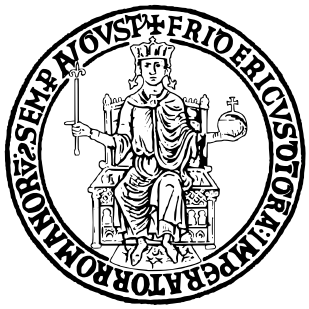
Università degli Studi di Napoli Federico II



Scuola Politecnica e delle Scienze di Base

Dipartimento di Ingegneria Elettrica e Tecnologie dell'Informazione

Corso di Laurea Magistrale in Informatica

Corso di Parallel and Distributed Computing

Calcolo del prodotto matrice-vettore

Relatore
Prof. Giuliano Laccetti
Prof.ssa Valeria Mele

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

Anno Accademico 2023-2024

Indice

Capitolo 1

Descrizione del problema

Il goal del problema è calcolare $A\vec{x} = \vec{b}$ dove $A \in \mathbb{R}^{N \times M}$, $\vec{x} \in \mathbb{R}^M$ e $\vec{b} \in \mathbb{R}^N$ in ambiente *parallelo* su architettura **MIMD** (**Multiple Instruction Multiple Data**) a memoria condivisa, utilizzando la libreria `openMP` in linguaggio C.

Più precisamente, vogliamo calcolare:

$$\begin{matrix} & A & & & & x & = & b \end{matrix}$$
$$\begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,m} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,m} \\ \vdots & \vdots & \vdots & \vdots \\ a_{n,1} & a_{n,2} & \cdots & a_{n,m} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix} = \begin{bmatrix} a_{1,1}x_1 + a_{1,2}x_2 + ... + a_{1,m}x_m \\ a_{2,1}x_1 + a_{2,2}x_2 + ... + a_{2,m}x_m \\ \vdots \\ a_{n,1}x_1 + a_{n,2}x_2 + ... + a_{n,m}x_m \end{bmatrix}$$

In particolare, verranno utilizzate matrici di dimensioni differenti, per effettuare tale prodotto e tramite dei grafici verranno mostrate le differenze tra le varie dimensioni della matrice in termini di **tempo di esecuzione**, **speed up** ed **efficienza**.

Capitolo 2

Descrizione algoritmo

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

2.1 Struttura del programma

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

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

dove

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

2.2 Analisi del programma

Il programma è diviso in 3 punti principali:

- Acquisizione dei dati
- Calcolo in parallelo del prodotto matrice-vettore mediante la direttiva *prama omp parallel* che ci permette di eseguire un blocco di codice in parallelo.

- Stampa del risultato: monitoriamo il tempo impiegato per il calcolo utilizzando la funzione `gettimeofday` e semplicemente viene stampato tale tempo.

Più precisamente, quello che viene fatto, in pseudocodice è:

Algorithm 1: Pseudocodice per il calcolo $A\vec{x} = \vec{b}$

Data: $N, M, num.threads > 0$
Result: $A\vec{x} = \vec{b}$
Inizializza e riempe A;
Inizializza e riempe $\vec{x}$;
Inizializza $\vec{b}$;
$inizio\_tempo = gettimeofday$;
$\#pragma\ omp\ parallel$;
Calcolo $A\vec{x} = \vec{b}$;
$fine\_tempo = gettimeofday$;
Stampa risultato $inizio\_tempo - fine\_tempo$

dove, $\#pragma\ omp\ parallel$ è la direttiva che permette di parallelizzare un blocco di codice, nello specifico abbiamo:

```
1  void matxvet(double *A, int N, int M, double *x, double *b){
2      int i,j;
3      #pragma omp parallel for default(none) shared(N, M, A, x, b) ↩
       private(i, j)
4      // prodotto matrice vettore
5      for (i = 0; i < N; i++){
6          for (j = 0; j < M; j++){
7              b[i] += A[i * M + j] * x[j];
8          }
9      }
10 }
```

dove,

- `for`: tale costrutto specifica che le iterazioni del ciclo contenuto devono essere distribuite tra i diversi thread
- `default(none)`: mediante questa clausola sarà il programmatore a definire quali variabili saranno condivise e quali no
- `shared(N, M, A, x, b)`: le variabili condivise fra i thread
- `private(i, j)`: le variabili private per ogni thread

2.3 Input/output

Il programma prende in input 3 parametri, che vanno inseriti esattamente nel seguente ordine:

- N: il numero di righe della matrice A
- M: il numero di colonne della matrice A
- $num\_threads$: il numero di thread da impiegare per il calcolo prodotto matrice-vettore

In output il programma restituisce il vettore risultante e il tempo impiegato. Tuttavia, nel programma viene stampato solo il risultato in quanto la stampa del vettore risultante è superflua (soprattutto nei casi in cui i dati sono troppo grandi).

2.4 Errori

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

- N, M e il numero dei threads devono essere necessariamente maggiori di 0
- naturalmente, il numero dei parametri deve essere esattamente 3

Capitolo 3

Descrizione routine

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

3.1 Funzioni *OpenMP* utilizzate

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

Descrizione: Imposta il numero di thread da usare in parallelo.

Parametri di input:

- `num_threads`: intero che indica il numero di thread da usare

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

3.2 Funzioni ausiliare

3.2.1 La funzione *matxvet*

```
void void matxvet(double *A, int N, int M, double *x, double *b);
```

Descrizione: Calcola $A\vec{x} = \vec{b}$ facendo uso della liberia `OpenMP`.

Parametri di input:

- `A`: puntatore alla matrice
- `N`: numero di righe della matrice
- `M`: numero di colonne della matrice

- `x`: puntatore al vettore

Parametri di output:

- `b`: puntatore al vettore risultante

3.2.2 La funzione *fill_matrix*

```
void fill_matrix(double *matrix, int row, int col);
```

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

Parametri di input:

- `A`: puntatore alla matrice
- `row`: numero di righe della matrice
- `col`: numero di colonne della matrice

3.2.3 La funzione *print_matrix*

```
void print_matrix(double *matrix, int row, int col);
```

Descrizione: Stampa la matrice su sdt output.

Parametri di input:

- `A`: puntatore alla matrice
- `row`: numero di righe della matrice
- `col`: numero di colonne della matrice

3.2.4 La funzione *read_input*

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

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

Parametri di input:

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

Parametri di output:

- `N`: puntatore all'intero che conterrà il numero di righe della matrice
- `M`: puntatore all'intero che conterrà il numero di colonne della matrice
- `n_threads`: puntatore all'intero che conterrà il numero massimo di thread che si potranno usare

Errors: Esce dal programma se $N, M, n\_threads$ se non sono maggiori di 0.

3.2.5 La funzione *initialize_matrix*

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

Descrizione: Inizializza la matrice e se non ci sono errori la riempe chiamando la funzione `fill_matrix` (3.2.2).

Parametri di input:

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

Parametri di output:

- `A`: puntatore all matrice da inizializzare

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

3.2.6 La funzione *initialize_array*

```
void initialize_array(double **array, int size);
```

Descrizione: Alloca memoria per l'array. N.B.: non chiama in automatico la funzione che riempe l'array in quanto finirebbe per riempire anche il vettore risultante ($\vec{b}$).

Parametri di input:

- `size`: la dimensione del vettore

Parametri di output:

- `array`: il puntatore al vettore

Errors: Esce dal programma se non è possibile allocare memoria per il vettore.

3.2.7 La funzione *print_array*

```
void print_array(double **array, int size);
```

Descrizione: Stampa il vettore su std output.

Parametri di input:

- `array`: puntatore al vettore da stampare
- `size`: la dimensione del vettore

3.2.8 La funzione *fill_array*

```
void fill_array(double **array, int size);
```

Descrizione: Riempe randomicamente il vettore utilizzando valori double.

Parametri di input:

- `array`: puntatore al vettore da stampare
- `size`: la dimensione del vettore

Capitolo 4

Esempio d'uso

In questo capitolo mostriamo il pbs utilizzato e il relativo output. Semplicemente, nel pbs, iteriamo il numero di volte in cui deve essere eseguito il programma e con quanti thread dovrà essere eseguito. Quindi, per ciascun valore di $threads$ eseguiremo il programma dieci volte in modo da poter fare poi una media dei tempi ottenuti in seguito. Nel seguente esempio, inoltre, vi è il caso $N = 1k$, $M = 10k$.

4.1 File *pbs* utilizzato

```
#!/bin/bash

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

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

PBS_O_WORKDIR=$PBS_O_HOME/ProgettoMatVet
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_ENVIROMENT
echo PBS: job identifier is $PBS_JOBID
echo PBS: job name is $PBS_JOBNAME
echo PBS: node file is $PBS_NODEFILE
echo PBS: current home directory is $PBS_O_HOME
echo PBS: PATH = $PBS_O_PATH
echo -------------------------

export PSC_OMP_AFFINITY=TRUE

```

```
n=1000
m=10000

for threads in 1 2 4 8
do
    for i in {1..10}
    do
        echo ""
        echo "************************************************"
        echo "* Iteration="$i" - N="$n " - M="$m" - n_threads="$threads
        "    *"
        echo "************************************************"

        echo "Compilo..."
        gcc -fopenmp -lgomp -o $PBS_O_WORKDIR/result $PBS_O_WORKDIR/↩
    Main.c $PBS_O_WORKDIR/Lib.c -std=c99

        echo "Eseguo..."
        $PBS_O_WORKDIR/result $n $m $threads
    done
done
```

4.2 Esempio di output del *pbs*

```

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/2024/FLCGNN97K/ProgettoMatVet
PBS: execution mode is
PBS: job identifier is 4017506.torque02.scope.unina.it
PBS: job name is result
PBS: node file is /var/spool/pbs/aux//4017506.torque02.scope.unina.it
PBS: current home directory is /homes/DMA/PDC/2024/FLCGNN97K
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/2024/FLCGNN97K/bin
-------------------------

*******************************************************
* Iteration=1 - N=1000  - M=10000 - n_threads=1      *
*******************************************************
Compilo...
Eseguo...

Il tempo registrato e' stato 0.101026s

*******************************************************
* Iteration=2 - N=1000  - M=10000 - n_threads=1      *
*******************************************************
Compilo...
Eseguo...

Il tempo registrato e' stato 0.101060s

*******************************************************
* Iteration=3 - N=1000  - M=10000 - n_threads=1      *
*******************************************************
Compilo...
Eseguo...

Il tempo registrato e' stato 0.101067s

*******************************************************
* Iteration=4 - N=1000  - M=10000 - n_threads=1      *
*******************************************************
Compilo...
Eseguo...

Il tempo registrato e' stato 0.101066s

*******************************************************
* Iteration=5 - N=1000  - M=10000 - n_threads=1      *
*******************************************************
Compilo...
Eseguo...

Il tempo registrato e' stato 0.101070s

*******************************************************
* Iteration=6 - N=1000  - M=10000 - n_threads=1      *
*******************************************************
Compilo...
Eseguo...

Il tempo registrato e' stato 0.100989s

*******************************************************
```

```
* Iteration=7 - N=1000  - M=10000 - n_threads=1      *
*****************************************************
Compilo...
Eseguo...

Il tempo registrato e' stato 0.101451s

*****************************************************
* Iteration=8 - N=1000  - M=10000 - n_threads=1      *
*****************************************************
Compilo...
Eseguo...

Il tempo registrato e' stato 0.101192s

*****************************************************
* Iteration=9 - N=1000  - M=10000 - n_threads=1      *
*****************************************************
Compilo...
Eseguo...

Il tempo registrato e' stato 0.101341s

*****************************************************
* Iteration=10 - N=1000  - M=10000 - n_threads=1     *
*****************************************************
Compilo...
Eseguo...

Il tempo registrato e' stato 0.101072s

*****************************************************
* Iteration=1 - N=1000  - M=10000 - n_threads=2      *
*****************************************************
Compilo...
Eseguo...

Il tempo registrato e' stato 0.056981s

*****************************************************
* Iteration=2 - N=1000  - M=10000 - n_threads=2      *
*****************************************************
Compilo...
Eseguo...

Il tempo registrato e' stato 0.054695s

*****************************************************
* Iteration=3 - N=1000  - M=10000 - n_threads=2      *
*****************************************************
Compilo...
Eseguo...

Il tempo registrato e' stato 0.058713s
```

```
127
128 ******************************************************
129 * Iteration=4 - N=1000  - M=10000 - n_threads=2      *
130 ******************************************************
131 Compilo...
132 Eseguo...
133
134 Il tempo registrato e' stato 0.053989s
135
136 ******************************************************
137 * Iteration=5 - N=1000  - M=10000 - n_threads=2      *
138 ******************************************************
139 Compilo...
140 Eseguo...
141
142 Il tempo registrato e' stato 0.055620s
143
144 ******************************************************
145 * Iteration=6 - N=1000  - M=10000 - n_threads=2      *
146 ******************************************************
147 Compilo...
148 Eseguo...
149
150 Il tempo registrato e' stato 0.055779s
151
152 ******************************************************
153 * Iteration=7 - N=1000  - M=10000 - n_threads=2      *
154 ******************************************************
155 Compilo...
156 Eseguo...
157
158 Il tempo registrato e' stato 0.055502s
159
160 ******************************************************
161 * Iteration=8 - N=1000  - M=10000 - n_threads=2      *
162 ******************************************************
163 Compilo...
164 Eseguo...
165
166 Il tempo registrato e' stato 0.057320s
167
168 ******************************************************
169 * Iteration=9 - N=1000  - M=10000 - n_threads=2      *
170 ******************************************************
171 Compilo...
172 Eseguo...
173
174 Il tempo registrato e' stato 0.054970s
175
176 ******************************************************
177 * Iteration=10 - N=1000  - M=10000 - n_threads=2     *
178 ******************************************************
179 Compilo...
180 Eseguo...
```

```
181
182 Il tempo registrato e' stato 0.058441s
183
184 *******************************************************
185 * Iteration=1 - N=1000  - M=10000 - n_threads=4      *
186 *******************************************************
187 Compilo...
188 Eseguo...
189
190 Il tempo registrato e' stato 0.032171s
191
192 *******************************************************
193 * Iteration=2 - N=1000  - M=10000 - n_threads=4      *
194 *******************************************************
195 Compilo...
196 Eseguo...
197
198 Il tempo registrato e' stato 0.032300s
199
200 *******************************************************
201 * Iteration=3 - N=1000  - M=10000 - n_threads=4      *
202 *******************************************************
203 Compilo...
204 Eseguo...
205
206 Il tempo registrato e' stato 0.031027s
207
208 *******************************************************
209 * Iteration=4 - N=1000  - M=10000 - n_threads=4      *
210 *******************************************************
211 Compilo...
212 Eseguo...
213
214 Il tempo registrato e' stato 0.033420s
215
216 *******************************************************
217 * Iteration=5 - N=1000  - M=10000 - n_threads=4      *
218 *******************************************************
219 Compilo...
220 Eseguo...
221
222 Il tempo registrato e' stato 0.031769s
223
224 *******************************************************
225 * Iteration=6 - N=1000  - M=10000 - n_threads=4      *
226 *******************************************************
227 Compilo...
228 Eseguo...
229
230 Il tempo registrato e' stato 0.033563s
231
232 *******************************************************
233 * Iteration=7 - N=1000  - M=10000 - n_threads=4      *
234 *******************************************************
```

```
Compilo...
Eseguo...

Il tempo registrato e' stato 0.030364s

*********************************************************
* Iteration=8 - N=1000  - M=10000 - n_threads=4      *
*********************************************************
Compilo...
Eseguo...

Il tempo registrato e' stato 0.033865s

*********************************************************
* Iteration=9 - N=1000  - M=10000 - n_threads=4      *
*********************************************************
Compilo...
Eseguo...

Il tempo registrato e' stato 0.036133s

*********************************************************
* Iteration=10 - N=1000  - M=10000 - n_threads=4     *
*********************************************************
Compilo...
Eseguo...

Il tempo registrato e' stato 0.032641s

*********************************************************
* Iteration=1 - N=1000  - M=10000 - n_threads=8      *
*********************************************************
Compilo...
Eseguo...

Il tempo registrato e' stato 0.020226s

*********************************************************
* Iteration=2 - N=1000  - M=10000 - n_threads=8      *
*********************************************************
Compilo...
Eseguo...

Il tempo registrato e' stato 0.026161s

*********************************************************
* Iteration=3 - N=1000  - M=10000 - n_threads=8      *
*********************************************************
Compilo...
Eseguo...

Il tempo registrato e' stato 0.021348s

*********************************************************
```

```
289 * Iteration=4 - N=1000  - M=10000 - n_threads=8      *
290 ******************************************************
291 Compilo...
292 Eseguo...
293
294 Il tempo registrato e' stato 0.020873s
295
296 ******************************************************
297 * Iteration=5 - N=1000  - M=10000 - n_threads=8      *
298 ******************************************************
299 Compilo...
300 Eseguo...
301
302 Il tempo registrato e' stato 0.022938s
303
304 ******************************************************
305 * Iteration=6 - N=1000  - M=10000 - n_threads=8      *
306 ******************************************************
307 Compilo...
308 Eseguo...
309
310 Il tempo registrato e' stato 0.020112s
311
312 ******************************************************
313 * Iteration=7 - N=1000  - M=10000 - n_threads=8      *
314 ******************************************************
315 Compilo...
316 Eseguo...
317
318 Il tempo registrato e' stato 0.021786s
319
320 ******************************************************
321 * Iteration=8 - N=1000  - M=10000 - n_threads=8      *
322 ******************************************************
323 Compilo...
324 Eseguo...
325
326 Il tempo registrato e' stato 0.021582s
327
328 ******************************************************
329 * Iteration=9 - N=1000  - M=10000 - n_threads=8      *
330 ******************************************************
331 Compilo...
332 Eseguo...
333
334 Il tempo registrato e' stato 0.023111s
335
336 ******************************************************
337 * Iteration=10 - N=1000  - M=10000 - n_threads=8     *
338 ******************************************************
339 Compilo...
340 Eseguo...
341
342 Il tempo registrato e' stato 0.023881s
```

Capitolo 5

Analisi performance

In questo capitolo analizzeremo il tempo medio impiegato, lo speed up e l'efficienza al variare del numero di threads per ogni matrice di grandezza diversa. Detto ciò, mostriamo un breve excursus di quello che verrà affrontato in questo capitolo:

- Nel primo caso affronteremo calcoli inerenti una matrice quadrata $N \times N$ con $N = 1.000$, nel secondo e nel terzo caso effettueremo calcoli inerenti matrici rettangolari $M \times N$ rispettivamente con $M = 1.000$ e $N = 10.000$ ed $M = 10.000$ e $N = 1.000$. Infine considereremo un'altra matrice quadrata $N \times N$ con $N = 10.000$
- Come conseguenza alle grandezze della matrice avremo un vettore che dovrà essere moltiplicato con essa e che quindi avrà una lunghezza pari a N pari alla lunghezza delle righe della matrice
- Per ognuno dei prodotti matrice vettore avremo un numero di threads T_i con $i \in 1, 2, 4, 8$
- Per calcolare il tempo medio, per ciascun caso il programma è stato eseguito esattamente 10 volte per considerare, appunto, la media aritmetica
- Una volta ricavato il tempo medio è stato possibile calcolare lo speed up e l'efficienza mediante le seguenti formule:
  - Speed up $S(P) = \frac{T(1)}{T(P)}$, ricordando che lo speed up ideale è uguale a P
  - Efficienza $E(P) = \frac{S(P)}{P}$, tenendo presente che l'efficienza ideale è uguale ad 1

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

5.1 Analisi con $N = 10^3 \times M = 10^3$

N. threads	T1	T2	T4	T8
	0.010847	0.005660	0.008392	0.011685
	0.011620	0.008422	0.006811	0.007719
	0.010003	0.006087	0.005959	0.005919
	0.010038	0.008647	0.005615	0.009207
	0.009953	0.008522	0.006380	0.007775
	0.010203	0.006684	0.005671	0.007197
	0.010045	0.006762	0.005237	0.010817
	0.011616	0.007646	0.005746	0.011288
	0.010565	0.006452	0.005251	0.005362
	0.011600	0.007454	0.005909	0.010531
Tempo medio	0.010649	0.0072336	0.0060971	0.00875
Speed up	1	1.47215	1.74656	1.217028
Efficienza	1	0.736075	0.43664	0.1521285

Tabella 5.1: Analisi con $N = 10^3 \times M = 10^3$

5.2 Analisi con $N = 10^3 \times M = 10^4$

N. threads	T1	T2	T4	T8
	0.101026	0.056981	0.032171	0.020226
	0.101060	0.056981	0.032300	0.026161
	0.101067	0.058713	0.031027	0.021348
	0.101066	0.053989	0.033420	0.020873
	0.101070	0.055620	0.031769	0.022938
	0.100989	0.055779	0.033563	0.020112
	0.101451	0.055502	0.030364	0.021786
	0.101192	0.057320	0.033865	0.021582
	0.101341	0.054970	0.036133	0.023111
	0.101072	0.058441	0.032641	0.023881
Tempo medio	0.1011334	0.0564296	0.0327253	0.0222018
Speed up	1	1.792205	3.090374	4.555189
Efficienza	1	0.896102	0.772593	0.569399

Tabella 5.2: Analisi con $N = 10^3 \times M = 10^4$

5.3 Analisi con $N = 10^4 \times M = 10^3$

N. threads	T1	T2	T4	T8
	0.101041	0.058218	0.028934	0.022579
	0.101393	0.056603	0.031636	0.023441
	0.101036	0.055800	0.032633	0.022323
	0.101118	0.056460	0.029878	0.020894
	0.101034	0.057856	0.032125	0.023210
	0.101052	0.057713	0.032395	0.019759
	0.101063	0.054479	0.030382	0.020057
	0.101144	0.056354	0.029941	0.022445
	0.101257	0.057111	0.034656	0.020571
	0.101065	0.057049	0.031425	0.020462
Tempo medio	0.1011203	0.0567643	0.0314005	0.0215741
Speed up	1	1.7814066	3.2203404	4.6871155
Efficienza	1	0.8907033	0.8050851	0.585889

Tabella 5.3: Analisi con $N = 10^4 \times M = 10^3$

5.4 Analisi con $N = 10^4 \times M = 10^4$

N. threads	T1	T2	T4	T8
	1.011132	0.510350	0.260784	0.144077
	1.010608	0.511326	0.261689	0.143054
	1.011336	0.512241	0.263499	0.140835
	1.011232	0.512370	0.263833	0.141532
	1.011273	0.510142	0.260518	0.140402
	1.010300	0.514010	0.260347	0.139855
	1.009511	0.512794	0.262592	0.140960
	1.011312	0.510953	0.264974	0.140790
	1.010759	0.512619	0.261403	0.142829
	1.010497	0.511654	0.260561	0.141618
Tempo medio	1.010796	0.5118459	0.26202	0.1415952
Speed up	1	1.974805	3.857705	7.138631
Efficienza	1	0.9874025	0.96442625	0.892328875

Tabella 5.4: Analisi con $N = 10^4 \times M = 10^4$

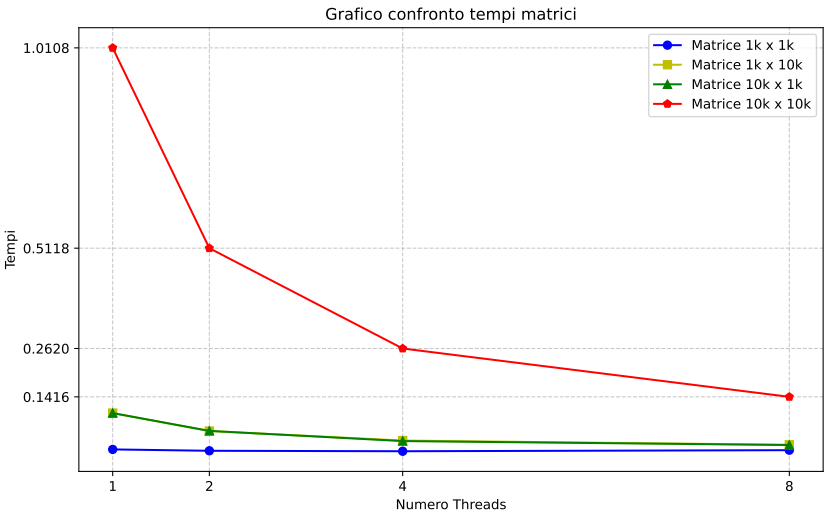


Figura 5.1: Plot Confronto del tempo al variare del numero dei thread

Nel grafico 5.1 si evince quanto i tempi della matrice $1k \times 1k$, con l'aumentare dei thread, non sempre diminuiscono a causa della grandezza ridotta della matrice, in particolare come si evince anche dalla tabella 5.1 i tempi medi diminuiscono fino a 4 thread, per 8 thread, invece, il tempo medio aumenta di circa 0.0027s. Nelle altre matrici i tempi diminuiscono all'aumentare dei thread ed in particolare con la matrice 5.4 la riduzione dei tempi è superiore in quanto si sfrutta meglio il parallelismo.

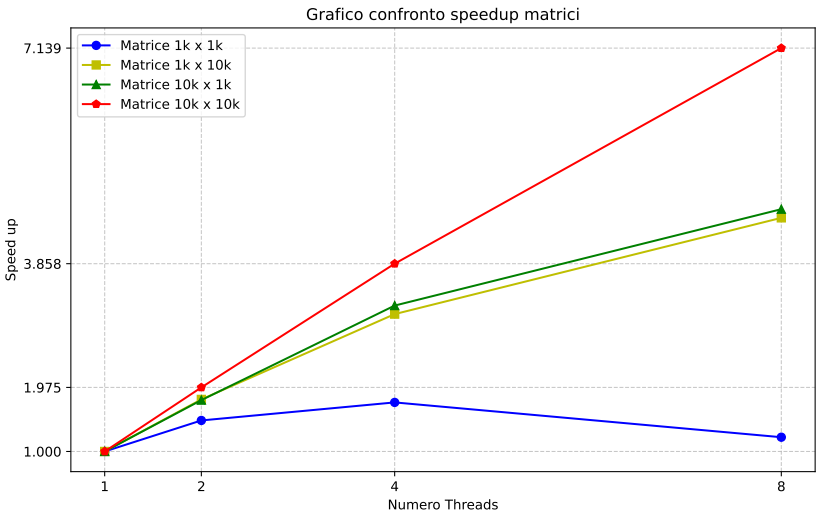


Figura 5.2: Plot Confronto dello speed up al variare del numero dei thread

Nonostante i tempi diminuiscano come nel grafico 5.1, ciò non vuol dire che lo speedup si avvicini a quello ideale, anzi si può notare come già dal se-

condo thread sia uno spreco calcolare in parallelo una matrice di dimensioni $1k \times 1k$. Nel caso della matrice $10k \times 10k$, invece, utilizzare più thread risulta vantaggioso, in quanto si può notare che all'aumentare del numero dei thread lo speedup è vicino a quello ideale. Negli altri casi, invece, si può notare che solo fino a 4 thread ha senso lavorare in parallelo dal momento che con 8 lo speedup si allontana di molto da quello ideale.

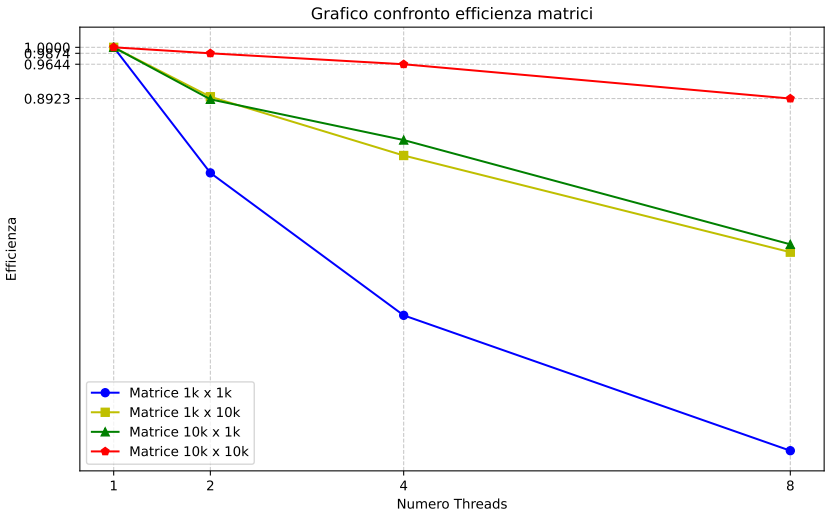


Figura 5.3: Plot Confronto dell'efficienza al variare del numero dei thread

Invece, in questo grafico, si nota che all'aumentare della dimensione della matrice e del numero dei thread l'efficienza degrada. Tuttavia, risulta più sensibile nel momento in cui la dimensione della matrice non è elevata ed il numero dei thread è alto.

In conclusione, possiamo notare che nel caso $N = 10k$ e $M = 10k$ vale la pena utilizzare il parallelismo, in quanto speedup ed efficienza sono prossimi a quelli ideali, ergo all'aumentare della dimensione del problema risulta vantaggioso sfruttare il parallelismo.

Capitolo 6

Source code

6.1 *Main.c*

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

# include "Lib.h"

int main(int argc, char *argv[])
{
    int N, M;       // numero di righe e colonne della matrice di valori numerici
    int n_threads;  // numero di thread da usare per il calcolo in parallelo
    double *A;      // matrice di elementi fornita in input
    double *x;      // vettore di elementi generato randomicamente
    double *b;      // vettore risultante
    double start, end;
    struct timeval time;

    // get input data and initialize the matrix
    read_input(argc, argv, &N, &M, &n_threads);
    initialize_matrix(&A, N, M);
    initialize_array(&x, M);
    fill_array(x, M);
    initialize_array(&b, N);

    // set max number of threads
    omp_set_num_threads(n_threads);

    gettimeofday(&time, NULL);
    start = time.tv_sec + (time.tv_usec / 1000000.0);

    // compute Ax=b in parallel
    matxvet(A, N, M, x, b);

    gettimeofday(&time, NULL);
```

```
35      end = time.tv_sec + (time.tv_usec / 1000000.0);
36
37      // for debug: print of Ax=b
38      // print_matrix(A, N, M);
39      // print_array(x, M);
40      // printf("Result of Ax=b:\n");
41      // print_array(b, N);
42
43      printf("\n");
44      printf("Il tempo registrato e' stato %fs\n", end - start);
45      return 0;
46  }
```

6.2 *Lib.c*

```
1   # include <stdio.h>
2   # include <stdlib.h>
3   # include <omp.h>
4   # include <time.h>
5   # include <sys/time.h>
6
7   # include "Lib.h"
8
9   void matxvet(double *A, int N, int M, double *x, double *b){
10      int i,j;
11      #pragma omp parallel for default(none) shared(N, M, A, x, b) ↩
        private(i, j)
12      // prodotto matrice vettore
13      for (i = 0; i < N; i++){
14          for (j = 0; j < M; j++){
15              b[i] += A[i * M + j] * x[j];
16          }
17          //printf("Hello from thread %d, nthreads %d\n", ↩
        omp_get_thread_num(), omp_get_num_threads());
18      }
19  }
20
21  void initialize_matrix(double **A, int N, int M) {
22      *A = (double *)calloc(N * M, sizeof(double));
23
24      if (*A == NULL) {
25          fprintf(stderr, "Error allocating memory for the
26          'matrix' array!\n");
27          exit(EXIT_FAILURE);
28      }
29
30      // fill matrix randomly
31      fill_matrix(*A, N, M);
32  }
33
34
35  void fill_matrix(double *A, int rows, int cols){
```

```
36      srand(time(NULL));
37
38      int index = 0, value = 0;
39      for(int i = 0; i < rows; i++){
40          for(int j = 0; j < cols; j++)
41              A[index++] = ((double) rand() / RAND_MAX) * (100 - 1) + 1;
42      }
43  }
44
45
46  void print_matrix(double *matrix, int rows, int cols) {
47      printf("[");
48      for (int i = 0; i < rows; i++) {
49          (i == 0) ? printf(" ") : printf("  ");
50          for (int j = 0; j < cols; j++) {
51              // Calcola l'indice nell'array
52              int index = i * cols + j;
53
54              printf("%2f", matrix[index]);
55
56              // Aggiungi la virgola se non e' l'ultimo elemento della ↩
    riga
57              if (j < cols - 1) {
58                  printf(", ");
59              }
60          }
61
62          // Aggiungi il punto e virgola se non e' l'ultima riga
63          if (i < rows - 1) {
64              printf("\n");
65          }
66      }
67      printf(" ]\n");
68  }
69
70
71  void read_input(int argc, char **argv, int *N, int *M, int *n_threads){
72      if(argc != 4){
73          fprintf(stderr, "Usage: <rows matrix> <column matrix>
74          <n. threads>\n");
75          fprintf(stderr, "Exit from program...");
76          exit(EXIT_FAILURE);
77      }
78
79      *N = atoi(argv[1]);
80      *M = atoi(argv[2]);
81      *n_threads = atoi(argv[3]);
82
83      if (*N < 1 || *M < 1){
84          fprintf(stderr, "Usage: <N> (risp. <M>) must be greater
85          or equal than 1!\nExit from program...\n");
86          exit(EXIT_FAILURE);
87      }
88
```

```
89      if(*n_threads < 1){
90          fprintf(stderr, "Usage: <n_threads> must be greater
91          or equal than 1!\nExit from program...\n");
92          exit(EXIT_FAILURE);
93      }
94  }
95
96
97  void initialize_array(double **array, int size){
98      // array malloc
99      *array = (double *)calloc(sizeof(double), size);
100     if(*array == NULL){
101         fprintf(stderr, "Errore nell'allocazione della memoria
102         per l'array 'elements'!\n");
103         exit(EXIT_FAILURE);
104     }
105 }
106
107
108 void fill_array(double *array, int size){
109     srand(time(NULL));
110
111     for(int i = 0; i < size; i++){
112         array[i] = ((double) rand() / RAND_MAX) * (100 - 1) + 1;
113     }
114 }
115
116
117 void print_array(double *array, int size){
118     printf("Vettore: [");
119     for(int i = 0; i < size; i++)
120         printf("%f ", array[i]);
121     printf("]\n");
122 }
```

6.3 *Lib.h*

```
1  #ifndef LIB_H
2  #define LIB_H
3
4  /**
5   * @brief Compute the product Ax=b using threads.
6   *
7   * @param A Pointer to the matrix.
8   * @param N Number of rows of A.
9   * @param M number of columns of A.
10  * @param x Pointer to array.
11  * @param b Result of product.
12  */
13 void matxvet(double *A, int N, int M, double *x, double *b);
14
15
```

```c
/**
 * @brief Fills a 1D matrix with random values.
 *
 * @param matrix Pointer to the matrix.
 * @param row Number of rows.
 * @param col Number of columns.
 */
void fill_matrix(double *matrix, int row, int col);

/**
 * @brief Prints the contents of a 1D matrix.
 *
 * @param matrix Pointer to the matrix.
 * @param row Number of rows.
 * @param col Number of columns.
 */
void print_matrix(double *matrix, int row, int col);

/**
 * @brief Reads input arguments and exits from program if they are incorrect.
 *
 * @param argc Number of command-line arguments.
 * @param argv Array of command-line argument strings.
 * @param N Pointer to the number of matrix rows variable.
 * @param M Pointer to the number of matrix columns variable.
 * @param n_thread Pointer to the number of thread to use.
 */
void read_input(int argc, char **argv, int *N, int *M, int *n_threads);

/**
 * @brief Initializes a 1D matrix with random values.
 *
 * @param matrix Pointer to the matrix.
 * @param N Number of rows.
 * @param M Number of columns.
 */
void initialize_matrix(double **matrix, int N, int M);

/**
 * @brief Initializes a 1D array with values based on the index.
 *
 * @param array Pointer to the array.
 * @param size Size of the array.
 */
void initialize_array(double **array, int size);

/**
 * @brief Prints the contents of a 1D array.
 *
 * @param array Pointer to the array.
 * @param size Size of the array.
 */
void print_array(double *array, int size);
```

```
69
70 /**
71  * @brief Initializes a 1D array with random values.
72  *
73  * @param array Pointer to the array.
74  * @param size Size of the array.
75  */
76 void fill_array(double *array, int size);
77
78 #endif
```