ELABORATO II

Prodotto matrice-vettore su architettura MIMD-SM

UNIVERSITÀ DEGLI STUDI DI NAPOLI "FEDERICO II"
ROBERTO BASILE GIANNINI
N97000340

Descrizione del problema

Si vuole implementare un algoritmo per il calcolo del prodotto matrice-vettore in ambiente di calcolo parallelo su architettura MIMD a memoria condivisa, utilizzando la libreria OpenMP. Non essendo stata espressa esplicitamente una richiesta sugli input da inserire, si è scelto di dare la libertà all'utente di scegliere le dimensioni della matrice coinvolta (parametri che verranno opportunamente controllati). Tuttavia, in queste sperimentazioni verranno considerate solo matrici quadrate. I valori (dei numeri interi) della matrice e del vettore vengono generati random in un intervallo che va da 0 a 9: questa scelta è stata presa esclusivamente per garantire un risultato a video più chiaro per l'individuazione delle colonne.

Input e Output

Il software richiede in ingresso i seguenti parametri:

- *int mat_rig*, il numero di righe della matrice, che dovrà essere strettamente maggiore di 0;

- *int mat_col*, il numero di colonne della matrice che dovrà essere strettamente maggiore di 0.

Gli input vengono specificati tramite il file *.pbs*, indicandoli con lo stesso ordine col quale sono stati presentati:

mat_rig mat_col

L'eventuale gestione degli errori legati alla coerenza dei parametri inseriti viene garantita da specifici controlli effettuati a monte delle operazioni che li coinvolgono: nella sezione dedicata agli *Indicatori di errore* se ne parlerà in maggiore dettaglio. Come anticipato, gli elementi della matrice e del vettore non vengono inseriti dall'utente, ma generati random.

Gli output a video sono i seguenti:

- messaggi relativi allo stato dell'algoritmo;

- eventuali messaggi di errore;

- matrice e vettore generati;

- il vettore risultato (di dimensione *mat_rig* e di elementi *int*);
- il tempo di esecuzione e il numero di thread coinvolti.

Per ogni messaggio di errore è previsto uno specifico codice.

Descrizione dell’algoritmo

L’idea di base è quella di sfruttare gli strumenti messi a disposizione dalla libreria OpenMP al fine di realizzare un algoritmo parallelo tale da garantire alte prestazioni al crescere della dimensione del problema. In questo caso, essendo un’architettura SM, non c’è bisogno di gestire la comunicazione tra processori: vengono sfruttati i core di un singolo processore, i quali accedono ad un’area di memoria condivisa. Una rappresentazione fortemente astratta dell’algoritmo è la seguente:

```
INIZIO

//0.
    Dichiarazioni e lettura input

//1.
    Controllo input ed eventuale gestione errori

//2.
    Inizializzazione di A e x

//3.
    Calcolo Ax = b

//4.
    Stampa del risultato

//5.
    Stampa dei tempi

//6.
    Finalizzazione

FINE
```

I passi che vanno da *0* a *2* e quelli da *4* a *6* vengono eseguiti sequenzialmente, dal momento che la regione parallela è presente solo in occasione del calcolo del prodotto matrice-vettore. In dettaglio, nel passo 0 avviene sostanzialmente la dichiarazione delle strutture dati e la lettura degli input, la cui validità verrà controllata nel passo 1. Successivamente, allocate le strutture dati, nel passo 2 avviene la loro inizializzazione e la stampa a video del loro contenuto: come anticipato, gli interi della matrice e del vettore sono generati casualmente.

Raggiunto il passo 3 si entra nella fase di calcolo del prodotto matrice-vettore. Prima di entrare nella regione parallela, inizia la misura del tempo di esecuzione, misura che terminerà con la fine della regione parallela stessa.

```
1.  ...
2.  //3.
3.      Allocazione vettore res (mat_rig elementi)
4.
5.      Inizia conteggio tempo
6.      Inizio regione parallela
7.          Variabili condivise: mat_rig, mat_col, A, x, res
8.          Variabili private: contatori
9.
10.         Per ogni thread
11.             Calcola alcuni dei mat_rig elementi di res (condiviso)
12.             Memorizza risultati in res (condiviso)
13.
14.     Fine regione parallela
15.     Fine conteggio tempo
16.
17.     Misura intervallo di tempo
18. //
19.  ...
```

Osserviamo le variabili coinvolte nella regione parallela:

- *Variabili condivise:*
    - Matrice *A*, dal momento che viene solo letta e non c'è dunque bisogno di copiarla più volte;
    - *mat_rig*, *mat_col e* vettore *x*, per medesime osservazioni;

    - vettore *res*, coinvolto in operazioni di scrittura, ma bisogna considerare che ad una specifica porzione del vettore accederà *esclusivamente* uno *specifico thread*. Quindi, diversi thread coinvolti accederanno ad aree del vettore diverse, non dovendo gestire degli accessi concorrenti. Ciò è possibile solo se il contatore del *for* esterno (che stabilisce su quale porzione di *res* si sta scrivendo) è privato.

È importante osservare che se queste variabili venissero definite come private, verrebbero de-inizializzate (ed in particolare il vettore *res* non sarebbe accessibile dall'esterno, ed il suo contenuto verrebbe perso al termine della regione parallela).

- *Variabili private:*

    - *Contatori dei cicli for*, sia per quanto osservato sopra, sia per evitare che thread possano modificare contatori di altri thread, generando risultati imprevedibili.

Nei passi 4, 5 e 6 avvengono rispettivamente la stampa del risultato, la stampa del tempo di esecuzione misurato e la deallocazione delle risorse.

Di seguito viene descritta in maggior dettaglio (mantenendo comunque una forte astrazione) la soluzione implementata:

```
1.  //I codici di errore sono indicati nella sezione relativa agli Indicatori di errore.
2.  //La stringa di input si presenta come mat_rig, mat_col
3.
4.
5.  INIZIO
6.
7.  //0.
8.      Dichiarazione e lettura input
9.
10.
11. //1.
12.     Se mat_rig non è valido
13.         Stampa errore E0
14.         Gestisci rispettivo errore
15.
16.     Se mat_col non è valido
17.         Stampa errore E1
18.         Gestisci rispettivo errore
19.
20.     Allocazione A e x
21. //
22.
23. //2.
24.     Genera matrice A (interi random)
25.     Stampa matrice A
26.
27.     Genera vettore x (interi random)
28.     Stampa vettore x
29. //
30.
31. //3.
32.     Allocazione vettore res (mat_rig elementi)
33.
34.     Inizia conteggio tempo
35.     Inizio regione parallela
36.         Variabili condivise: mat_rig, mat_col
37.                                  A, x, res
38.         Variabili private: contatori
39.
40.         Per ogni thread
41.             Calcola alcuni dei mat_rig elementi di res (condiviso)
42.             Memorizza risultati in res (condiviso)
43.     Fine regione parallela
44.     Fine conteggio tempo
45.
46.     Misura intervallo di tempo
47. //
48.
49. //4.
50.     Stampa del risultato
```

```
51.
52.
53. //5.
54.     Stampa dei tempi
55.
56.
57. //6.
58.     Dealloca risorse
59.
60. FINE
```

Indicatori di errore

Gli errori previsti riguardano esclusivamente dati di input non validi. Indipendentemente dall'errore, si è deciso di procedere sempre con l'esecuzione del programma, prevedendo una precisa gestione degli input non validi: questi verranno sostituiti con dei valori compatibili. Non è previsto alcun meccanismo di interruzione forzata dell'algoritmo.

Di seguito vengono riportate le diverse circostanze di errore ed indicata la rispettiva gestione:

- *Numero di righe* ≤ 1

  In questo caso, si è deciso di sostituire il valore posto in input con la dimensione *20*, stampando a video la segnalazione dell'errore e la sua gestione.

```
1. //controllo validità mat_rig
2. if (mat_rig < 1) {
3.     printf ("[E0] Numero %d di righe non valido. Valore default: 20.\n", mat_rig);
4.     mat_rig = 20;
5. }
```

- *Numero di colonne* ≤ 1

  In questo caso, si è deciso di sostituire il valore posto in input con la dimensione *20*, stampando a video la segnalazione dell'errore e la sua gestione.

```
1.  //controllo validità mat_col
2.  if (mat_col < 1) {
3.      printf ("[E1] Numero %d di colonne non valido. Valore default: 20.\n", mat_col);
4.      mat_col = 20;
5.  }
```

È presente un file *.pbs* simulante input tali da generare condizioni di errore. Per l'esattezza, il file *matvet_E0_E1.pbs* simula gli errori *E0* e *E1*. È presente il relativo file *.out* con i risultati ottenuti.

Subroutine

Sono stati utilizzati strumenti messi a disposizione dalla libreria di OpenMP ed una funzione scritta personalmente. Per generare la regione parallela, sono state utilizzate le direttive OpenMP, specificando costrutti e clausole utili al caso. In particolare, nel caso in esame:

```
1.  #pragma omp parallel for default(none) shared(mat_rig, mat_col, A, x, res) private(i,j)
```

Si è adottato il costrutto composto *parallel for* dal momento che nel costrutto *parallel* era presente solo il costrutto *for*. In questo caso, al chiudersi della parentesi del *for* esterno (relativo dunque al costrutto *for*) si chiuderà anche la parallel, giungendo alla barriera di sincronizzazione in cui tutti i thread terminano ad eccezione del master-thread. La clausola *default(none)* implica la necessità di esplicitare quali saranno le variabili condivise e quali quelle private: le motivazioni delle rispettive scelte sono state indicate nella sezione relativa alla descrizione dell'algoritmo.

Per quanto concerne la funzione scritta personalmente:

- *int* matxvet(int mat_rig, int mat_col, int* x, int* A, double* tdiff),* funzione demandata al calcolo vero e proprio del prodotto matrice-vettore. Presenta come valori di ingresso le dimensioni della matrice *A* e il vettore *x*, più un riferimento ad una variabile *tdiff* nella quale verrà posta la misura del tempo. Viene restituito un riferimento *int* al vettore risultato. In questa funzione è presente la regione parallela, costruita per mezzo delle direttive OpenMP.

Analisi dei tempi

La misura dei tempi è stata effettuata sfruttando la libreria *<sys/time.h>*, misurando il tempo subito prima l'inizio della regione parallela e memorizzando il risultato nella variabile *double ti*, e subito dopo la fine della regione parallela memorizzando il risultato nella variabile *double tf*. La misura del tempo di esecuzione è data dalla differenza *tf – ti*, stampata a video insieme al numero di thread coinvolti (così da agevolare la raccolta dei dati). Ogni caso di misura è strutturato nel seguente modo:

dimensione problema, numero dei thread, tempo di esecuzione

Sono state considerate 4 diverse dimensioni del problema, ed il calcolo è stato effettuato con 1, 2, 4 e 8 thread. Ogni misura è stata ripetuta tre volte, riportando come tempi la media delle tre misure.

Dimensione matrice 10x10

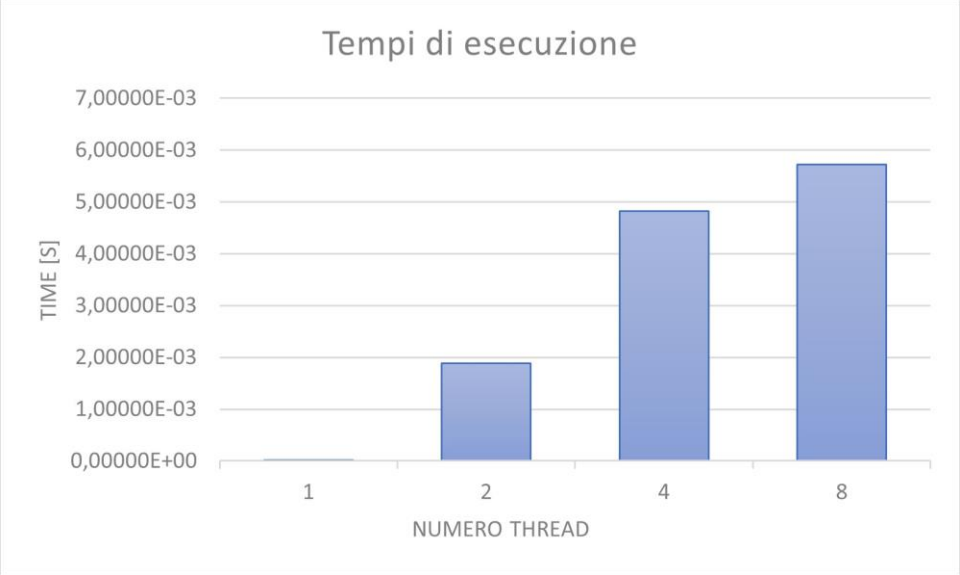
# thread	Time[s]
1	1,40667E-05
2	1,88895E-03
4	4,82600E-03
8	5,71975E-03

Notiamo subito un'anomalia. Il tempo di esecuzione per le soluzioni con un numero di thread superiore strettamente ad 1, risultano peggiori rispetto al caso sequenziale, il che spinge ad indagare sulla possibile causa di questo fenomeno. Quello che ci si aspetta, infatti, è che il tempo di esecuzione sia minore al crescere del numero di thread. Non essendo questo un caso distribuito, non ci si aspetta un overhead dovuto alla comunicazione tra processori, dal momento che il processore è unico e che la memoria è condivisa (tra i thread). Per questa ragione, sono stati condotti degli esperimenti al fine di misurare i tempi funzionali ai costrutti relativi alla regione parallela, nella fattispecie il costrutto composto *parallel for*. Si è considerata la regione parallela prevedendo solo il ciclo *for* esterno e nessuna operazione interna a quest'ultimo, misurando il tempo come indicato nell'introduzione. Si è considerato un numero di thread pari a 8 e la misurazione è stata effettuata per le 4 dimensioni del problema:

Dimensione della matrice	Time [s]
10 x 10	6.25419E-03
100 x 100	6.29009E-03
1000 x 1000	6.60408E-03
10000 x 10000	8.53705E-03

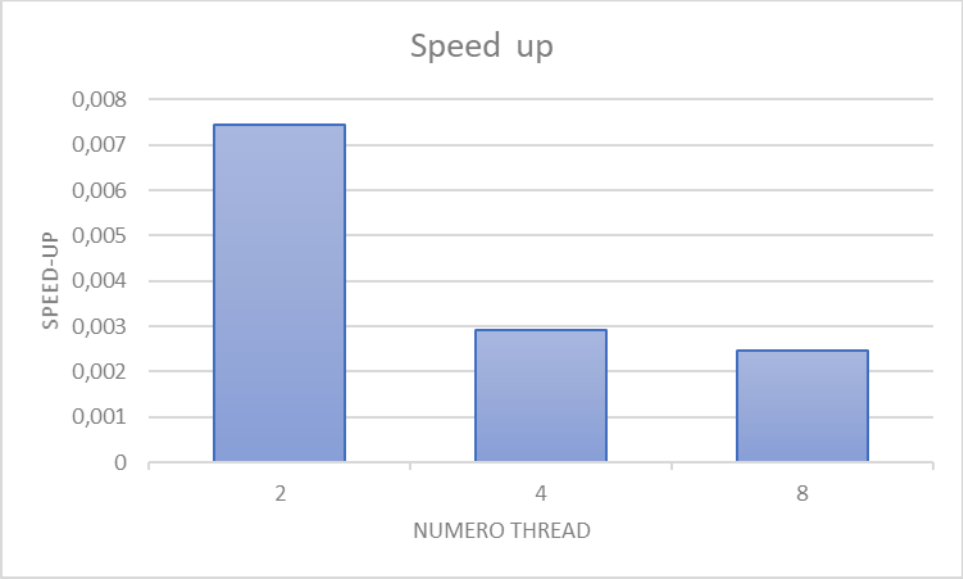
Si osserva come per tutte e 4 le dimensioni del problema, il tempo di esecuzione (che ricordo non riguardare alcuna operazione interna al ciclo for esterno) è sempre nell'ordine di 10^{-3} secondi, comportando di fatti un limite inferiore per i tempi di esecuzione nel caso di un numero di thread superiore ad 1. Infatti, quando il numero di thread è pari a 1, dunque nel caso sequenziale, la stessa misura restituisce (per dimensioni del problema ridotte) un tempo nell'ordine di 10^{-6} secondi. Per completezza, medesime verifiche sono state effettuate nel caso di 2 e 4 thread, osservando la stessa circostanza (non vengono riportate per ragioni di brevità). L'anomalia evidenziata dalle misure spinge a concludere che per dimensioni del problema molto ridotte, non vale la pena utilizzare un numero di thread maggiore ad 1, dal momento che le prestazioni risultano estremamente degradate.

Di seguito il grafico dei tempi di esecuzione del prodotto matrice-vettore:

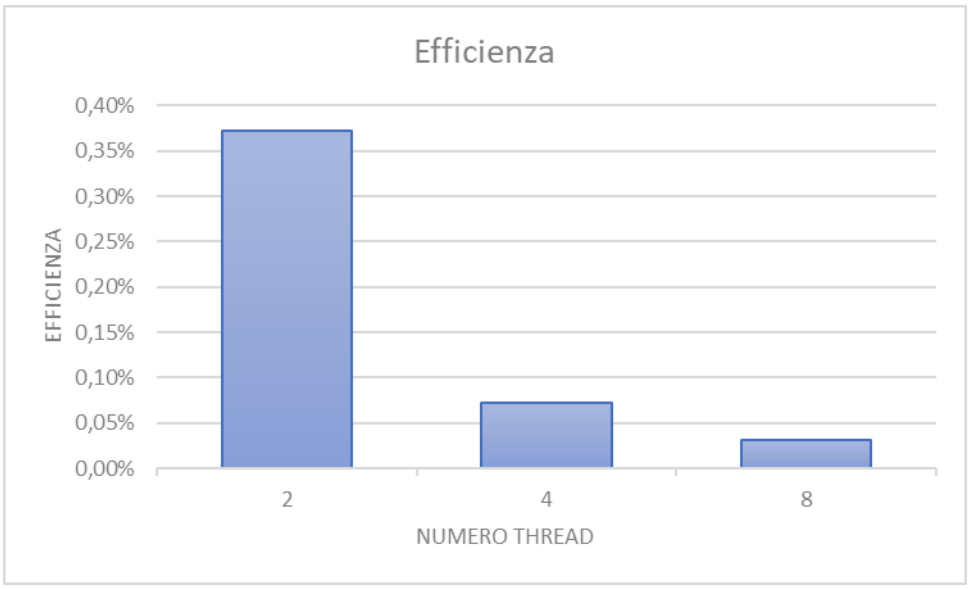


Per quanto riguarda il calcolo di speed-up ed efficienza, vengono di seguito riportati tabelle e grafici, osservando valori estremamente bassi e lontani dai riferimenti ideali.

# thread	Speed-up effettivo	Speed-up ideale
2	0,007	2
4	0,003	4
8	0,002	8



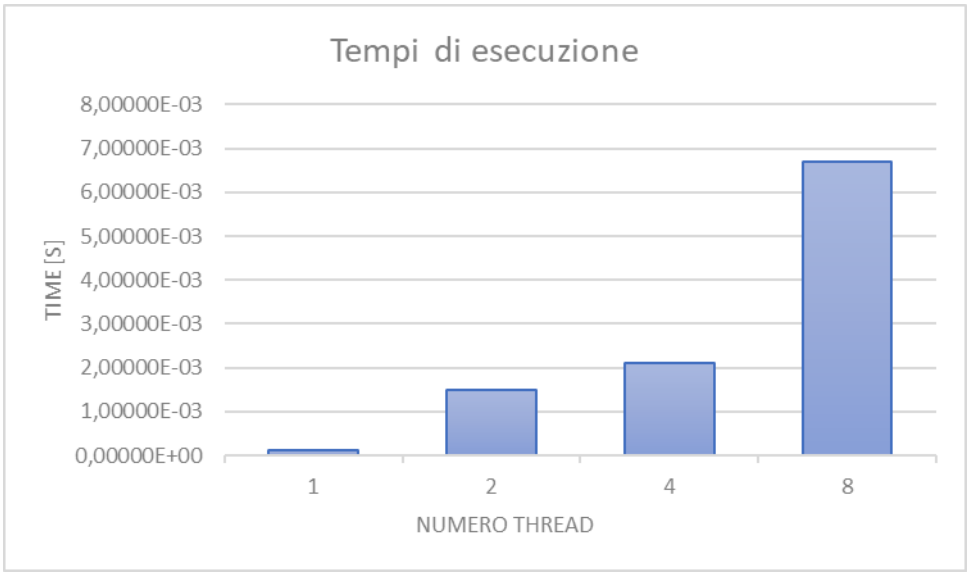
# thread	Efficienza
2	0,37%
4	0,07%
8	0,03%



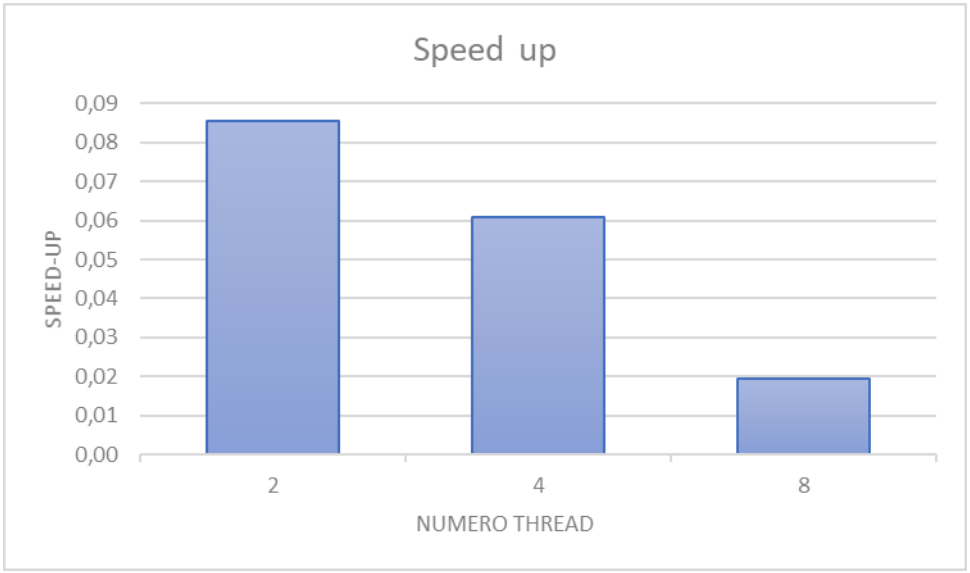
Dimensione matrice 100x100

# thread	Time[s]
1	1,28985E-04
2	1,50609E-03
4	2,11501E-03
8	6,68003E-03

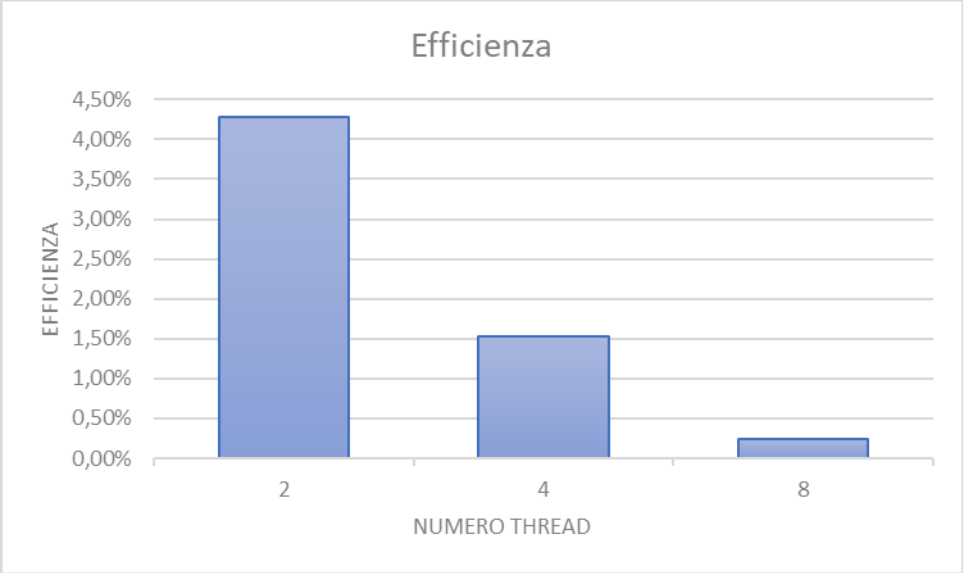
Come nel caso precedente, anche per questa dimensione del problema osserviamo le medesime anomalie riscontrate.



# thread	Speed-up effettivo	Speed-up ideale
2	0,09	2
4	0,06	4
8	0,02	8

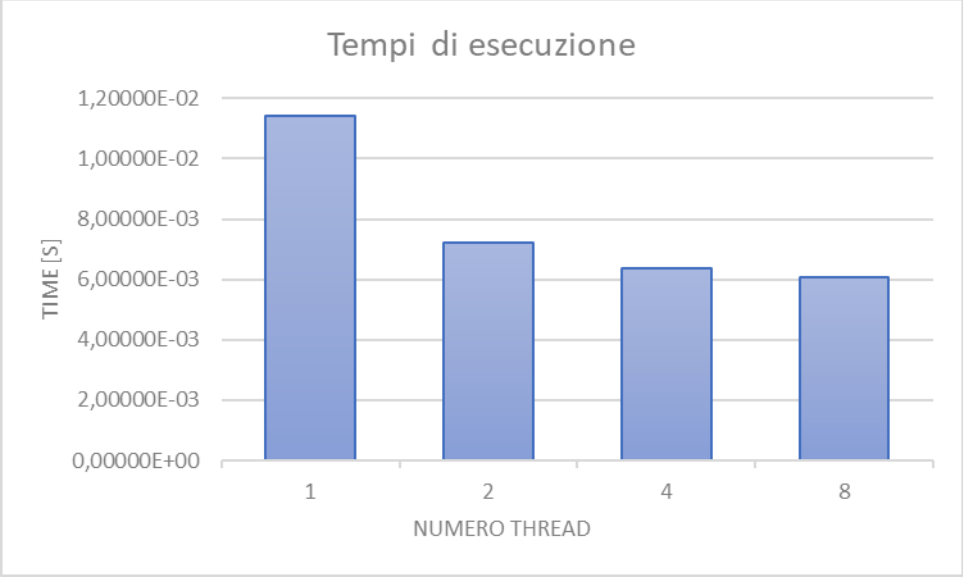


# thread	Efficienza
2	4,28%
4	1,52%
8	0,24%



Dimensione matrice 1000x1000

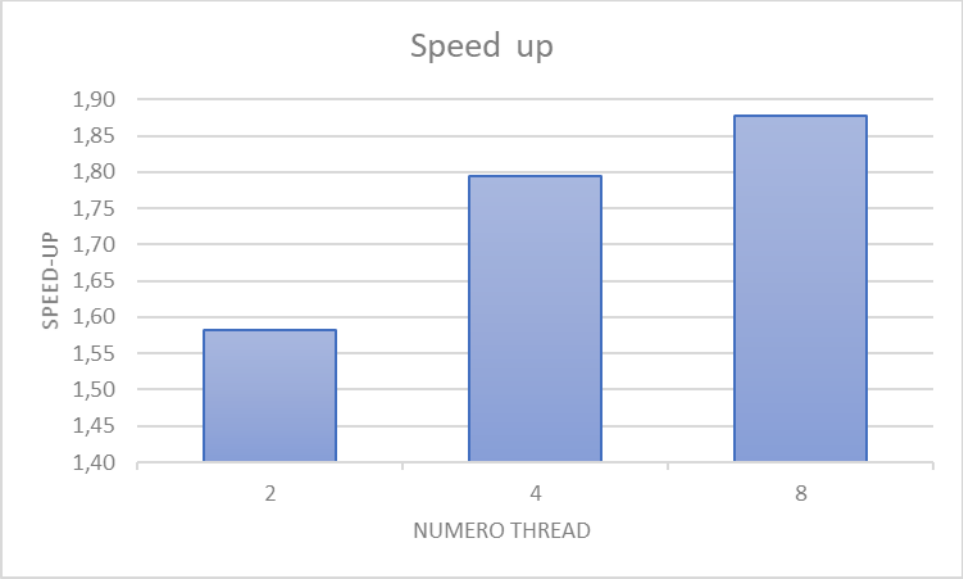
# thread	Time[s]
1	1,14060E-02
2	7,20811E-03
4	6,35719E-03
8	6,07697E-03



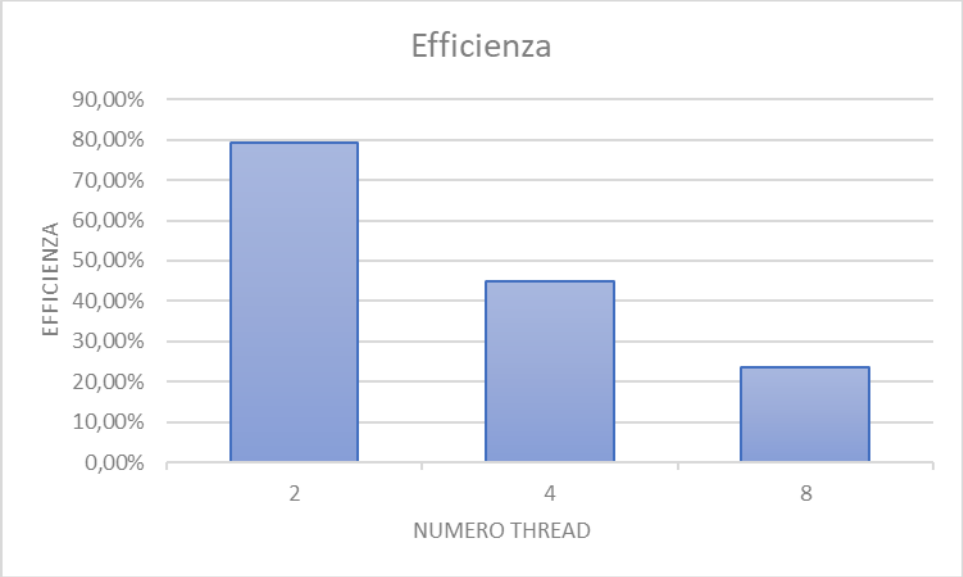
Osserviamo come per questa dimensione del problema la presenza di un numero di thread superiore ad 1 è giustificata. I tempi con 2, 4 e 8 thread risultano minori di almeno un ordine di grandezza rispetto al caso di un solo thread. Il tempo tra 4 e 8 thread, tuttavia, è pressocché uguale.

Calcoliamo speed-up ed efficienza:

# thread	Speed-up effettivo	Speed-up ideale
2	1,58	2
4	1,79	4
8	1,88	8



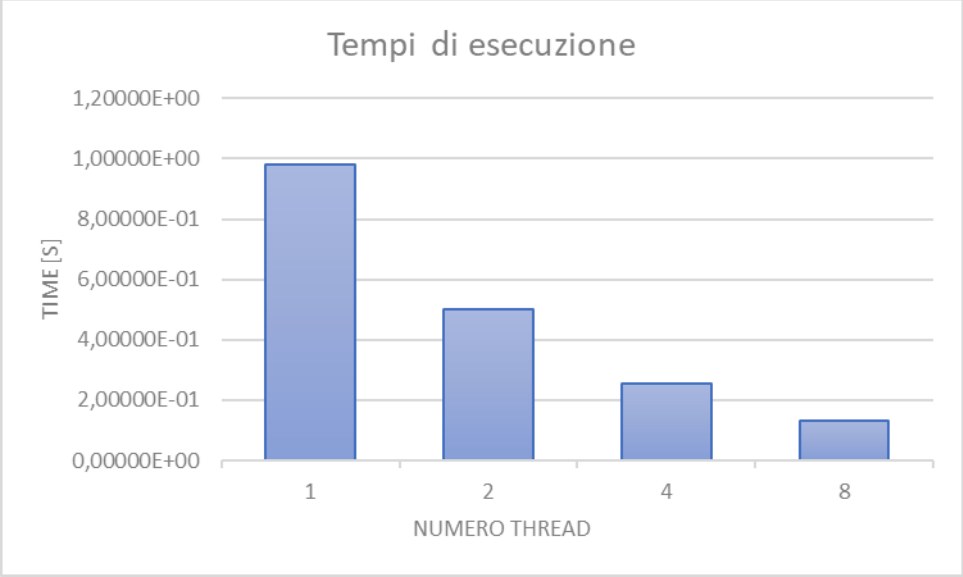
# thread	Efficienza
2	79,12%
4	44,85%
8	23,46%



Per 2 thread lo speed-up è molto vicino a quello ideale, al punto da registrare un'efficienza di quasi l'80%. La soluzione con 8 thread continua ad avere un'efficienza ancora bassa, mentre quella con 4 thread è prossima al 50%.

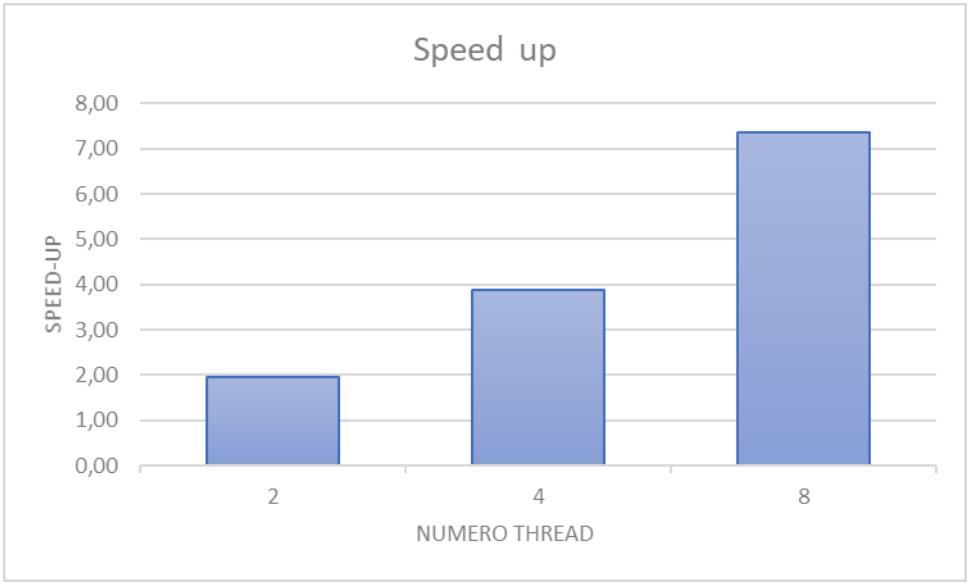
Dimensione matrice 10000x10000

# thread	Time[s]
1	9,79863E-01
2	5,02145E-01
4	2,53616E-01
8	1,33084E-01

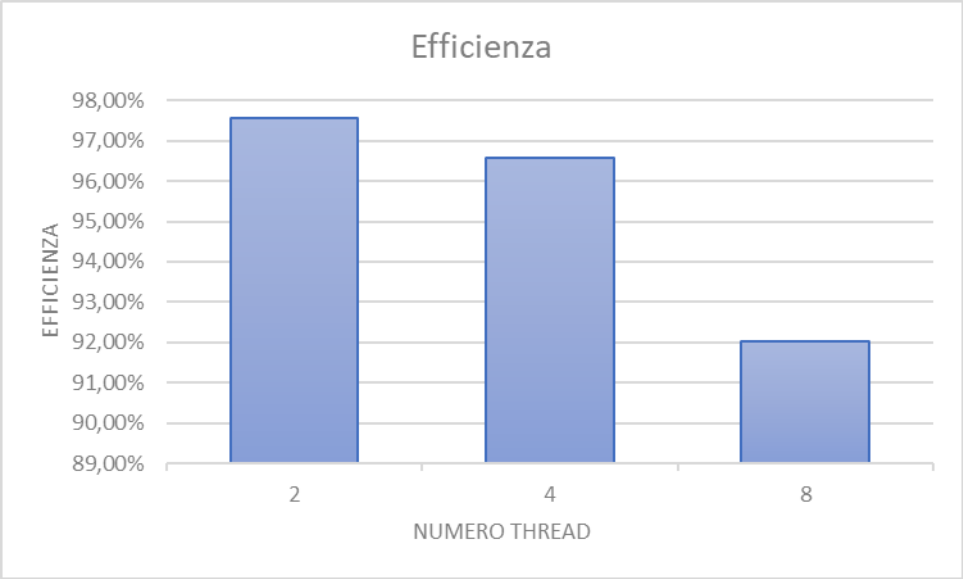


Raddoppiando il numero di thread, i tempi risultano quasi dimezzati. I valori di speed up sono prossimi a quelli ideali per tutte le soluzioni multi-thread e dunque l'efficienza risulta molto alta in tutte e tre le soluzioni:

# thread	Speed-up effettivo	Speed-up ideale
2	1,95	2
4	3,86	4
8	7,36	8



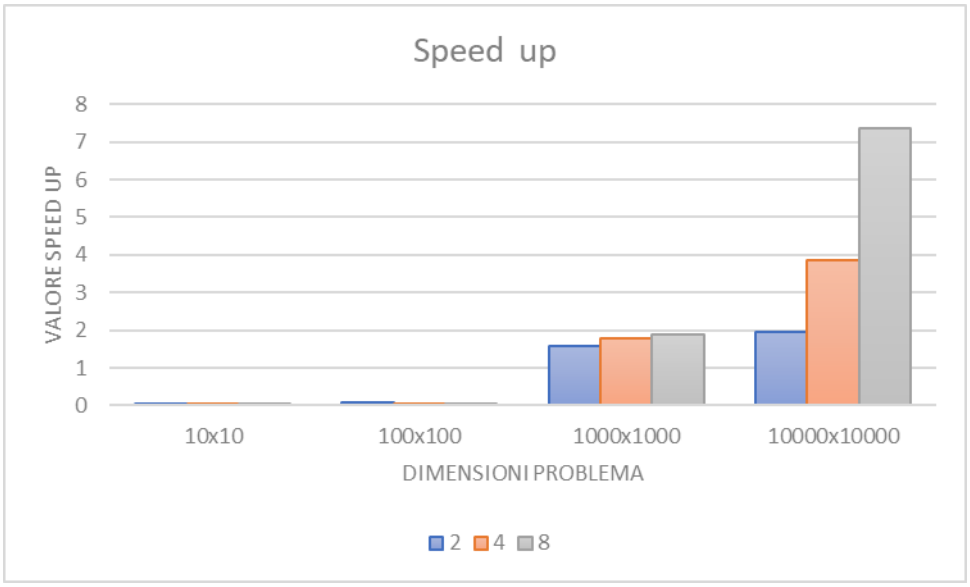
# thread	Efficienza
2	97,57%
4	96,59%
8	92,03%

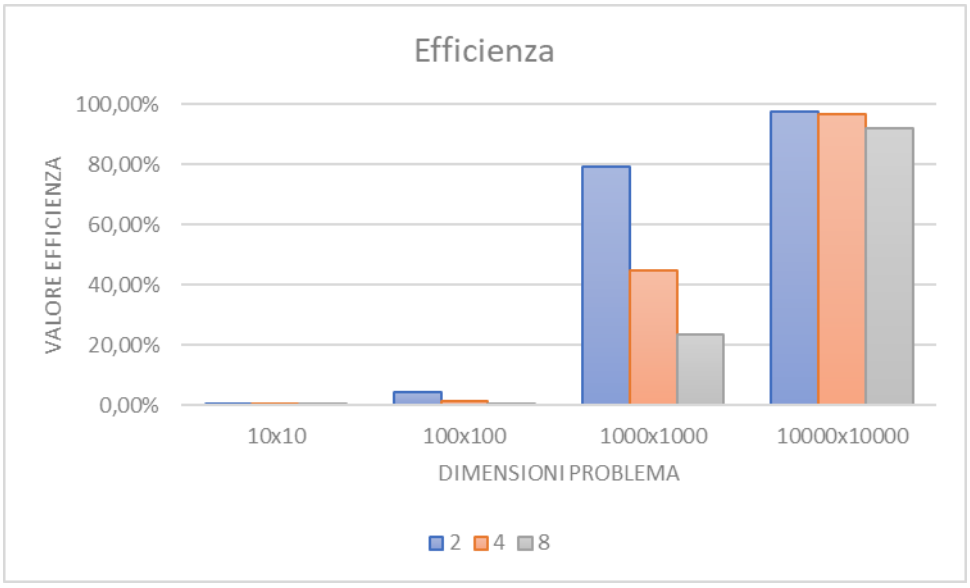


Conclusioni

Con una dimensione del problema molto ridotta ci rediamo conto che qualsiasi soluzione multi-thread non comporta nessuna miglioria ed anzi si rilevano anomalie nei tempi per i casi *10x10* e *100x100*. Al crescere della dimensione, oltre ad evidenziare una riduzione importante dei tempi di esecuzione rispetto al caso di un solo thread, osserviamo un incremento progressivo di speed-up ed efficienza, avvicinandoci alle condizioni ideali con la dimensione *10000x10000*.

Ai fini riassuntivi, vengono di seguito riportati i grafici di speed-up ed efficienza al variare della dimensione del problema.





Esempi d’uso

Sono stati riportati due esempi d’uso, uno relativo ad una matrice *20x20* le cui dimensioni sono state inserite correttamente ed uno relativo agli errori *E0* e *E1* (i file *.pbs* e *.out* sono rispettivamente presenti nelle cartelle *Esempi d’uso* e *Errori*). Di seguito vengono riportati sia i dati di input che i risultati ottenuti da terminale.

1. Input: *mat_rig = 20, mat_col = 20.*

   Output:

```
Eseguo
Generazione matrice 20x20...
1 3 7 2 8 6 3 9 6 5 1 4 8 8 6 6 6 3 2 8
3 9 5 9 6 4 1 7 6 5 6 6 7 3 7 4 8 7 1 5
9 1 8 5 7 2 2 3 5 3 8 7 9 1 4 3 2 2 1 7
5 6 1 9 6 5 3 4 2 3 6 8 2 4 4 8 4 3 8 8
5 4 3 3 5 4 3 6 5 3 2 9 6 2 7 2 5 7 4 4
8 9 3 7 2 4 3 5 6 1 3 1 4 5 1 6 6 4 1 1
4 9 8 8 8 5 9 3 9 1 7 7 1 9 4 9 3 6 4 6
4 4 6 6 6 5 9 2 8 9 9 9 6 7 7 5 9 7 7 9
5 2 4 5 8 6 4 8 2 5 4 6 8 8 9 2 3 8 3 8
8 3 7 2 7 5 6 7 9 2 4 4 3 5 7 2 1 8 9 1
4 4 4 9 2 3 2 2 1 4 9 6 4 4 8 2 6 2 6 3
3 7 5 6 2 2 5 1 9 4 1 1 5 4 1 4 6 2 3 4
3 9 8 5 4 6 4 9 7 7 1 8 4 5 4 3 6 6 1 3
7 1 4 1 2 2 4 7 3 5 9 3 4 7 5 7 1 8 5 5
5 5 3 6 9 4 9 3 9 7 3 7 8 6 7 9 7 8 5 7
3 4 8 7 8 3 2 8 9 4 3 4 6 4 8 3 7 7 5 7
2 8 2 9 2 6 7 7 4 2 2 5 3 9 9 1 1 1 6 9
5 8 1 1 9 8 2 7 3 6 2 5 2 3 2 2 6 8 8 9
7 7 4 9 7 3 7 5 2 3 2 4 9 9 2 8 8 3 3 8
7 4 1 6 4 3 7 9 8 3 7 6 1 1 3 5 2 1 9 1
```

```
Generazione vettore...
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
Calcolo Ax = b...
Inizio regione parallela...
Fine regione parallela.
Risultato:
102
109
89
99
89
80
120
134
108
100
85
75
103
90
127
110
95
97
110
88
Tempo di esecuzione con 8.000135e-03s threads: 8
```

2. Input: *mat_rig = -10, mat_col = -10.*

   Output:

```
Eseguo
[E0] Numero -10 di righe non valido. Valore default: 20.
[E1] Numero -10 di colonne non valido. Valore default: 20.
Generazione matrice 20x20...
6 1 8 7 2 4 8 5 1 3 6 6 9 1 3 3 1 5 1 2
6 3 8 9 4 3 3 3 5 2 2 1 2 9 5 2 1 3 4 1
5 9 5 2 7 7 5 8 2 5 9 6 5 5 3 9 5 6 9 9
7 8 9 7 6 5 8 6 5 9 5 1 8 9 2 6 6 4 2 5
7 1 1 2 4 4 8 8 7 5 6 4 4 3 8 9 5 6 3 1
5 7 8 2 6 9 5 9 2 4 5 8 2 5 9 5 6 6 2 3
1 5 5 2 7 3 8 3 7 2 1 9 8 8 8 3 5 1 9 6
4 4 4 6 7 2 8 3 5 7 4 5 2 8 5 9 1 3 9 5
4 9 5 1 7 3 1 2 4 9 6 5 4 9 1 8 8 9 1 3
6 2 6 6 9 1 5 8 1 4 3 5 1 7 3 7 8 3 6 9
9 2 4 1 9 5 8 7 2 7 8 5 8 4 1 6 2 3 4 2
4 6 4 4 2 6 1 7 6 7 6 6 6 9 4 5 2 3 3 1
9 8 6 5 2 4 1 1 7 2 2 1 6 4 3 7 9 3 2 4
7 7 7 4 4 1 8 4 1 8 4 7 6 7 3 5 9 1 5 4
3 5 2 8 8 4 3 5 7 2 6 2 8 3 3 9 2 2 3 2
7 5 9 4 2 9 6 8 7 9 2 9 2 4 5 9 7 5 2 2
6 8 2 4 8 4 2 9 5 4 9 3 6 6 4 8 3 7 6 9
6 6 9 7 9 2 4 6 7 6 6 3 2 7 5 9 1 6 7 4
7 4 4 4 9 5 9 9 2 3 8 6 8 5 3 7 7 5 2 4
8 5 6 9 2 8 6 9 4 1 1 2 4 4 3 1 8 2 9 8
Generazione vettore...
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
```

```
Calcolo Ax = b...
Inizio regione parallela...
Fine regione parallela.
Risultato:
82
76
121
118
96
108
101
101
99
100
97
92
86
102
87
113
113
112
111
100
Tempo di esecuzione con 6.029129e-03s threads: 8
```