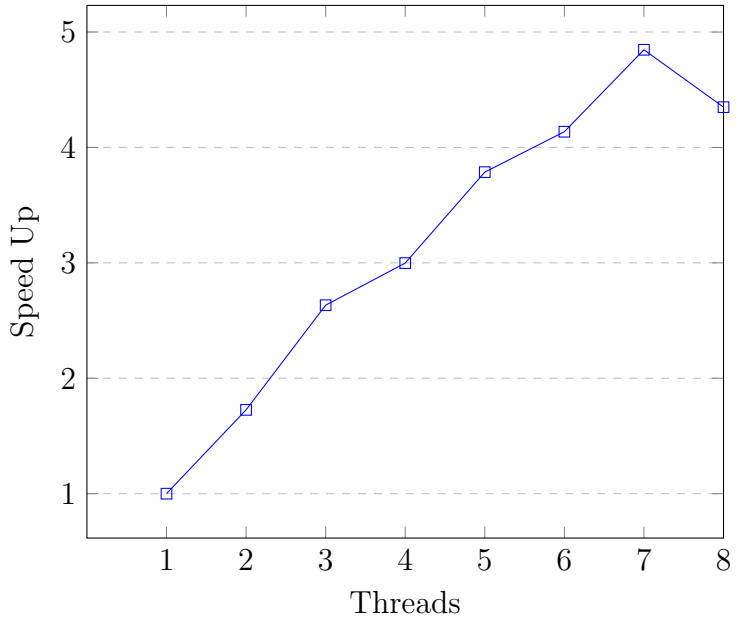


Figura 23: Grafico dello Speed Up Matrice $10^4 \times 10^3$ - Vettore 10^3

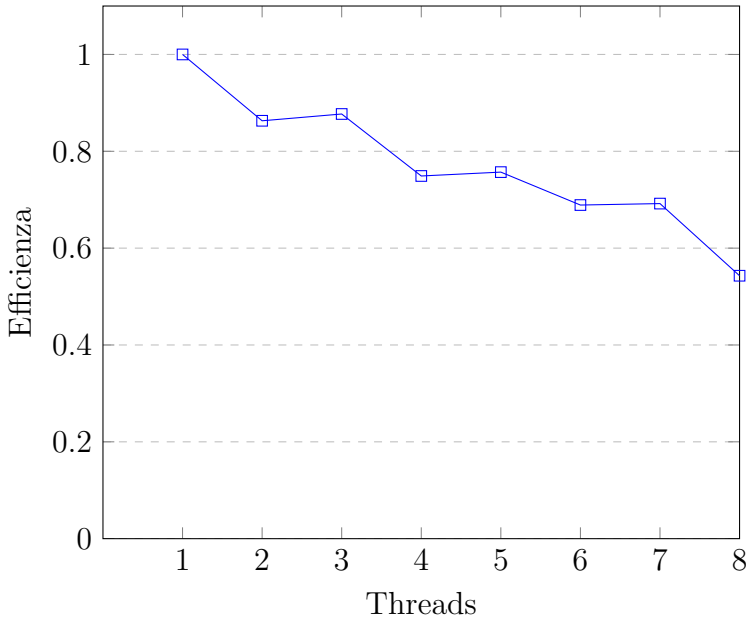


Figura 24: Grafico dell'Efficienza Matrice $10^4 \times 10^3$ - Vettore 10^3

6.10 Analisi Matrice $10^4 \times 10^4$ - Vettore 10^4

	Matrice $10^4 \times 10^4$ - Vettore 10^4		
Threads	**Tempo medio**	**Speed Up**	**Efficienza**
T1	0,957655	1,000000	1,000000
T2	0,486176	1,969773	0,984886
T3	0,329128	2,909676	0,969892
T4	0,249077	3,844820	0,961205
T5	0,203267	4,711318	0,942264
T6	0,172148	5,562964	0,927161
T7	0,152452	6,281668	0,897381
T8	0,141433	6,771070	0,846384

Tabella 19: Dati rilevati con Matrice $10^4 \times 10^4$ - Vettore 10^4

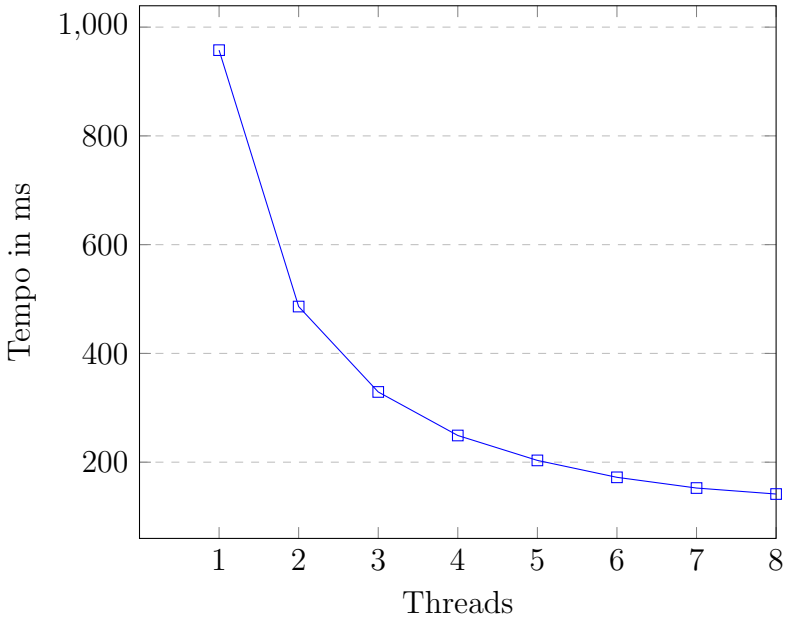


Figura 25: Grafico del tempo medio Matrice $10^4 \times 10^4$ - Vettore 10^4

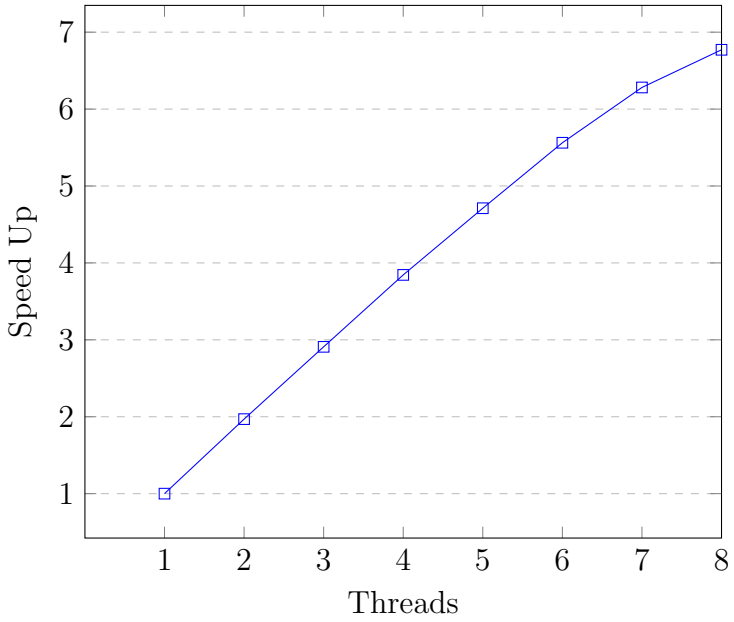


Figura 26: Grafico dello Speed Up Matrice $10^4 \times 10^4$ - Vettore 10^4

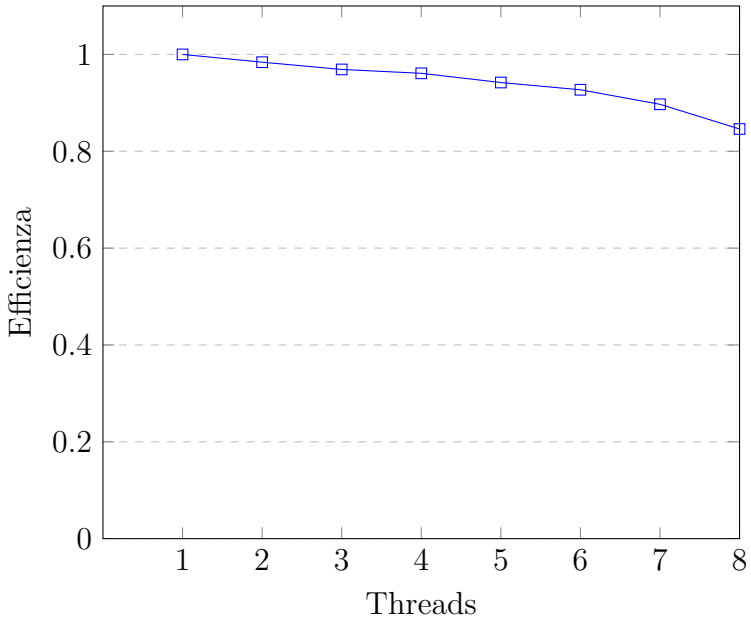


Figura 27: Grafico dell'Efficienza Matrice $10^4 \times 10^4$ - Vettore 10^4

6.11 Considerazioni sui risultati ottenuti

In questo studio, sono stati condotti diversi test tenendo in considerazione i seguenti parametri:

- r: numero totale di righe della matrice A. Tale parametro assume valori in $R = \{10^e : e \in [2, 4]\}$;
- c: numero totale di colonne della matrice A e dimensione del vettore b. Tale parametro assume valori in $C = \{10^e : e \in [2, 4]\}$;
- t: numero di thread impiegati. Tale parametro assume valori in $T = [1, 8]$.

Ogni test è identificato dalla tupla (r, c, t, i) dove i rappresenta l'i-esimo test ($i \in I = [1, 5]$). Pertanto, essendo $(r, c, t, i) \in \{(r, c, t, i) : r \in R \wedge c \in C \wedge t \in T \wedge i \in I\} := R \times C \times T \times I = Test$, il numero di test totali effettuali è $|Test| = 360$.
I risultati ottenuti durante i test hanno mostrato come sia poco vantaggioso utilizzare molti thread per matrici di piccole dimensioni. Infatti, in questa circostanza, speed up ed efficienza assumono valori bassi.
Ad esempio, nel nostro caso specifico, fissato il numero di thread a 8, si ottengono valori ragionevoli di speed up ed efficienza per matrici di grandezza almeno $10^3 \times 10^4$; per matrici di dimensione minore, risulta poco proficuo utilizzare 8 thread.
Valori prossimi a quelli ideali si ottengono con la matrice $10^4 \times 10^4$ con $t = 2$. Si pone l'attenzione, inoltre, sul fatto che i test eseguiti su matrici $n \times m$ forniscono risultati diversi dai test effettuati su matrici $m \times n$. Tale esito è dovuto alla diversa dimensione del vettore di output che ha dimensione pari al numero di righe n della matrice.
Come ci si aspettava, all'aumentare della dimensione dell'input, i thread vengono maggiormente sfruttati e, in tal modo, speed up ed efficienza raggiungono quasi i valori ideali.

7 Esempi d'uso

7.1 Esecuzione dei test

L'esecuzione dei test sul cluster avviene mediante l'esecuzione del seguente script PBS. Per variare il numero di righe e colonne (e dimensione del vettore) è necessario modificare le variabili ROWS e COLUMNS, le quali, nel seguente PBS, sono state impostate rispettivamente a 1000 e 100. Si noti che per semplicità di verifica la stampa della matrice e del vettore generati e del vettore risultante sono stati disattivati.

```
#!/bin/bash

############################
##  The PBS directives    ##
############################

#PBS -q studenti
#PBS -l nodes=1:ppn=8

#PBS -N MatriceVettoreShared
#PBS -o MatriceVettoreShared.out
#PBS -e MatriceVettoreShared.err

#########################
## Informazioni sul Job ##
#########################

echo ------------------------------
echo 'Job is running on node(s):'
cat $PBS_NODEFILE

PBS_O_WORKDIR=$PBS_O_HOME/MatriceVettoreShared
echo ----------------------------------
echo PBS: qsub is running on $PBS_O_HOST
echo PBS: originating queue is $PBS_O_QUEUE
echo PBS: executing queue is $PBS_QUEUE
echo PBS: working directory is $PBS_O_WORKDIR
echo PBS: execution mode is $PBS_ENVIRONMENT
echo PBS: job identifier is $PBS_JOBID
```

```
30 echo PBS: job name is $PBS_JOBNAME
31 echo PBS: node file is $PBS_NODEFILE
32 echo PBS: current home directory is $PBS_O_HOME
33 echo PBS: PATH = $PBS_O_PATH
34 echo ------------------------------
35
36 export PSC_OMP_AFFINITY=TRUE
37
38 gcc -std=c99 -fopenmp -lgomp -o $PBS_O_WORKDIR/MatriceVettoreShared $PBS_O_WORKDIR/
       MatriceVettoreShared.c
39
40 ROWS=1000
41 COLUMNS=100
42
43 for THREADS in {1..8}
44 do
45   for TEST in {1..5}
46   do
47     echo -e "<============[TEST: $TEST - THREADS: $THREADS - ($ROWS, $COLUMNS)
       ]============>\n"
48     $PBS_O_WORKDIR/MatriceVettoreShared --rows $ROWS --columns $COLUMNS --threads
       $THREADS
49     echo -e "
       <================================================================================>\n
       "
50   done
51 done
```

Il risultato ottenuto dall'esecuzione dello script PBS è contenuto nel file MatriceVettoreShared.out mentre eventuali errori nel file MatriceVettoreShared.err:

```
------------------------------
Job is running on node(s):
wn278.scope.unina.it
wn278.scope.unina.it
wn278.scope.unina.it
wn278.scope.unina.it
wn278.scope.unina.it
wn278.scope.unina.it
wn278.scope.unina.it
wn278.scope.unina.it
---------------------------------
PBS: qsub is running on ui-studenti.scope.unina.it
```

```
PBS: originating queue is studenti
PBS: executing queue is studenti
PBS: working directory is /homes/DMA/PDC/2021/LRCGLC98Q/MatriceVettoreShared
PBS: execution mode is PBS_BATCH
PBS: job identifier is 3993546.torque02.scope.unina.it
PBS: job name is MatriceVettoreShared
PBS: node file is /var/spool/pbs/aux//3993546.torque02.scope.unina.it
PBS: current home directory is /homes/DMA/PDC/2021/LRCGLC98Q
PBS: PATH = /usr/lib64/openmpi/1.2.7-gcc/bin:/usr/kerberos/bin:/opt/exp_soft/unina.
    it/intel/composer_xe_2013_sp1.3.174/bin/intel64:/opt/exp_soft/unina.it/intel/
    composer_xe_2013_sp1.3.174/mpirt/bin/intel64:/opt/exp_soft/unina.it/intel/
    composer_xe_2013_sp1.3.174/bin/intel64:/opt/exp_soft/unina.it/intel/
    composer_xe_2013_sp1.3.174/bin/intel64_mic:/opt/exp_soft/unina.it/intel/
    composer_xe_2013_sp1.3.174/debugger/gui/intel64:/opt/d-cache/srm/bin:/opt/d-
    cache/dcap/bin:/opt/edg/bin:/opt/glite/bin:/opt/globus/bin:/opt/lcg/bin:/usr/
    local/bin:/bin:/usr/bin:/opt/exp_soft/HADOOP/hadoop-1.0.3/bin:/opt/exp_soft/
    unina.it/intel/composerxe/bin/intel64/:/opt/exp_soft/unina.it/MPJExpress/mpj-
    v0_38/bin:/homes/DMA/PDC/2021/LRCGLC98Q/bin
------------------------------

<============[TEST: 1 - THREADS: 1 - (1000, 100)]============>
 > Overall time: 0.001113
<=============================================================>

<============[TEST: 2 - THREADS: 1 - (1000, 100)]============>
 > Overall time: 0.001117
<=============================================================>

<============[TEST: 3 - THREADS: 1 - (1000, 100)]============>
 > Overall time: 0.001113
<=============================================================>

<============[TEST: 4 - THREADS: 1 - (1000, 100)]============>
 > Overall time: 0.001117
<=============================================================>

<============[TEST: 5 - THREADS: 1 - (1000, 100)]============>
 > Overall time: 0.001110
<=============================================================>


...
...
...
```

```
<============[TEST: 1 - THREADS: 8 - (1000, 100)]============>
 > Overall time: 0.006803
<============================================================>

<============[TEST: 2 - THREADS: 8 - (1000, 100)]============>
 > Overall time: 0.004386
<============================================================>

<============[TEST: 3 - THREADS: 8 - (1000, 100)]============>
 > Overall time: 0.009609
<============================================================>

<============[TEST: 4 - THREADS: 8 - (1000, 100)]============>
 > Overall time: 0.009761
<============================================================>

<============[TEST: 5 - THREADS: 8 - (1000, 100)]============>
 > Overall time: 0.007670
<============================================================>
```

7.2 Esecuzione utente

L'esecuzione del programma sul cluster con argomenti variabili a discrezione dell'utente avviene mediante l'esecuzione del seguente script PBS:

```
#!/bin/bash

############################
##  The PBS directives    ##
############################

#PBS -q studenti
#PBS -l nodes=1:ppn=8

#PBS -N MatriceVettoreShared
#PBS -o MatriceVettoreShared.out
#PBS -e MatriceVettoreShared.err

##########################
## Informazioni sul Job ##
##########################
```

```
17
18 echo ------------------------------
19 echo 'Job is running on node(s):'
20 cat $PBS_NODEFILE
21
22 PBS_O_WORKDIR=$PBS_O_HOME/MatriceVettoreShared
23 echo ----------------------------------
24 echo PBS: qsub is running on $PBS_O_HOST
25 echo PBS: originating queue is $PBS_O_QUEUE
26 echo PBS: executing queue is $PBS_QUEUE
27 echo PBS: working directory is $PBS_O_WORKDIR
28 echo PBS: execution mode is $PBS_ENVIRONMENT
29 echo PBS: job identifier is $PBS_JOBID
30 echo PBS: job name is $PBS_JOBNAME
31 echo PBS: node file is $PBS_NODEFILE
32 echo PBS: current home directory is $PBS_O_HOME
33 echo PBS: PATH = $PBS_O_PATH
34 echo ------------------------------
35
36 export PSC_OMP_AFFINITY=TRUE
37
38 gcc -std=c99 -fopenmp -lgomp -o $PBS_O_WORKDIR/MatriceVettoreShared $PBS_O_WORKDIR/
       MatriceVettoreShared.c
39
40 ROWS=1000
41 COLUMNS=100
42 THREADS=3
43
44 echo -e "<============[THREADS: $THREADS - (ROWS: $ROWS, COLUMNS: $COLUMNS)
       ]============>\n"
45 $PBS_O_WORKDIR/MatriceVettoreShared --rows $ROWS --columns $COLUMNS --threads
       $THREADS
46 echo -e "
       <================================================================================>\n
       "
```

```
------------------------------
Job is running on node(s):
wn280.scope.unina.it
wn280.scope.unina.it
wn280.scope.unina.it
wn280.scope.unina.it
wn280.scope.unina.it
```

```
wn280.scope.unina.it
wn280.scope.unina.it
wn280.scope.unina.it
-----------------------------------
PBS: qsub is running on ui-studenti.scope.unina.it
PBS: originating queue is studenti
PBS: executing queue is studenti
PBS: working directory is /homes/DMA/PDC/2021/LRCGLC98Q/MatriceVettoreShared
PBS: execution mode is PBS_BATCH
PBS: job identifier is 3993554.torque02.scope.unina.it
PBS: job name is MatriceVettoreShared
PBS: node file is /var/spool/pbs/aux//3993554.torque02.scope.unina.it
PBS: current home directory is /homes/DMA/PDC/2021/LRCGLC98Q
PBS: PATH = /usr/lib64/openmpi/1.2.7-gcc/bin:/usr/kerberos/bin:/opt/exp_soft/unina.
    it/intel/composer_xe_2013_sp1.3.174/bin/intel64:/opt/exp_soft/unina.it/intel/
    composer_xe_2013_sp1.3.174/mpirt/bin/intel64:/opt/exp_soft/unina.it/intel/
    composer_xe_2013_sp1.3.174/bin/intel64:/opt/exp_soft/unina.it/intel/
    composer_xe_2013_sp1.3.174/bin/intel64_mic:/opt/exp_soft/unina.it/intel/
    composer_xe_2013_sp1.3.174/debugger/gui/intel64:/opt/d-cache/srm/bin:/opt/d-
    cache/dcap/bin:/opt/edg/bin:/opt/glite/bin:/opt/globus/bin:/opt/lcg/bin:/usr/
    local/bin:/bin:/usr/bin:/opt/exp_soft/HADOOP/hadoop-1.0.3/bin:/opt/exp_soft/
    unina.it/intel/composerxe/bin/intel64/:/opt/exp_soft/unina.it/MPJExpress/mpj-
    v0_38/bin:/homes/DMA/PDC/2021/LRCGLC98Q/bin
-------------------------------

<============[THREADS: 3 - (ROWS: 10, COLUMNS: 5)]============>

 > Generated Matrix

 [4.054509] [3.430001] [2.122511] [0.537839] [1.644213]
 [2.084644] [4.332371] [0.103883] [0.145154] [4.696847]
 [3.815714] [3.944536] [0.916593] [1.707854] [3.208831]
 [1.166251] [1.152477] [4.475820] [0.400901] [4.022685]
 [4.337179] [3.731524] [3.615560] [4.200206] [2.717239]
 [0.307365] [2.293992] [2.311783] [1.485715] [3.591475]
 [1.451724] [0.540224] [2.021477] [3.574235] [1.078064]
 [3.665690] [0.658879] [0.410434] [3.769573] [0.804032]
 [0.107281] [2.585286] [4.748568] [1.023874] [4.293140]
 [2.957399] [2.190125] [0.445618] [2.433219] [2.591026]


 > Generated Vector
```

```
[4.468302]  [1.770398]  [1.322550]  [3.083862]  [0.970604]


> Product Vector

[30.250868]  [22.128639]  [33.626687]  [18.311754]  [46.358083]  [16.559763]
   [22.185474]  [30.493946]  [18.660989]  [27.699865]


> Overall time: 0.003075

<============================================================>
```

7.2.1 Composizione argomenti

Il programma prende in ingresso i seguenti argomenti (necessariamente in ordine):

- **-r** o equivalentemente –**rows**

```
# Righe della matrice: 1000
$PBS_O_WORKDIR/MatriceVettoreShared -r 1000

# Oppure, in modo equivalente:
$PBS_O_WORKDIR/MatriceVettoreShared --rows 1000
```

- **-c** o equivalentemente –**columns**

```
# Colonne della matrice: 100
$PBS_O_WORKDIR/MatriceVettoreShared -r 1000 -c 100

# Oppure, in modo equivalente:
$PBS_O_WORKDIR/MatriceVettoreShared -r 1000 --columns 100
```

- **-t** o equivalentemente –**threads**

```
# Numero di thread da impiegare: 8
$PBS_O_WORKDIR/MatriceVettoreShared --rows 1000 --columns 100 -t 8

# Oppure, in modo equivalente:
$PBS_O_WORKDIR/MatriceVettoreShared -r 1000 --columns 100 --threads 8
```

7.2.2 Help

Il programma offre anche la possibilità di stampare (sul file .out) l'help, ossia le informazioni sugli argomenti necessari per avviare il programmare correttamente:

```
...
...
...
# Per stampare l'help e' necessario passare al programma esclusivamente l'argomento
    -- help

echo -e "\ n\n\ n ################## INI EXEC ##################\ n"
$PBS_O_WORKDIR/MatriceVettoreShared --help
echo "################## END EXEC ##################"
```

Successivamente all'esecuzione dello script PBS riportato precedentemente, sul file .out troveremo:

```
> Usage: /homes/DMA/PDC/2021/LRCGLC98Q/MatriceVettoreShared/MatriceVettoreShared
   [-r --rows] <value> [-c --columns] <value> [-t --threads] <value>

     Mandatory arguments:
       -r  --rows              Number of rows of the matrix
       -c  --columns           Number of columns of the matrix
       -t  --threads           Number of threads to use

     Error codes:
       400 ERR_ARGC            Invalid number of arguments
       401 ERR_NO_ROWS         Mandatory argument [-r --rows] not provided
       402 ERR_NO_COLUMNS      Mandatory argument [-c --columns] not provided
       401 ERR_NO_ROWS         Mandatory argument [-r --rows] not provided
       404 ERR_INVLD_ROWS      Invalid number of rows provided
       405 ERR_INVLD_COLUMNS   Invalid number of columns provided
       406 ERR_INVLD_THREADS   Invalid number of threads provided
       407 ERR_MEMORY          Unable to allocate memory
```

7.2.3 Esempio di esecuzione

```
...
...
...
ROWS=5
```

```
COLUMNS=7
THREADS=3

echo -e "<============[THREADS: $THREADS - (ROWS: $ROWS, COLUMNS: $COLUMNS)
    ]============>\n"
$PBS_O_WORKDIR/MatriceVettoreShared --rows $ROWS --columns $COLUMNS --threads
    $THREADS
echo -e "
    <================================================================================>\n
    "
```

- Righe della matrice: 5;
- Colonne della matrice e dimensione del vettore: 7;
- Numero di thread impiegati: 3.

```
<============[THREADS: 3 - (ROWS: 5, COLUMNS: 7)]============>

 > Generated Matrix

 [1.118004] [1.448217] [1.071735] [4.816095] [1.047872] [0.435353] [3.899735]
 [1.771484] [0.030638] [3.705615] [2.133921] [0.789950] [2.938619] [4.725263]
 [1.613037] [1.598277] [1.232920] [3.766289] [2.230965] [1.077580] [3.797475]
 [4.850384] [0.304132] [4.498524] [0.986333] [1.246625] [3.998720] [0.899339]
 [2.643776] [4.928749] [3.957660] [3.761780] [1.376966] [0.029396] [3.577875]


 > Generated Vector

 [2.424839]  [0.464748]  [2.477610]  [4.196323]  [0.495386]  [1.183224]  [1.330244]


 > Product Vector

 [32.471087]  [32.599621]  [30.945186]  [33.732581]  [39.768895]


 > Overall time: 0.006236

<============================================================>
```

8 Codice sorgente

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <sys/time.h>
#include <string.h>
#include <omp.h>


#define SD_ARG_ROWS         "-r"
#define SD_ARG_COLUMNS      "-c"
#define SD_ARG_THREADS      "-t"

#define DD_ARG_ROWS         "--rows"
#define DD_ARG_COLUMNS      "--columns"
#define DD_ARG_THREADS      "--threads"
#define DD_ARG_HELP         "--help"

#define SCC_ARGS            200
#define SCC_HELP            201

#define ERR_ARGC            400
#define ERR_NO_ROWS         401
#define ERR_NO_COLUMNS      402
#define ERR_NO_THREADS      403
#define ERR_INVLD_ROWS      404
#define ERR_INVLD_COLUMNS   405
#define ERR_INVLD_THREADS   406
#define ERR_MEMORY          407


void help(char*);
int check_args(int, char**);
double** generate_random_matrix(int, int, double, double);
double* generate_random_vector(int, double, double);
double* product(int, int, double**, double* restrict);
void print_vector(double*, int);
void print_matrix(double**, int, int);


int main(int argc, char** argv) {

```

```
42  /*
43    rows: Numero di righe della matrice;
44    columns: Numero di colonne della matrice;
45    threads: Numero di thread impiegati;
46
47    matrix: Matrice da impiegare nel prodotto;
48    vector: Vettore da impiegare nel prodotto;
49    result: Vettore risultate dal prodotto tra matrix e vector
50
51    time_start: Timestamp all'inizio del calcolo del prodotto;
52    time_end: Timestamp al termine del calcolo del prodotto;
53    overall_time: Tempo complessivo impiegato per effettuare il prodotto;
54  */
55
56  int rows;
57  int columns;
58  int threads;
59
60  double** matrix;
61  double* vector;
62  double* result;
63
64  struct timeval time_start;
65  struct timeval time_end;
66  double overall_time;
67
68
69  /* Controllo degli argomenti passati in ingresso */
70
71  switch(check_args(argc, argv)) {
72
73    case SCC_HELP:
74      help(argv[0]);
75      return SCC_HELP;
76
77    case ERR_ARGC:
78      printf("\n <!> ERROR: Invalid number of arguments! For additional info type %
    s.\n", DD_ARG_HELP);
79      return ERR_ARGC;
80
81    case ERR_NO_ROWS:
82      printf(
83        "\n <!> ERROR: Expected [%s %s] argument! For additional info type %s.\n",
```

```
84          SD_ARG_ROWS, DD_ARG_ROWS, DD_ARG_HELP
85        );
86        return ERR_NO_ROWS;
87
88      case ERR_NO_COLUMNS:
89        printf(
90          "\n <!> ERROR: Expected [%s %s] argument! For additional info type %s.\n",
91          SD_ARG_COLUMNS, DD_ARG_COLUMNS, DD_ARG_HELP
92        );
93        return ERR_NO_COLUMNS;
94
95      case ERR_NO_THREADS:
96        printf(
97          "\n <!> ERROR: Expected [%s %s] argument! For additional info type %s.\n",
98          SD_ARG_THREADS, DD_ARG_THREADS, DD_ARG_HELP
99        );
100       return ERR_NO_THREADS;
101
102     case ERR_INVLD_ROWS:
103       printf("\n <!> ERROR: Invalid value for argument [%s %s]! For additional info
      type %s.\n",
104         SD_ARG_ROWS, DD_ARG_ROWS, DD_ARG_HELP
105       );
106       return ERR_INVLD_ROWS;
107
108     case ERR_INVLD_COLUMNS:
109       printf("\n <!> ERROR: Invalid value for argument [%s %s]! For additional info
      type %s.\n",
110         SD_ARG_COLUMNS, DD_ARG_COLUMNS, DD_ARG_HELP
111       );
112       return ERR_INVLD_COLUMNS;
113
114     case ERR_INVLD_THREADS:
115       printf("\n <!> ERROR: Invalid value for argument [%s %s]! For additional info
      type %s.\n",
116         SD_ARG_THREADS, DD_ARG_THREADS, DD_ARG_HELP
117       );
118       return ERR_INVLD_THREADS;
119
120   }
121
122
123   /* Lettura degli argomenti passati in ingresso */
```

```
124
125   rows = atoi(argv[2]);
126   columns = atoi(argv[4]);
127   threads = atoi(argv[6]);
128
129
130   /* Generazione della matrice e del vettore */
131
132   srand(time(NULL));
133   matrix = generate_random_matrix(rows, columns, 0.0, 5.0);
134   vector = generate_random_vector(columns, 0.0, 5.0);
135   if(!matrix || !vector) {
136     printf("\n <!> ERROR: Unable to allocate memory.\n");
137     return ERR_MEMORY;
138   }
139
140
141   /* Stampa della matrice e del vettore generati */
142
143   printf("\n > Generated Matrix \n\n");
144   print_matrix(matrix, rows, columns);
145   printf("\n\n > Generated Vector \n\n");
146   print_vector(vector, columns);
147
148
149   /* Imposta il numero di thread da impiegare */
150
151   omp_set_num_threads(threads);
152
153
154   /* Cattura il timestamp all'inizio del calcolo del prodotto */
155
156   gettimeofday(&time_start, NULL);
157
158
159   /* Effettua il prodotto tra la matrice e il vettore */
160
161     result = product(rows, columns, matrix, vector);
162     if(!result){
163     printf("\n <!> ERROR: Unable to allocate memory.\n");
164     return ERR_MEMORY;
165   }
166
```

```
167
168     /* Cattura il timestamp al termine del calcolo del prodotto */
169
170     gettimeofday(&time_end, NULL);
171
172
173     /* Calcolo il tempo impiegato */
174
175   overall_time = (time_end.tv_sec+(time_end.tv_usec/1000000.0)) -
176                  (time_start.tv_sec+(time_start.tv_usec/1000000.0));
177
178
179   /* Stampa del prodotto e del tempo impiegato */
180
181   printf("\n\n > Product Vector \n\n");
182   print_vector(result, rows);
183   printf("\n\n > Overall time: %lf\n", overall_time);
184
185   return 0;
186
187 }
188
189
190 /*
191
192   Stampa a video l'help del programma
193
194   @params:
195     char* program_name: Nome del programma
196
197   @return:
198     void
199
200 */
201
202 void help(char* program_name) {
203
204   printf(
205     "\n > Usage: %s [%s %s] <value> [%s %s] <value> [%s %s] <value>",
206     program_name,
207     SD_ARG_ROWS, DD_ARG_ROWS,
208     SD_ARG_COLUMNS, DD_ARG_COLUMNS,
209     SD_ARG_THREADS, DD_ARG_THREADS
```

```
210   );
211
212   printf("\n\n      Mandatory arguments:");
213   printf("\n        %s  %-20s Number of rows of the matrix", SD_ARG_ROWS,
        DD_ARG_ROWS);
214   printf("\n        %s  %-20s Number of columns of the matrix", SD_ARG_COLUMNS,
        DD_ARG_COLUMNS);
215   printf("\n        %s  %-20s Number of threads to use", SD_ARG_THREADS,
        DD_ARG_THREADS);
216
217   printf("\n\n      Error codes:");
218   printf("\n        %d %-20s Invalid number of arguments", ERR_ARGC, "ERR_ARGC");
219   printf(
220     "\n        %d %-20s Mandatory argument [%s %s] not provided",
221     ERR_NO_ROWS, "ERR_NO_ROWS",
222     SD_ARG_ROWS, DD_ARG_ROWS
223   );
224   printf(
225     "\n        %d %-20s Mandatory argument [%s %s] not provided",
226     ERR_NO_COLUMNS, "ERR_NO_COLUMNS",
227     SD_ARG_COLUMNS, DD_ARG_COLUMNS
228   );
229   printf(
230     "\n        %d %-20s Mandatory argument [%s %s] not provided",
231     ERR_NO_THREADS, "ERR_NO_THREADS",
232     SD_ARG_THREADS, DD_ARG_THREADS
233   );
234   printf("\n        %d %-20s Invalid number of rows provided", ERR_INVLD_ROWS, "
        ERR_INVLD_ROWS");
235   printf("\n        %d %-20s Invalid number of columns provided", ERR_INVLD_COLUMNS
        , "ERR_INVLD_COLUMNS");
236   printf("\n        %d %-20s Invalid number of threads provided", ERR_INVLD_THREADS
        , "ERR_INVLD_THREADS");
237   printf("\n        %d %-20s Unable to allocate memory\n", ERR_MEMORY, "ERR_MEMORY"
        );
238
239 }
240
241
242 /*
243
244   Verifica l'integrita' degli argomenti passati in ingresso al programma
245
```

```
246   @params:
247     int argc: Numero di argomenti passati in ingresso al programma
248     char* argv[]: Argomenti passati in ingresso al programma
249
250   @return:
251     int: Codice errore/successo
252
253 */
254
255 int check_args(int argc, char** argv) {
256
257   if(argc == 2 && !strcmp(argv[1], DD_ARG_HELP))
258     return SCC_HELP;
259
260   if(argc == 7) {
261
262     if(strcmp(argv[1], SD_ARG_ROWS) && strcmp(argv[1], DD_ARG_ROWS))
263       return ERR_NO_ROWS;
264
265     if(strcmp(argv[3], SD_ARG_COLUMNS) && strcmp(argv[3], DD_ARG_COLUMNS))
266       return ERR_NO_COLUMNS;
267
268     if(strcmp(argv[5], SD_ARG_THREADS) && strcmp(argv[5], DD_ARG_THREADS))
269       return ERR_NO_THREADS;
270
271     int rows = atoi(argv[2]);
272     int columns = atoi(argv[4]);
273     int threads = atoi(argv[6]);
274
275     if(rows <= 0)
276       return ERR_INVLD_ROWS;
277
278     if(columns <= 0)
279       return ERR_INVLD_COLUMNS;
280
281     if(threads <= 0)
282       return ERR_INVLD_THREADS;
283
284     return SCC_ARGS;
285
286   }
287
288   return ERR_ARGC;
```

```
289
290 }
291
292
293 /*
294
295   Genera una matrice di "rows x columns" elementi reali pseudo-randomici
296
297   @params:
298     int rows: Numero di righe della matrice
299     int columns: Numero di colonne della matrice
300     double lower: Limite inferiore del valore degli elementi
301     double upper: Limite superiore del valore degli elementi
302
303   @return:
304     double**: Matrice generata
305
306 */
307
308 double** generate_random_matrix(int rows, int columns, double lower, double upper)
      {
309
310   double** matrix = (double**) malloc(rows*sizeof(double*));
311     if(matrix) {
312         for(int i = 0; i < rows; i++) {
313           matrix[i] = (double*) calloc(columns, sizeof(double));
314           if(!matrix[i])
315             return NULL;
316           for(int j = 0; j < columns; j++)
317               matrix[i][j] = (((double)rand()*(upper-lower))/(double)RAND_MAX+lower
      );
318       }
319     }
320     return matrix;
321
322 }
323
324
325 /*
326
327   Genera un vettore di "dimension" elementi reali pseudo-randomici
328
329   @params:
```

```
330     int dimension: Dimensione del vettore
331     double lower: Limite inferiore del valore degli elementi
332     double upper: Limite superiore del valore degli elementi
333 
334   @return:
335     double**: Vettore generato
336 
337 */
338 
339 double* generate_random_vector(int dimension, double lower, double upper) {
340 
341   double* vector = (double*) calloc(dimension, sizeof(double));
342     if(vector) {
343         for(int i = 0; i < dimension; i++)
344             vector[i] = (((double)rand()*(upper-lower))/(double)RAND_MAX+lower);
345     }
346     return vector;
347 
348 }
349 
350 
351 /*
352 
353   Effettua la stampa del vettore passato in ingresso
354 
355   @params:
356     double* vector: Vettore da stampare
357     int dimension: Dimensione del vettore
358 
359   @return:
360     void
361 
362 */
363 
364 void print_vector(double* vector, int dimension) {
365 
366     for(int i = 0; i < dimension; i++)
367         printf(" [%lf] ", vector[i]);
368     printf("\n");
369 
370 }
371 
372 
```

```
373 /*
374
375   Effettua la stampa della matrice passata in ingresso
376
377   @params:
378     double** matrix: Matrice da stampare
379     int rows: Numero di righe della matrice
380     int columns: Numero di colonne della matrice
381
382   @return:
383     void
384
385 */
386
387
388 void print_matrix(double** matrix, int rows, int columns) {
389
390   for(int i = 0; i < rows; i++) {
391       for(int j = 0; j < columns; j++)
392         printf(" [%lf]", matrix[i][j]);
393       printf("\n");
394   }
395
396 }
397
398
399 /*
400
401   Effettua Il prodotto tra la matrice e il vettore passati in ingresso
402
403   @params:
404     int rows: Numero di righe della matrice
405     int columns: Numero di colonne della matrice
406     double** matrix: Matrice da impiegare nel calcolo del prodotto
407     double* vector: Vettore da impiegare nel calcolo del prodotto
408
409   @return:
410     Vettore di dimensione rows risultate dal prodotto
411
412 */
413
414 double* product(int rows, int columns, double** matrix, double* restrict vector) {
415
```

```
416   double* product = (double*) calloc(rows, sizeof(double));
417   if(product) {
418     #pragma omp parallel for default(none) shared(rows, columns, matrix, vector,
      product)
419     for(int i = 0; i < rows; i++)
420       for(int j = 0; j < columns; j++)
421         product[i] += matrix[i][j]*vector[j];
422   }
423   return product;
424
425 }
```