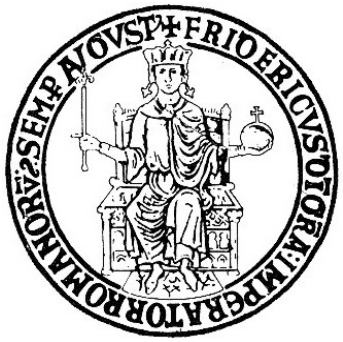
Università degli Studi di Napoli Federico II
Scuola Politecnica e delle Scienze di Base
Dipartimento di Ingegneria Elettrica e Tecnologie dell'Informazione



Corso di Laurea Magistrale in Informatica
Insegnamento di Parallel and Distributed Computing
Anno accademico 2020/2021

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

Autori:
Salvatore Bazzicalupo, N97000345
Annarita Della Rocca, N97000341

Docenti:
Prof. Giuliano Laccetti
Dott.sa Valeria Mele

Questa pagina è stata lasciata intenzionalmente bianca.

Indice

Capitolo 1

Definizione ed analisi del problema

Si vuole sviluppare un algoritmo per il calcolo del prodotto matrice-vettore, in ambiente di calcolo parallelo su architettura MIMD (Multiple Instruction stream Multiple Data) a memoria condivisa, che utilizzi la libreria OpenMp.

L'algoritmo prende in input da un script PBS il numero di righe e di colonne della matrice da generare, la dimensione del vettore sarà uguale al numero di colonne della matrice; i valori in essi contenuti sono generati in maniera casuale mediante un'opportuna funzione. Infine viene calcolato il prodotto matrice-vettore.

Capitolo 2

Descrizione algoritmo

L'algoritmo riceve alcuni parametri in input:

- il numero di colonne della matrice;
- il numero di righe della matrice.

2.1 Job-Script PBS

L'algoritmo utilizza uno script PBS per ricevere i parametri in input specificati nella sezione precedente, nello script è presente la seguente porzione di codice:

```
#########################
## CUSTOM VALUES ##
#########################
COLUMN=10000
ROWS=10000
THREADS=5
#########################
```

Dove queste variabili hanno il seguente significato:

- `COLUMN` - numero di colonne della matrice da generare.
- `ROWS` - Numero di righe della matrice da generare.
- `THREADS` - Numero di thread da utilizzare per l'esecuzione del programma.

2.2 Algoritmo in C

L'algoritmo prevede una fase di controlli di consistenza sui parametri passati in input. Nel dettaglio viene controllato che:

- l'argomento in input sia un numero reale;
- l'argomento in input sia positivo.

```
1  double* vectorXMatrix(int column, int row, const double* vector,
       double** matrix, double* timeElapsed) {
2      int i, j;
3      double* result = malloc(sizeof(double) * column);
4      struct timeval timeVal;
5      double startTime, endTime;
6
7      // Tempo iniziale
8      gettimeofday(&timeVal, NULL);
9      startTime = timeVal.tv_sec + (timeVal.tv_usec/1000000.0);
10
11     // Effettua il calcolo matrice x vettore utilizzando openmp
12     #pragma omp parallel for default(none) shared(column, row,
       vector, matrix, result, timeElapsed) private(i, j)
13     for (i = 0; i < row; i++) {
14         for (j = 0; j < column; j++) {
15             result[i] += matrix[i][j] * vector[j];
16         }
17     }
18
19     // Tempo finale
20     gettimeofday(&timeVal, NULL);
21     endTime = timeVal.tv_sec + (timeVal.tv_usec/1000000.0);
22
23     // Tempo totale ottenuto dalla differenza finale - iniziale
24     *timeElapsed = endTime - startTime;
25
26     return result;
27 }
```

Il metodo dichiara le variabili necessarie, salva l'istante di tempo d'inizio, inizia la sezione parallela dell'algoritmo che effettua il prodotto, salva l'istante di tempo finale, infine calcola il tempo impegato.

Capitolo 3

Input ed Output

L'algoritmo opera su numeri reali, richiede i seguenti valori da riga di comando:

- il numero N di colonne della matrice;
- il numero M di righe della matrice.

L'output dell'algoritmo, è della seguente forma:

```
<risultato>
Time elapsed: <ammontare-secondi>
```

I valori tra parentesi angolari corrispondono a:

- `<risultato>`: il risultato del prodotto tra la matrice e il vettore;
- `<ammontare-secondi>`: il tempo totale impiegato dai processori per calcolare il prodotto richiesto.

Capitolo 4

Indicatori di errore

Tutti i messaggi di errore dell'applicativo seguono la seguente forma:
"ERROR - <descrizione-errore>: <parametri>"

Con i valori tra parentesi angolari:

- `<descrizione-errore>`: Una sintetica descrizione dell'errore verificatosi;
- `<parametri>`: una lista della forma chiave:valore dei parametri che hanno causato l'errore.

In `main`:

```
1     // assegnazione di valori alle dimensioni
2     if (!(parseInt(argv[1], &column)) || !(parseInt(argv[2], &row)))
      {
3         fprintf(stderr, "ERROR - Cannot parse the number of column
    or rows with values provided.\n"
4                         "column:%s\nrow:%s\n\n", argv[1], argv[2]);
5         return 1;
6     }
7
8     if (column < 1 || row < 1) {
9         fprintf(stderr, "ERROR - Column and row number less than 1
    are not allowed.\ncolumn:%d\nrow:%d\n", column, row);
10        return 1;
11    }
```

Nella porzione di codice sopra elencata, si controlla che il numero di colonne e di righe passato in input sia valido.

Capitolo 5

Subroutine

Nell'algoritmo sono presenti varie subroutine utilizzate, queste sono documentate a livello interno del codice tramite commenti che hanno la seguente forma:

- Breve descrizione del metodo;
- Lista dei parametri con descrizione dettagliata del loro utilizzo;
- Se presente un output, a seconda delle diverse condizioni possibili, viene descritta la sua forma.

Sono quindi riportate le firme dei metodi, la loro documentazione interna ed eventualmente una descrizione aggiuntiva.

5.1 Subroutines personalizzate

Di seguito, verranno descritte alcune subroutine personalizzate utilizzate nel progetto.

Le prime subroutine sono dei brevi metodi che permettono di evitare ripetizioni nel programma favorendo il riuso del codice e di fare error checking.

```
1 /**
2  * Converte una stringa in input in int
3  * @param str la stringa da convertire
4  * @param val dove viene salvato il risultato della conversione
5  * @return true se la conversione termina con successo, falso
     altrimenti
6  */
7 bool parseInt(char* str, int* val);
```

```
8
9  /**
10  * Ritorna un numero casuale nel range definito
11  * @param min il numero minimo, incluso
12  * @param max il numero massimo, incluso
13  * @return float il numero casuale
14  */
15 double getRandomDoubleNumberInRange(int min, int max);
16
17 /**
18  * Alloca una matrice di dimensione column*row con valori casuali
19  * @param column il numero di colonne
20  * @param row il numero di righe
21  * @return la matrice allocata e riempita casualmente
22  */
23 double** getMatrixOfRandomNumbersOfSize(int column, int row);
24
25 /**
26  * Alloca un vettore di dimensione size con valori casuali
27  * @param size la dimensione del vettore
28  * @return il vettore allocato e riempito casualmente
29  */
30 double* getVectorOfRandomNumbersOfSize(int size);
```

La seguente subroutine è utilizzata per il calcolo del prodotto matrice-vettore:

```
1  /**
2   * Effettua il calcolo matrice per vettore e ritorna il risultato
3   * @param column il numero di colonne della matrice
4   * @param row il numero di righe della matrice
5   * @param vector il vettore di dimensione column
6   * @param matrix la matrice di dimensione column*row
7   * @param timeElapsed valore di ritorno che rappresenta il tempo
      impiegato
8   * @return il vettore risultato di dimensione column
9   */
10 double* vectorXMatrix(int column, int row, const double* vector,
      double** matrix, double* timeElapsed);
```

Questa subroutine, utilizza la funzione **gettimeofday** per poter calcolare il tempo impiegato per performare il prodotto.

5.2 Direttive OpenMp

Seguono le direttive OpenMp utilizzate nel progetto.
Esse sono documentate differentemente da quelle personalizzate in quanto la documentazione ufficiale è disponibile su openmp.org, viene però fornita una breve descrizione.

```
1 #pragma omp parallel for default(none) shared() private()
```

Descrizione: costrutto OpenMp a cui vengono applicate delle direttive.

Direttive:

- **parallel**: forma un team di thread ed avvia così un'esecuzione parallela;
- **for**: è uno dei tre costrutti chiamati *WorkSharing*. Si occupano della distribuzione del lavoro al team di thread, gli altri due sono *sections, single*. Il costrutto for specifica che le iterazioni del ciclo contenuto devono essere distribuite tra i thread del team.

Clausole:

- **default(shared|none)**: controlla gli attributi di data-sharing delle variabili in un costrutto. In particolare, *shared* fa si che tutte le variabili saranno considerate condivise, mentre se si utilizza *none*, sarà il programmatore a decidere;
- **shared**: vengono specificate le variabili condivise tra tutti i thread;
- **private**: vengono specificate le variabili non condivise tra i thread.

Capitolo 6

Analisi dei tempi

Sono state effettuate diverse analisi dei tempi di esecuzione dell'algoritmo in base al numero di thread utilizzati e alla dimensioni della matrice e del vettore.

I tempi che seguono sono una media di 10 esecuzioni per ogni dimensione della matrice presa in considerazione in relazione al numero di thread utilizzati (da 1 ad 8).

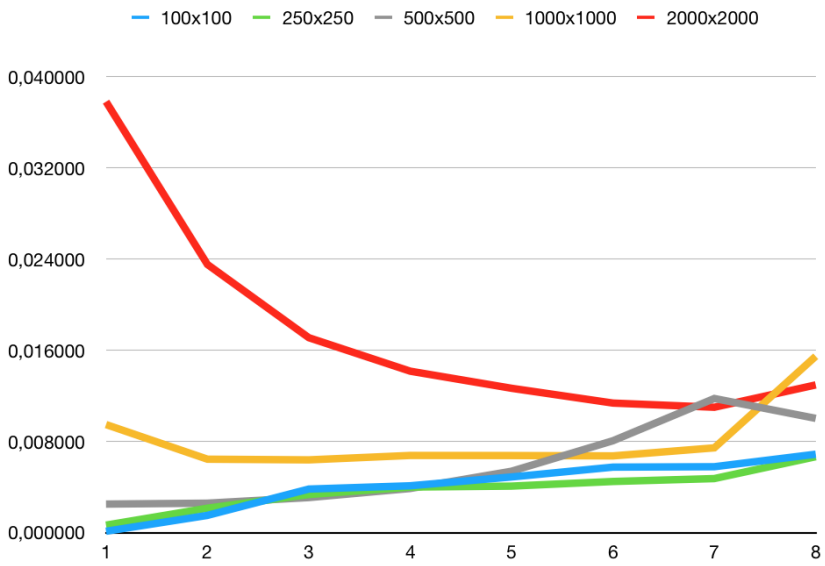
È stato testato l'algoritmo utilizzando delle matrici di dimensioni 100x100, 250x250, 500x500, 1000x1000, 2000x2000, 5000x5000, 10000x10000.

Seguono i valori di riferimento dei tempi ottenuti con un numero di thread diverso:

Tempi per processore

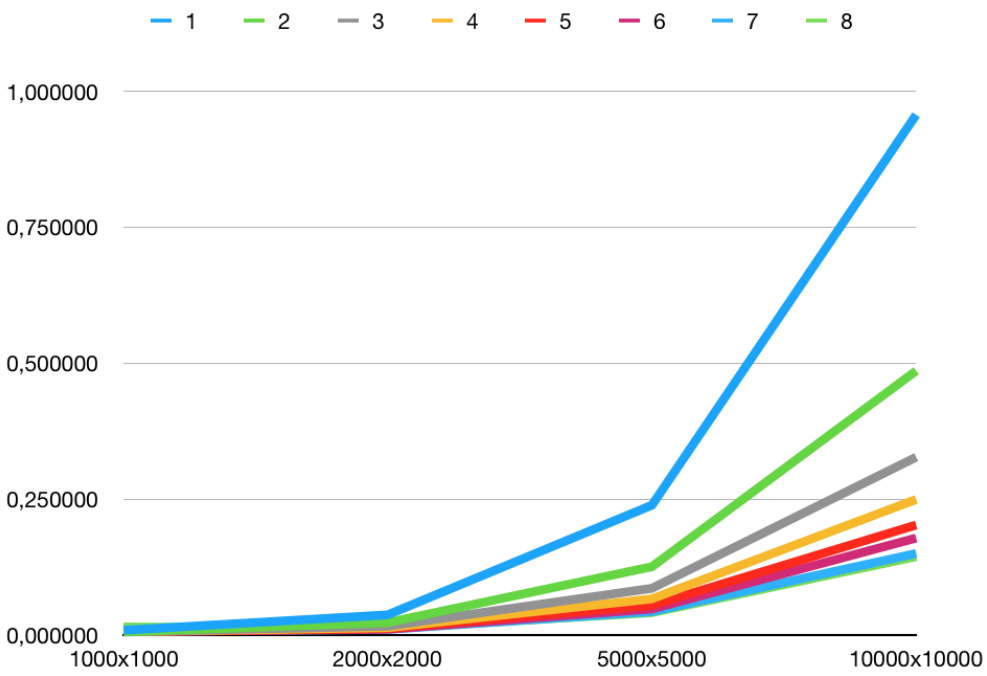
	100x100	250x250	500x500	1000x1000	2000x2000	5000x5000	10000x10000
1	0,000119	0,000657	0,002505	0,009486	0,037827	0,239457	0,956990
2	0,001513	0,002144	0,002585	0,006446	0,023540	0,126277	0,486406
3	0,003823	0,003344	0,003077	0,006386	0,017111	0,086648	0,327718
4	0,004111	0,003997	0,003874	0,006774	0,014172	0,067240	0,250006
5	0,004885	0,004083	0,005389	0,006766	0,012670	0,055298	0,202964
6	0,005749	0,004488	0,008064	0,006734	0,011372	0,048636	0,178980
7	0,005784	0,004744	0,011772	0,007442	0,011004	0,043715	0,150713
8	0,006881	0,006663	0,010030	0,015477	0,012964	0,041905	0,144755

Segue un grafico con il tempo impiegato per ogni dimensione di matrice per calcolare il prodotto, per chiarezza è rappresentato fino a 2000x2000.



Dal grafico si evince che il tempo impiegato per calcolare il prodotto di una matrice 2000x2000 diminuisca man mano che i thread aumentano, viceversa con matrici più piccole all'aumentare dei thread aumenta anche il tempo impiegato in quanto l'overhead di comunicazione tra i thread non è trascurabile.

Segue un grafico con tutti gli 8 thread e le matrici che mostra il tempo impiegato per il calcolo del prodotto.



Dal grafico si evince banalmente che con un minor numero di thread il tempo impiegato aumenta all'aumentare della dimensione della matrice.

6.1 Speed-up ed efficienza

Conseguentemente alle precedenti analisi è possibile calcolare lo speed-up e l'efficienza.

6.1.1 Calcolo dello speed-up

Lo speed-up misura la riduzione del tempo di esecuzione rispetto all'algoritmo mediante un solo thread.

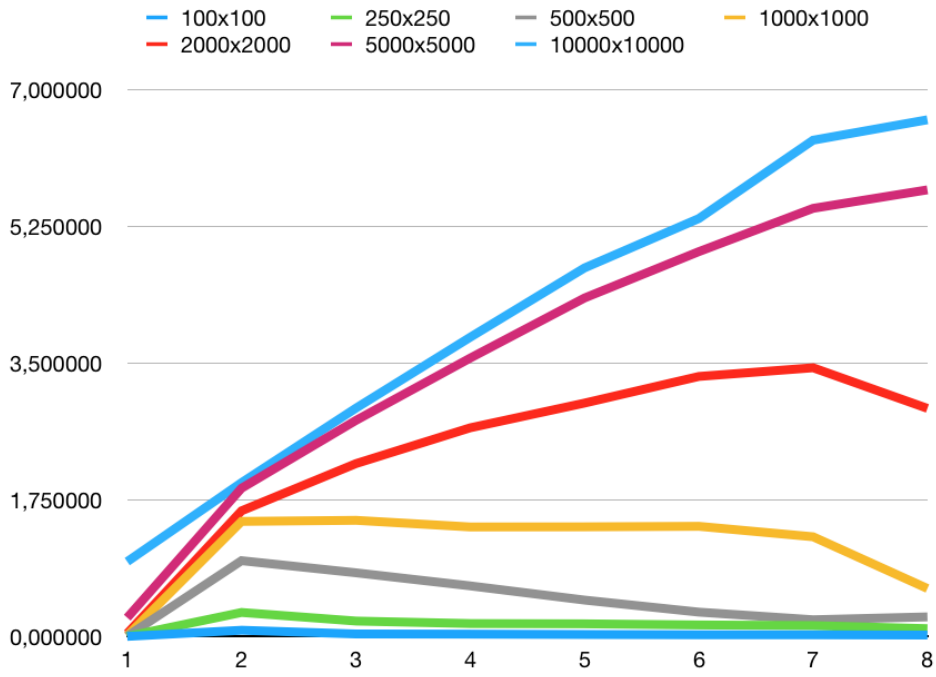
Definiamo lo speed-up come: $S(n) = \frac{T(1)}{T(n)}$, e cioè il rapporto tra il tempo impiegato dall'algoritmo per calcolare il risultato utilizzando un solo thread e il tempo impiegato per calcolare il risultato utilizzando n thread.

Seguono le tabelle contenenti i valori relativi al calcolo dello speed-up:

Speed-up

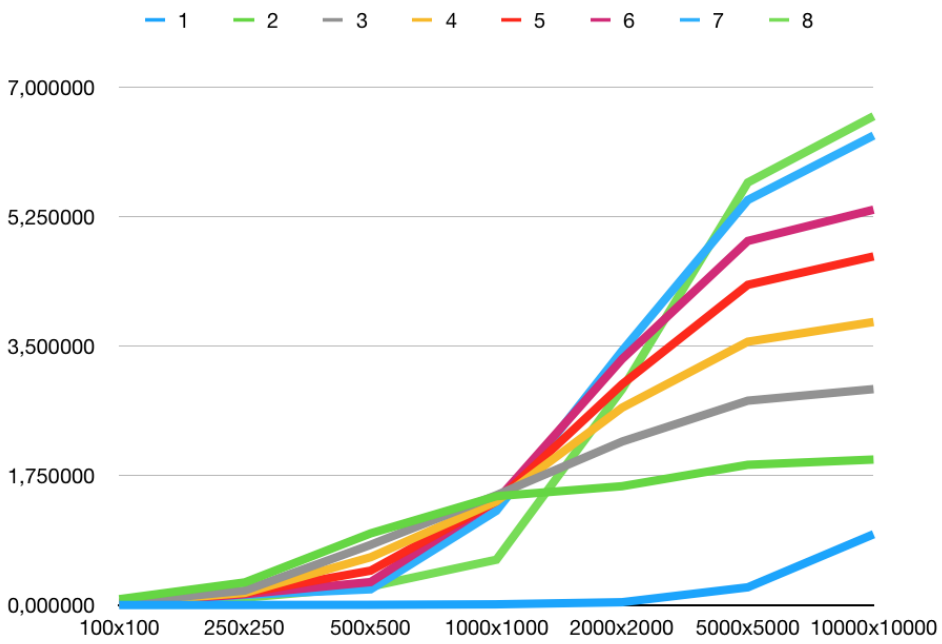
	100x100	250x250	500x500	1000x1000	2000x2000	5000x5000	10000x10000
1	0,000119	0,000657	0,002505	0,009486	0,037827	0,239457	0,956990
2	0,078652	0,306437	0,969052	1,471610	1,606924	1,896284	1,967472
3	0,031127	0,196471	0,814105	1,485437	2,210683	2,763561	2,920163
4	0,028947	0,164373	0,646618	1,400354	2,669136	3,561228	3,827868
5	0,024360	0,160911	0,464836	1,402010	2,985556	4,330301	4,715073
6	0,020699	0,146390	0,310640	1,408672	3,326328	4,923452	5,346910
7	0,020574	0,138491	0,212793	1,274657	3,437568	5,477685	6,349751
8	0,017294	0,098604	0,249751	0,612909	2,917849	5,714282	6,611102

Segue il grafico della tabella appena mostrata.



Si può notare come lo speed-up aumenta all'aumentare dei thread, mentre in alcuni casi come con una matrice 2000x2000 può diminuire con molti thread.

Segue il grafico relativo al numero di thread con le differenti dimensioni di matrici.


1
2
3
4
5
6
7
8
7,000000
5,250000
3,500000
1,750000
0,000000
100x100
250x250
500x500
1000x1000
2000x2000
5000x5000
10000x10000

6.1.2 Calcolo dell'efficienza

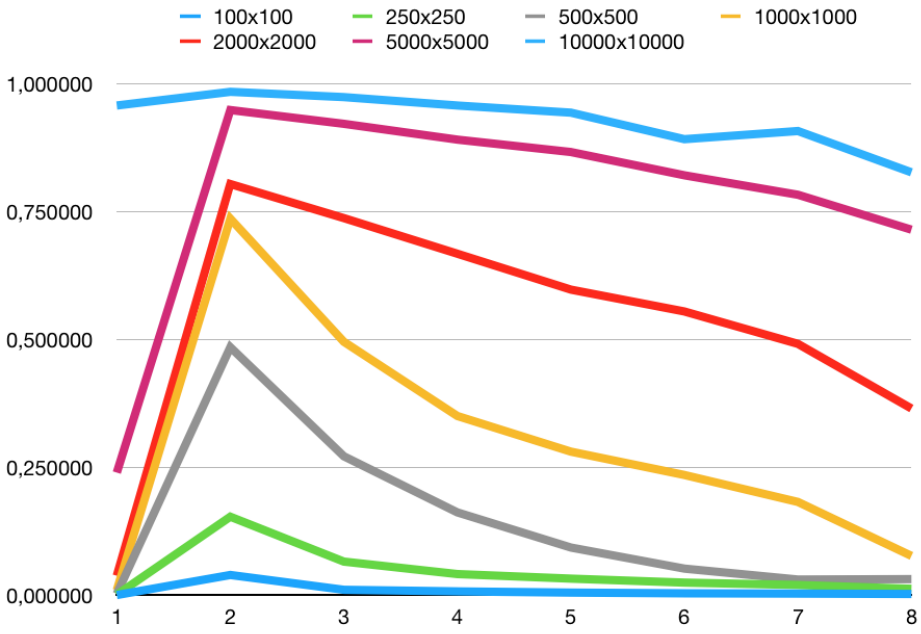
L'efficienza misura quanto l'algoritmo sfrutta il parallelismo del calcolatore. Definiamo l'efficienza come: $E(n) = \frac{S(n)}{n}$, e cioè il rapporto tra lo speed-up su n thread e il numero di thread.

Segue la tabella contenente i valori relativi al calcolo dell'efficienza.

Efficienza

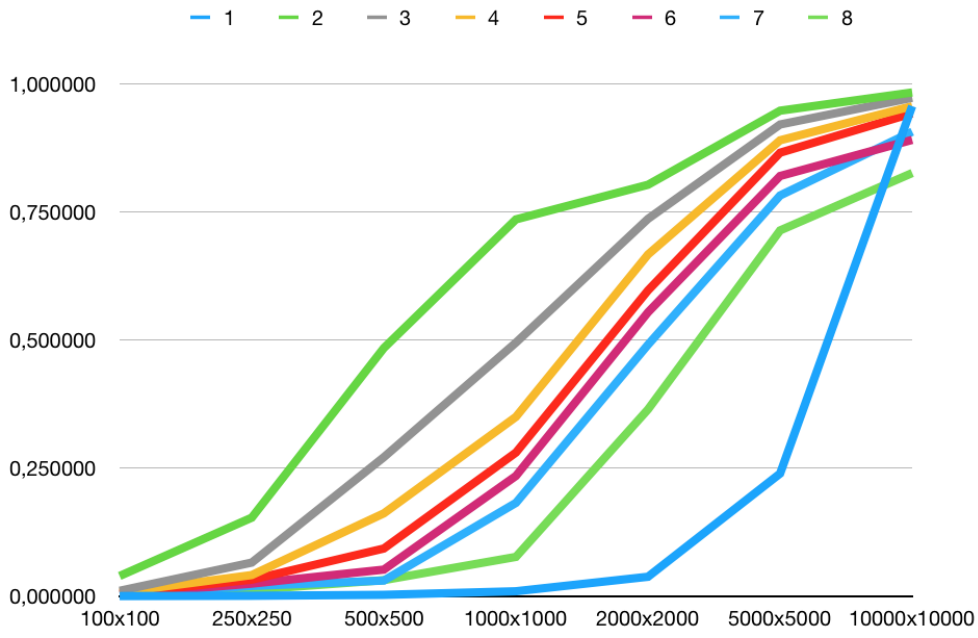
	100x100	250x250	500x500	1000x1000	2000x2000	5000x5000	10000x10000
1	0,000119	0,000657	0,002505	0,009486	0,037827	0,239457	0,956990
2	0,039326	0,153218	0,484526	0,735805	0,803462	0,948142	0,983736
3	0,010376	0,065490	0,271368	0,495146	0,736894	0,921187	0,973388
4	0,007237	0,041093	0,161655	0,350089	0,667284	0,890307	0,956967
5	0,004872	0,032182	0,092967	0,280402	0,597111	0,866060	0,943015
6	0,003450	0,024398	0,051773	0,234779	0,554388	0,820575	0,891152
7	0,002939	0,019784	0,030399	0,182094	0,491081	0,782526	0,907107
8	0,002162	0,012326	0,031219	0,076614	0,364731	0,714285	0,826388

Segue il grafico per le varie dimensioni di matrice:



Si può notare come la matrice 10000x10000 impieghi un tempo che si riduce all'aumentare dei processori, così come un comportamento interessante si ha quando il numero di thread è 2.

Segue il grafico relativo ai vari thread, che risultano più performanti all'aumentare della dimensione della matrice.



Capitolo 7

Esempi d'uso

Si può utilizzare l'algoritmo in due modi diversi: utilizzandolo in maniera diretta, quindi invocando in locale la libreria openMP; oppure utilizzando uno script PBS. Seguono esempi d'uso per entrambi i casi.

7.1 Uso diretto

Ipotizzando di avere nella stessa cartella il file main.c, contenente il codice sorgente, è necessario compilarlo eseguendo:

```
gcc -o main main.c -fopenmp -lgomp
```

Ipotizzando di voler utilizzare 4 thread, ed effettuare il prodotto di una matrice 10x10, sarà necessario settare il numero di thread e poi lanciare il programma eseguendo le seguenti istruzioni bash:

```
export OMP_NUM_THREADS=4
./main 10 10
```

Verrà stampato il seguente risultato:

```
19880.809975
23813.573187
19734.741161
19162.550869
21867.428056
16037.001328
22233.254453
21058.119391
```

```
18800.544583
19570.793370

Time elapsed: 1.296997e-03
```

Si noti che il risultato può variare in quanto i numeri sono generati casualmente, così come il tempo di esecuzione può variare in base all'hardware del dispositivo.

7.2 Uso tramite script PBS

Utilizzando l'algoritmo su un cluster è necessario utilizzare uno script PBS, in appendice ne è fornito uno che verrà utilizzato in questo caso d'uso ed è chiamato job-script.pbs.

Si ipotizzi di voler ottenere il prodotto di una matrice 10x10 e di voler utilizzare 8 thread.

Innanzitutto sarà necessario modificare la seguente parte dello script PBS:

```
1  ...
2  #PBS -l nodes=8:ppn=8
3  #PBS -o es2.out
4  ...
5  ######################
6  ## CUSTOM VALUES ##
7  ######################
8  COLUMN=10
9  ROWS=10
10 THREADS=8
11 ######################
```

Ed invocare lo script come segue:

```
qsub job-script.pbs
```

Attendere il completamento e, supponendo che il risultato sia memorizzato nel file es2.out (come scritto nel file pbs) digitare:

```
cat es2.out
```

Verrà stampato il seguente risultato:

```
19880.809975
23813.573187
19734.741161
19162.550869
21867.428056
16037.001328
22233.254453
21058.119391
18800.544583
19570.793370

Time elapsed: 1.296997e-03
```

Si noti che il risultato può variare in quanto i numeri sono generati casualmente, così come il tempo di esecuzione può variare in base all'hardware del dispositivo.

Bibliografia

[1] OpenMP Documentation,
`https://www.openmp.org`

Capitolo A

Codice

Segue il codice dell'algoritmo.

A.1 main.c

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <errno.h>
#include <limits.h>
#include <time.h>
#include <sys/time.h>

#define MIN_RANDOM 0
#define MAX_RANDOM 100

double* vectorXMatrix(int column, int row, const double* vector,
    double** matrix, double* timeElapsed);

// Funzioni accessorie
double getRandomDoubleNumberInRange(int min, int max);
double** getMatrixOfRandomNumbersOfSize(int column, int row);
double* getVectorOfRandomNumbersOfSize(int size);
bool parseInt(char* str, int* val);

/**
```

```
21  * Gli argomenti vengono passati nella forma: column row
      numberOfThreads
22  * @param column numero di colonne della matrice
23  * @param row numero di righe della matrice
24  */
25 int main(int argc, char** argv) {
26     srand(time(NULL));
27 
28     // Dichiarazione delle variabili
29     int i, column, row;
30     double* result = NULL;
31     double timeElapsed = 0;
32 
33     // assegnazione di valori alle dimensioni
34     if (!(parseInt(argv[1], &column)) || !(parseInt(argv[2], &row)))
       {
35         fprintf(stderr, "ERROR - Cannot parse the number of column
       or rows with values provided.\n"
36                         "column:%s\nrow:%s\n\n", argv[1], argv[2]);
37         return 1;
38     }
39 
40     if (column < 1 || row < 1) {
41         fprintf(stderr, "ERROR - Column and row number less than 1
       are not allowed.\ncolumn:%d\nrow:%d\n", column, row);
42         return 1;
43     }
44 
45     // assegnazione dei valori alla matrice A ed al vettore vector
46     double** matrix = getMatrixOfRandomNumbersOfSize(column, row);
47     double* vector = getVectorOfRandomNumbersOfSize(column);
48 
49     // Ottiene e stampa il risultato
50     result = vectorXMatrix(column, row, vector, matrix, &timeElapsed
       );
51     for (i = 0; i < row; ++i) {
52         printf("%f\n", result[i]);
53     }
54     printf("\nTime elapsed: %e\n\n", timeElapsed);
```

```
55
56      free(result);
57      free(vector);
58      for (i = 0; i < column; ++i) {
59          free(matrix[i]);
60      }
61      free(matrix);
62
63      return 0;
64  }
65
66  /**
67   * Effettua il calcolo matrice per vettore e ritorna il risultato
68   * @param column il numero di colonne della matrice
69   * @param row il numero di righe della matrice
70   * @param vector il vettore di dimensione column
71   * @param matrix la matrice di dimensione column*row
72   * @param timeElapsed valore di ritorno che rappresenta il tempo
       impiegato
73   * @return il vettore risultato di dimensione column
74   */
75  double* vectorXMatrix(int column, int row, const double* vector,
       double** matrix, double* timeElapsed) {
76      int i, j;
77      double* result = malloc(sizeof(double) * column);
78      struct timeval timeVal;
79      double startTime, endTime;
80
81      // Tempo iniziale
82      gettimeofday(&timeVal, NULL);
83      startTime = timeVal.tv_sec + (timeVal.tv_usec/1000000.0);
84
85      // Effettua il calcolo matrice x vettore utilizzando openmp
86      #pragma omp parallel for default(none) shared(column, row,
       vector, matrix, result, timeElapsed) private(i, j)
87      for (i = 0; i < row; i++) {
88          for (j = 0; j < column; j++) {
89              result[i] += matrix[i][j] * vector[j];
90          }
```

```
91      }
92
93      // Tempo finale
94      gettimeofday(&timeVal, NULL);
95      endTime = timeVal.tv_sec + (timeVal.tv_usec/1000000.0);
96
97      // Tempo totale ottenuto dalla differenza finale - iniziale
98      *timeElapsed = endTime - startTime;
99
100     return result;
101 }
102
103 /**
104  * Converte una stringa in input in int
105  * @param str la stringa da convertire
106  * @param val dove viene salvato il risultato della conversione
107  * @return true se la conversione termina con successo, falso
        altrimenti
108  */
109 bool parseInt(char* str, int* val) {
110     char *temp;
111     bool result = true;
112     errno = 0;
113     long ret = strtol(str, &temp, 0);
114
115     if (temp == str || *temp != '\0' || ((ret == LONG_MIN || ret ==
        LONG_MAX) && errno == ERANGE)) {
116         result = false;
117     }
118
119     *val = (int) ret;
120     return result;
121 }
122
123 /**
124  * Ritorna un numero casuale nel range definito
125  * @param min il numero minimo, incluso
126  * @param max il numero massimo, incluso
127  * @return float il numero casuale
```

```
128  */
129 double getRandomDoubleNumberInRange(int min, int max) {
130     return (double) min + rand() / (double) RAND_MAX * max - min;
131 }
132
133 /**
134  * Alloca una matrice di dimensione column*row con valori casuali
135  * @param column il numero di colonne
136  * @param row il numero di righe
137  * @return la matrice allocata e riempita casualmente
138  */
139 double** getMatrixOfRandomNumbersOfSize(int column, int row) {
140     int i, j;
141     double** matrix = malloc(sizeof(double*) * column);
142     for (i = 0; i < column; ++i) {
143         matrix[i] = malloc(sizeof(double*) * row);
144         for (j = 0; j < row; ++j) {
145             matrix[i][j] = getRandomDoubleNumberInRange(MIN_RANDOM,
    MAX_RANDOM);
146         }
147     }
148     return matrix;
149 }
150
151 /**
152  * Alloca un vettore di dimensione size con valori casuali
153  * @param size la dimensione del vettore
154  * @return il vettore allocato e riempito casualmente
155  */
156 double* getVectorOfRandomNumbersOfSize(int size) {
157     int i;
158     double* vector = malloc(sizeof(double) * size);
159     for (i = 0; i < size; ++i) {
160         vector[i] = getRandomDoubleNumberInRange(MIN_RANDOM,
    MAX_RANDOM);
161     }
162     return vector;
163 }
```

A.2 job-script.pbs

```
#!/bin/bash

# Setting env variables, NO LINES OF CODE MUST BE BEFORE THOSE
    DIRECTIVES
#PBS -q studenti
#PBS -l nodes=8:ppn=8
#PBS -N es2
#PBS -o es2.out
#PBS -e es2.err

sort -u $PBS_NODEFILE > hostlist

echo "[Job-Script] Compiling..."
PBS_O_WORKDIR=$PBS_O_HOME/20-december
gcc -fopenmp -lgomp -o $PBS_O_WORKDIR/es2 $PBS_O_WORKDIR/es2.c

########################
## CUSTOM VALUES ##
########################
COLUMN=10000
ROWS=10000
THREADS=5
########################

echo "[Job-Script] Starting with "$THREADS" Threads..."

export OMP_NUM_THREADS=$THREADS
export OMP_SCHEDULE="dynamic,2"
echo -e "\n=== Threads "$THREADS" - Matrix "$COLUMN"x"$ROWS" ===\n"
$PBS_O_WORKDIR/es2 $COLUMN $ROWS
```