Università degli Studi di Napoli
"Parthenope"

Corso di Calcolo Parallelo e Distribuito

Progetto Matrice per Matrice

Strategia BMR

Giovanni Di Cecca
Matr. 108/1569

© 2009 – Giovanni Di Cecca – http://www.dicecca.net

Sorgente disponibile sotto licenza GNU/GPL License

Indice

Definizione ed analisi del Problema……………………………………………….....5

Descrizione dell'algoritmo…………………………………………………….….…7
Introduzione……………………………………………………………….…...7
Inizializzazione dell'ambiente di calcolo………………………………...…...8
Inserimento dei dati…………………………………...……………….....8
Distribuzione dei dati…………………………………...…………….11
Calcolo del prodotto matriciale parziale e totale………………...……….....11

Compilazione ed esecuzione………………………………………………………. 15
Compilazione sotto Linux…………………………………………………... 15
Esecuzione del programma sotto Linux………………………………….. 15
Compilazione sotto Microsoft Windows XP………………..…………... 16
Esecuzione del programma sotto Microsoft Windows XP...……………….. 16

Indicatori di errore……………………………………………………………….17

Funzioni utilzzate……………………………………………………………. 19

Esempi d'uso………...…………………………………………………………25

Analisi dei tempi…………………………………………………………. 27
Introduzione ………………...……………………………………………... 27
Tabella dei tempi di esecuzione...………………………………………... 29
Tabella di Speed – Up ………...…………………………………………... 33
Tabella dell'Efficienza………...…………………………………………... 37
Commenti ai tempi…………...…………………………………………... 41

Bibliografia…...……………………………………………………………….. 43

Codice sorgente..……………………………………………………………….. 45

Definizione ed analisi del problema

Scopo: il software che si analizzerà di seguito ha lo scopo di effettuare il prodotto Matrice per Matrice usando un’architettura di tipo MIMD distribuendo il calcolo a n^2 processi disposto secondo una griglia a topologia bidimensionale, con le seguenti caratteristiche:

1. il numero di processi concorrenti è del tipo n^2;
2. l’ordine delle due matrici è proporzionale al numero di processi

$$\underbrace{\begin{bmatrix} C_{00} & C_{01} & C_{02} \\ C_{10} & C_{11} & C_{12} \\ C_{20} & C_{21} & C_{22} \end{bmatrix}}_{C} = \underbrace{\begin{bmatrix} A_{00} & A_{01} & A_{02} \\ A_{10} & A_{11} & A_{12} \\ A_{20} & A_{21} & A_{22} \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} B_{00} & B_{01} & B_{02} \\ B_{10} & B_{11} & B_{12} \\ B_{20} & B_{21} & B_{22} \end{bmatrix}}_{B}$$

$$\begin{aligned} C_{00} &= A_{00}B_{00} + A_{01}B_{10} + A_{02}B_{20} \\ C_{01} &= A_{00}B_{01} + A_{01}B_{11} + A_{02}B_{21} \\ C_{02} &= A_{00}B_{02} + A_{01}B_{12} + A_{02}B_{22} \end{aligned}$$

$$\begin{aligned} C_{10} &= A_{10}B_{00} + A_{11}B_{10} + A_{12}B_{20} \\ C_{11} &= A_{10}B_{01} + A_{11}B_{11} + A_{12}B_{21} \\ C_{12} &= A_{10}B_{02} + A_{11}B_{12} + A_{12}B_{22} \end{aligned}$$

$$\begin{aligned} C_{20} &= A_{20}B_{00} + A_{21}B_{10} + A_{22}B_{20} \\ C_{21} &= A_{20}B_{01} + A_{21}B_{11} + A_{22}B_{21} \\ C_{22} &= A_{20}B_{02} + A_{21}B_{12} + A_{22}B_{22} \end{aligned}$$

Descrizione dell'Algoritmo

Introduzione

La strategia usata per risolvere il calcolo del prodotto Matrice Matrice è quella della **B**roadcast **M**ultiply **R**olling (**BMR**) tecnica che prevede la decomposizione delle matrici di input in blocchi quadrati, e ciascuno di tali blocchi verrà poi assegnato ai processori disposti lungo una griglia bidimensionale periodica capace di distribuire una matrice $A \in \Re^{n \times m}$ e $B \in \Re^{m \times k}$ scorporandola in $p \times p$ processi su di una topologia di griglia bidimensionale.

Vediamo ora nel dettaglio le varie parti in gioco dell'algoritmo.

Descrizione dell'Algoritmo

L'algoritmo può essere suddiviso in cinque parti principali:

- Inizializzazione dell'ambiente di calcolo
- Inserimento dei dati
- Distribuzione dei dati
- Calcolo del prodotto mat mat parziale e totale
- Calcolo dei tempi

Per poter meglio analizzare le performances dell'algoritmo al suo interno è stato inserito il sistema di controllo del tempo.

Analizzeremo nel dettaglio le parti indicate

- **Inizializzazione dell’ambiente di calcolo**

```
//*****************************************************
// Inizio fase di inizializzazione delle variabili
//*****************************************************

int menum, nproc;
int m, n, k; // numero di righe e colonne delle matrici
int p, q, flag=0, sinc=1, offsetR=0, offsetC=0, dim_prod_par;
int N, i, j, z, y, tag, mittente, mittenter, destinatarior;
float *A, *B, *C, *subA, *subB, *subC, *Temp, *printTemp;


// Parametri per valutare le prestazioni
double t_inizio, t_fine, T1, Tp=0.F, speedup, Ep;

// Variabili di MPI
MPI_Status info;
MPI_Request rqst;
MPI_Comm griglia, grigliar, grigliac;
int coordinate[2];
int mittBcast[2];
int destRoll[2];
int mittRoll[2];

//Inizializzazione dell'ambiente MPI
MPI_Init(&argc, &argv);

//Calcolo dell'identificativo in MPI_COMM_WORLD
MPI_Comm_rank(MPI_COMM_WORLD, &menum);

//Calcolo del numero di processori
MPI_Comm_size(MPI_COMM_WORLD, &nproc);
```

- **Inserimento dei dati**

La prima verifica viene effettuata sul numero di processori utilizzati. Tale numero deve essere un quadrato perfetto (condizione di applicabilità della BMR), tale da poter generare una griglia quadrata. Se ciò non si verifica il programma termina

```
if(sqrt(nproc)*sqrt(nproc)!=nproc)
{
      if(menum==0)
      {
            printf("ERRORE:Impossibile applicare la strategia...\n");
            printf("Il numero di processori deve essere tale da generare una griglia quadrata.\n\n Ad esempio 1,4,9,16,25,...\n");
      }
      MPI_Finalize();
      return 0;
}
```

Un secondo controllo viene poi effettuato sulle dimensioni delle matrici inserite. Tali dimensioni devono essere divisibili per la radice quadrata del numero di processori, altrimenti viene generato un errore.

```
      // Procedura di inserimento dati nella Matrice A
      while(flag==0)
      {
            printf("\nInserire il numero di righe della matrice A:");
            fflush(stdin);
            scanf("%d", &m);

      //Calcola il valore di p. Esso verrà utilizzato per verificare se le dimensioni delle matrici sono divisibili...
      //...per il numero di processori
      p=sqrt(nproc);

      //Si richiede che il numero di righe di A sia multiplo di p
      if(m%p!=0)
            printf("ATTENZIONE:Numero di righe non divisibile per p=%d!\n", p);
            else
                  flag=1;
}

while(flag==1)
{
      //Si richiede che il numero di colonne di A sia multiplo di p
      printf("\nInserire il numero di colonne della matrice A:");
      fflush(stdin);
```

```
	scanf("%d", &n);

	//Si richiede che il numero di colonne di A sia multiplo di p
	if(n%p!=0)
		printf("ATTENZIONE:Numero di colonne non divisibile per p=%d!\n", 
p);
	else
		flag=0;
}

// Procedura di inserimento dati nella Matrice A
while(flag==0)
{
	printf("\nInserire il numero di colonne della matrice B:");
	fflush(stdin);
	scanf("%d", &k);

	//Si richiede che il numero di colonne di B sia multiplo di p
	if(k%p!=0)
		printf("ATTENZIONE:Numero di colonne non divisibile per p=%d!\n", 
p);
	else
		flag=1;
}
```

Dopo aver effettuato il controllo sulle matrici, viene allocato dinamicamente lo spazio, e si provvede all'inserimento dei dati. Tale inserimento dei dati può essere random oppure manuale (testo in blu).

```
//Allocazione dinamica delle matrici A, B e C solo in P0
//Il numero di righe di B è pari al numero di colonne di A
A=(float *)malloc(m*n*sizeof(float));
B=(float *)malloc(n*k*sizeof(float));
C=(float *)calloc(m*k, sizeof(float));

printf("\n\n\n*****************************************************\n");
printf("- Acquisizione elementi di A matrice %dx%d -\n", m, n );
printf("\n*****************************************************\n");
for(i=0;i<m;i++)
{
	for(j=0;j<n;j++)
```

```
        {
                // Comandi per l'inserimento dei dati a mano
                //printf("Digita elemento A[%d][%d]: ", i, j);
                //scanf("%f", A+i*n+j);

                //per valutare la prestazioni si attiva la riga seguente
                *(A+i*n+j)=(float)rand()/((float)RAND_MAX+(float)1);
        }
}

printf("\n\n\n*****************************************************\n");
printf("- Acquisizione elementi di B matrice %dx%d -\n",n, k );
printf("\n*****************************************************\n");
for(i=0;i<n;i++)
{
        for(j=0;j<k;j++)
        {
                // Comandi per l'inserimento dei dati a mano
                //printf("Digita elemento B[%d][%d]: ", i, j);
                //scanf("%f", B+i*k+j);

                //per valutare la prestazioni si attiva la riga seguente
                *(B+i*k+j)=(float)rand()/((float)RAND_MAX+(float)1);
        }
}
```

- **Distribuzione dei dati**

Successivamente, il processore P0 esegue un broadcast per comunicare a tutti i processori le dimensioni delle matrici.

Vi è poi la chiamata della routine di libreria **`crea_griglia`** che provvede a creare la griglia bidimensionale di processori.

```
crea_griglia(&griglia, &grigliar, &grigliac, menum, p, coordinate);
```

- **Calcolo delle prodotto matriciale parziale e totale**

A questo punto inizia al fase di spedizione e calcolo.

```
	for(i=1;i<nproc;i++)
	{
		//Gli offset sono necessari a P0 per individuare in A il blocco da
	spedire
		offsetC=m/p*(i/p); //Individua la riga da cui partire
		offsetR=n/p*(i%p); //Individua la colonna da cui partire

		for(j=0;j<m/p;j++)
		{
			tag=10+i;
			//Spedisce gli elementi in vettori di dimensione n/p
			MPI_Send(A+offsetC*n+offsetR,   n/p,   MPI_FLOAT,   i,   tag,
MPI_COMM_WORLD);
			offsetC++; //l'offset di colonna viene aggiornato
		}// end for
	}//end for


	for(i=1;i<nproc;i++)
	{
		//Stessa cosa avviene per spedire sottoblocchi di B
		offsetC=n/p*(i/p);
		offsetR=k/p*(i%p);

		for(j=0;j<n/p;j++)
		{
			tag=20+i;
			//Spedisce gli elementi di B in vettori di dimensione k/p
			MPI_Send(B+offsetC*k+offsetR,   k/p,   MPI_FLOAT,   i,   tag,
MPI_COMM_WORLD);
			offsetC++; //l'offset di colonna viene aggiornato
		}//end for interno
	}// end for esterno

	//P0 inizializza il suo blocco subA
	for(j=0;j<m/p;j++)
	{
		for(z=0;z<n/p;z++)
			*(subA+n/p*j+z) = *(A+n*j+z);
	}
```

```
//P0 inizializza il suo blocco subB
for(j=0;j<n/p;j++)
{
      for(z=0;z<k/p;z++)
            *(subB+k/p*j+z) = *(B+k*j+z);
                  }
} //end if di riga 252
```

Il processore P0 calcola la dimensione di A e di B da spedire e spedisce

Se il processore in questione non è il P0, riceve i dati.

```
else
{
      //Tutti i processori ricevono il rispettivo subA
      for(j=0;j<m/p;j++)
      {
            tag=10+menum;
            MPI_Recv(subA+n/p*j, n/p, MPI_FLOAT, 0, tag, MPI_COMM_WORLD, &info);
      }

      //Ricevono subB
      for(j=0;j<n/p;j++)
      {
            tag=20+menum;
            MPI_Recv(subB+k/p*j, k/p, MPI_FLOAT, 0, tag, MPI_COMM_WORLD, &info);
      }
}// end else di riga 302
```

Fatto ciò inizia tutta la comunicazione dei dati lungo la griglia e l’elaborazione del risultato (da riga 303 a 394 del sorgente allegato)

Il programma si conclude con l’eventuale stampa del risultato e dei parametri di valutazione.

Compilazione ed esecuzione

Il programma è studiato per essere il più possibile indipendente dalla piattaforma usata (Linux o Windows).

- Compilazione sotto Linux

Se il programma viene compilato in un sistema Linux, la sintassi è la seguente

`$ mpicc mxm.c <invio>`

Successivamente sarà generato un file a eseguibile

DEVE ESSERE PERESENTE NELLA CARTELLA DOVE SI COMPILA ANCHE IL FILE **MXM-AUX.C**

- Esecuzione del programma sotto Linux

Per eseguire il programma si deve seguire la seguente sintassi:

`$ mpirun –np x –machinefile lista a.out <invio>`

con *x* il numero di processori che si vuole usare.

lista è l'elenco di tutti processori della rete (che coinvolgiamo nel calcolo)

A questo punto il programma in esecuzione chiede quanti dati devono essere calcolati.

Per semplificare il tutto, si può usare uno script di shell, chiamato **`go.sh`**, che ha il compito di semplificare l'uso all'utente, contentente la stringa di sopra (vedi sorgente scritpt)

Si può usare

```
go.sh x
```

con *x* numero di processori che si vuole usare.

Il numero dei processori deve essere un quadrato perfetto tipo 1,4,9,16,25,…

- Compilazione sotto Microsoft Windows XP

Se il programma viene compilato in Windows la sintassi è la seguente

Aprire il **Prompt dei comandi**[1]:

```
C:\>gcc mxv.c -lmpich
```

Successivamente sarà generato un file **a.exe**

- Esecuzione del programma sotto Microsoft Windows[2]

Per eseguire il programma si deve seguire la seguente sintassi:

```
C:\> mpirun –np x a <invio>
```

con *x* il numero di processori che si vuole usare.

A questo punto il programma in esecuzione chiede quanti dati devono essere calcolati.

Si può applicare lo metodo usato per Linux per semplificare l'esecuzione in Windows, usando uno script batch (vedi sorgente).

---

[1] Si presuppone che sia stato installato il compilatore MingW, le librerie MPICH NT e le **Variabili d'ambiente** contengano il path del compialtore e delle librerie

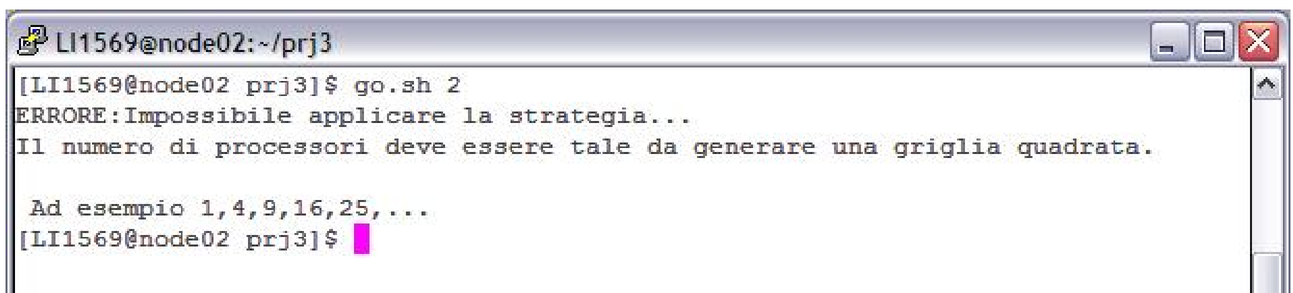
[2] Si presuppone che il sistema abbia identificato la presesenza di 1 o più computer e che sia stato configurato il programma **MPICH Configuration tool**.

Indicatori di errore

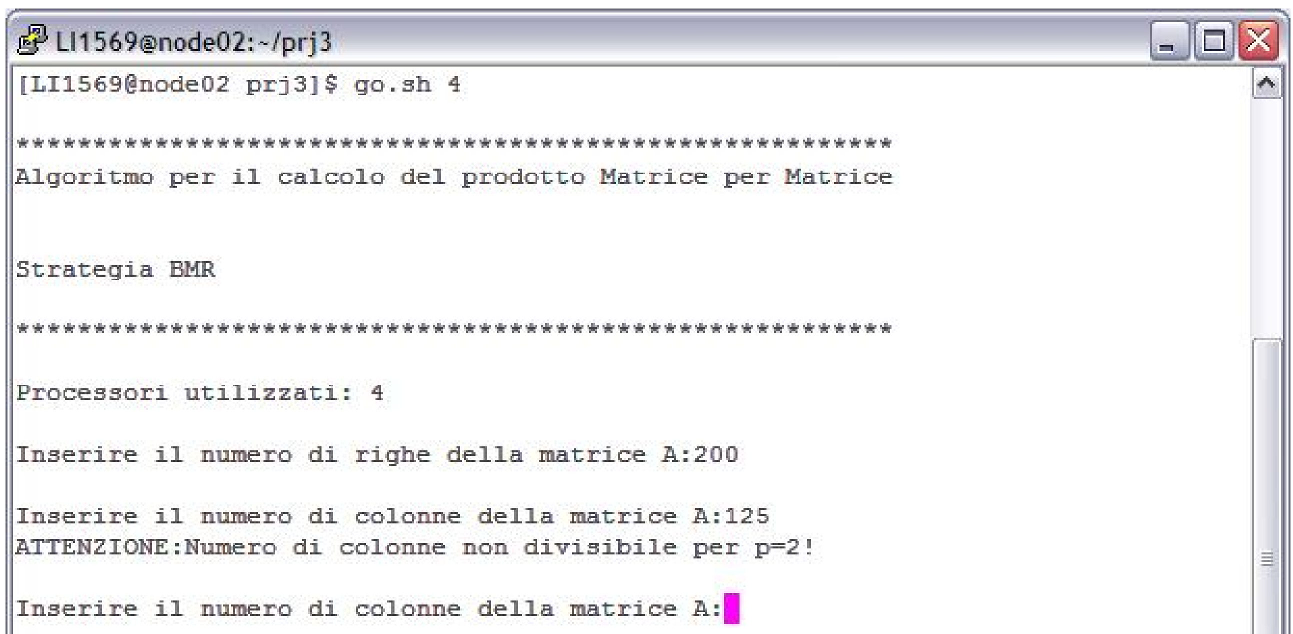
La strategia di BMR si basa sulla creazione di griglie quadrate.

Si possono verificare i seguenti errori:

- il numero di processori non è un quadrato perfetto (1,4,9,16,…)
- il numero di elementi che si inserisce nella griglia non è divisibile per il numero dei processori



Errore sul numero di Processori usati



Errore sul numero di divisibilità degli elementi

Funzioni Utilizzate

Il software descritto si compone di 2 files.

Il primo file rappresenta la funzione chiamante, e in esso troviamo l'utilizzo di alcune routine della libreria MPI per il calcolo parallelo.

Tali funzioni compaiono anche nel secondo file, costituito da una libreria utente contenente alcune funzioni usate dal programma principale

Di seguito, per ognuna di esse si fornisce il prototipo utilizzato nel programma e la descrizione dei relativi parametri di input e di output. Per alcune funzioni, poiché nel programma vi è più di una chiamata, si fornisce il prototipo della prima chiamata rilevata, sottintendendo che il ragionamento è analogo anche per gli altri casi.

Funzioni per inizializzare l'ambiente MPI

- **MPI_Init (&argc, &argv);**
  argc e *argv* sono gli argomenti del main

- **MPI_Comm_rank (MPI_COMM_WORLD, &menum);**
  MPI_COMM_WORLD (input): identificativo del comunicatore entro cui avvengono le comunicazioni

  menum (output): identificativo di processore nel gruppo del comunicatore specificato

- **MPI_Comm_size(MPI_COMM_WORLD, &nproc);**
  MPI_COMM_WORLD (input): nome del comunicatore entro cui avvengono le comunicazioni

  nproc (output): numero di processori nel gruppo del comunicatore specificato

Funzioni di comunicazione collettiva in ambiente MPI

- **MPI_Bcast(&m, 1, MPI_INT, 0, MPI_COMM_WORLD);**
  comunicazione di un messaggio a tutti i processori appartenenti al comunicatore specificato.

I parametri sono:

m: indirizzo dei dati da spedire
1: numero dei dati da spedire
MPI_INT: tipo dei dati da spedire
0 : identificativo del processore che spedisce a tutti
MPI_COMM_WORLD : identificativo del comunicatore entro cui avvengono le comunicazioni

Funzioni di comunicazione bloccante in ambiente MPI

- **MPI_Send(A+offsetC*n+offsetR,n/p, MPI_FLOAT, i, tag, MPI_COMM_WORLD);**

  spedizione di dati

  I parametri sono:
  *A+offsetC*n+offsetR (input):* indirizzo del dato da spedire
  n/p (input): numero dei dati da spedire
  MPI_FLOAT (input): tipo del dato inviato
  i (input): identificativo del processore destinatario
  tag (input): identificativo del messaggio inviato
  MPI_COMM_WORLD (input): comunicatore usato per l'invio del messaggio

- **MPI_Recv (subA+n/p*j, n/p, MPI_FLOAT, 0, tag, MPI_COMM_WORLD, &info);**

  ricezione di dati

  I parametri sono:
  *subA+n/p*j*: indirizzo del dato su cui ricevere
  n/p : numero dei dati da ricevere
  MPI_FLOAT : tipo dei dati da ricevere
  0: identificativo del processore da cui ricevere
  tag (input): identificativo del messaggio
  MPI_COMM_WORLD (input): comunicatore usato per la ricezione del messaggio
  info: vettore che contiene informazioni sulla ricezione del messaggio

Funzioni di comunicazione non bloccante in ambiente MPI

- **MPI_Isend (subB, n/p*k/p, MPI_FLOAT, destinatarior, tag, grigliac, &rqst);**

  spedizione non bloccante di dati

  I parametri sono:

  *subB:*indirizzo del dato da spedire
  *n/p*k/p:*dimensione del dato da spedire
  *MPI_FLOAT:*tipo del dato da spedire
  *destinatarior:*identificativo del destinatario
  *tag:*identificativo del messaggio
  grigliac:comunicatore entro cuo avvengono le comunicazioni
  rqst: oggetto che crea un nesso tra la trasmissione e la ricezione del messaggio

Funzione di sincronizzazione MPI

- **MPI_Barrier (MPI_COMM_WORLD);**
  La funzione fornisce un meccanismo sincronizzante per tutti i processori del comunicatore MPI_COMM_WORLD

- **MPI_Wtime( )**
  Tale funzione restituisce un tempo in secondi

Funzione di chiusura ambiente MPI

- **MPI_Finalize( );**

La funzione determina la fine di un programma MPI. Dopo di essa non si può più chiamare nessuna altra routine MPI.

Funzione di creazione e gestione della topologia a griglia bidimensionale in ambiente MPI

- **MPI_Cart_create (MPI_COMM_WORLD, dim, ndim, period, reorder, griglia);**

Operazione collettiva che restituisce un nuovo comunicatore in cui i processi sono organizzati in una griglia di dimensione *dim* .

I parametri sono:

MPI_COMM_WORLD: identificativo del comunicatore di input
dim: numero di dimensioni della griglia
ndim: vettore di dimensione dim contenente le lunghezze di ciascuna dimensione
period: vettore di dimensione dim contenente la periodicità di ciascuna dimensione
reorder: permesso di riordinare i menum processori (1=si, 0=no)
griglia: nuovo comunicatore di output associato alla griglia

- **MPI_Comm_rank (*griglia, &menum_griglia);**

Analogo alla chiamata in sede di inizializzazione dell'ambiente MPI. Restituisce l'identificativo del processore nella griglia

griglia (input): identificativo del comunicatore entro cui avvengono le comunicazioni

menum_griglia (output): identificativo di processore nel gruppo del comunicatore specificato

- **MPI_Cart_coords (*griglia, menum_griglia, dim, coordinate);**

ogni processo calcola le proprie due coordinate nel nuovo ambiente, cioè nella griglia.

I parametri sono:

griglia : identificativo del comunicatore entro cui avvengono le comunicazioni

menum : identificativo di processore nel gruppo del comunicatore specificato, cioè nella griglia

dim: numero di dimensioni della griglia

coordinate: vettore di dimensione dim i cui elementi rappresentano le coordinate del processore all'interno della griglia

- **MPI_Cart_sub(*griglia, vc, grigliar);**
- **MPI_Cart_sub(*griglia, vc, grigliac);**

crea sotto-griglie di dimensione inferiore a quella originale. Ciascuna sotto-griglia viene identificata da un comunicatore differente.
Partiziona il communicator nel sottogruppo grigliar (o grigliac).

I parametri sono:

griglia: comunicatore di griglia
vc: dimensioni della sottogriglia
grigliar (o *grigliac*): comunicatore che include la sottogriglia contenente i processi desiderati

- **MPI_Cart_rank (grigliac, destRoll, destinatarior);**

determina il rango di un processore a partire dalle coordinate cartesiane del medesimo in una topologia a griglia bidimensionale.

In pratica, date le coordinate cartesiane del progetto ritorna il rank associato.

I parametri sono:

grigliac: comunicatore con topologia cartesiana

destRoll: coordinate del processore del quale si vuole conoscere il rango

destinatarior: rango del processore con coordinate *destRoll*

Il secondo file è una libreria utente contenente alcune function utilizzate per il calcolo, e cioè:

// FUNZIONE crea_griglia

- **void crea_griglia (MPI_Comm *griglia, MPI_Comm *grigliar, MPI_Comm *grigliac, int menum, int lato, int*coordinate)**

crea una griglia bidimensionale periodica utile a combinare i risultati parziali ottenuti da tutti i processori

I parametri sono:

*griglia:*comunicatore che identifica la griglia
grigliar: comunicatore che identifica la sotto-griglia delle righe
grigliac: comunicatore che identifica la sotto-griglia delle colonne
menum: identificativo di processore
*lato:*dimensioni della griglia. In questo caso, poiché la dimensione è unica, la griglia risulterà quadrata
coordinate: vettore di dimensione dim i cui elementi rappresentano le coordinate del processore all'interno della griglia

// Function per il CALCOLO DEL PRODOTTO MATRICE VETTORE

- **double mat_mat_righe(float *A, , int m, int n, float *B, int k, float *C)**

esegue il calcolo del prodotto matrice per matrice

A: matrice di input

B: matrice di input

m,n,k : dimensioni delle matrici

C: matrice risultato

Esempi d'uso

Esempio di lancio su 4 processori

La seguente schermata mostra una esecuzione del programma su un numero eccezionalmente esiguo di valori, ma utili per mostrare come il funzionamento del programma.

Se si inseriscono meno di 100 elementi da calcolare vengono visualizzati i valori inseriti, per velocizzare l'esecuzione del programma, e non perdere tempo nella visualizzazione dei dati generati automaticamente.

```
LI1569@node02:~/prj3
*********************************************************
Algoritmo per il calcolo del prodotto Matrice per Matrice


Strategia BMR

*********************************************************

Processori utilizzati: 4

Inserire il numero di righe della matrice A:128

Inserire il numero di colonne della matrice A:128

Inserire il numero di colonne della matrice B:128


*****************************************************
- Acquisizione elementi di A matrice 128x128 -

*****************************************************



*****************************************************
- Acquisizione elementi di B matrice 128x128 -

*****************************************************

La matrice è di grandi dimensioni

 e non verrà stampata

*****************************************************
- Costruzione della griglia (2x2) di processori -
*****************************************************

*********************************************************
* Stampa del risultato e dei parametri di valutazione *

*********************************************************
Matrice risultato:

La matrice è di grandi dimensioni

 e non verrà stampata


Il tempo di esecuzione su un processore e' 0.055471 secondi
Il tempo di esecuzione su 4 processori e' 0.015369 secondi
Lo Speed Up ottenuto e' 3.609278
L'efficienza risulta essere 0.902320
[LI1569@node02 prj3]$
```

Esempio con esecuzione su un numero di processori primo

Analisi dei tempi

Introduzione

Ora osserveremo le prestazioni che ha il nostro algoritmo quando viene usato. Per fare ciò analizzeremo le seguenti caratteristiche:

- *Tempo di esecuzione utilizzando un numero p>1 processori.*
  Generalmente indicheremo tale parametro con il simbolo T_p

- ***Speed – Up***
  riduzione del tempo di esecuzione rispetto all'utilizzo di un solo processore, utilizzando invece p processori.

  In simboli ……

$$S_p = \frac{T_1}{T_p}$$

  Il valore dello speedup ideale dovrebbe essere pari al numero p dei processori, perciò l'algoritmo parallelo risulta migliore quanto più S_p è prossimo a p.

- **Efficienza**
  Calcolare solo lo speed-up spesso non basta per effettuare una valutazione corretta, poiché occorre *"rapportare lo speed-up al numero di processori"*, e questo può essere effettuato valutando l'efficienza.
  Siano dunque p il numero di processori ed S_p lo speed - up ad esso relativi.
  Si definisce efficienza il parametro……

$$E_p = \frac{S_P}{p}$$

Essa fornisce *un'indicazione di quanto sia stato usato il parallelismo nel calcolatore.*

Idealmente, dovremmo avere che:

$$E_p = 1$$

e quindi l'algoritmo parallelo risulta migliore quanto più E_p è vicina ad 1.

Nelle prossime pagine analizzeremo i tempi delle elaborazioni, presi direttamente dal calcolo, ed in seguito analizzeremo lo speed – up e l'efficienza.

Tempi delle elaborazioni

	800 x 800	1600 x 800	2400 x 1600
NP=1	13,145273	13,197992	39,845088
NP=4	3,486600	3,420317	9,877586
NP=16	1,709943	2,164640	6,429235
NP=25	2,049647	3,039892	8,668707

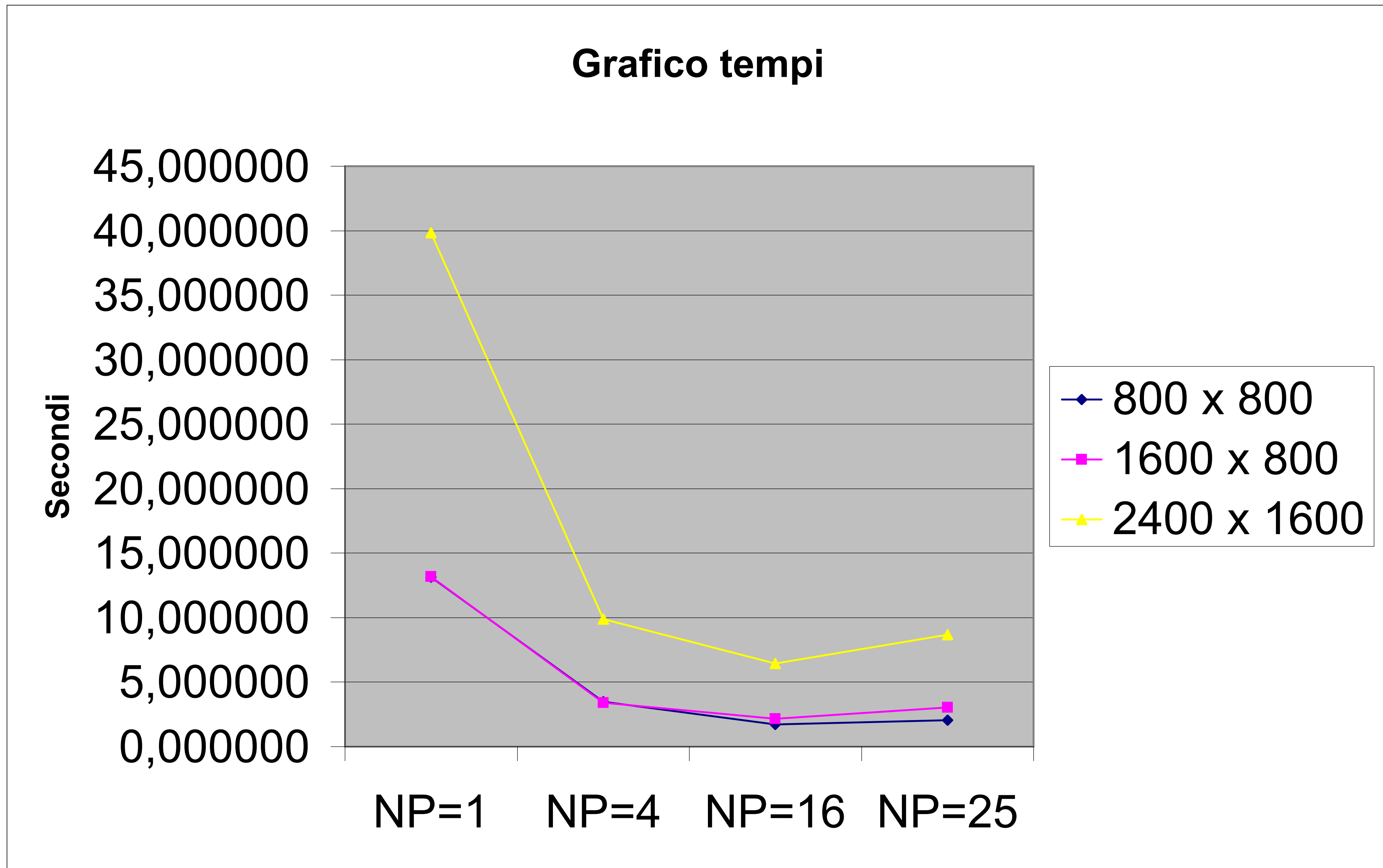


Grafico tempi
Secondi
45,000000
40,000000
35,000000
30,000000
25,000000
20,000000
15,000000
10,000000
5,000000
0,000000
NP=1
NP=4
NP=16
NP=25
800 x 800
1600 x 800
2400 x 1600

Speed - Up

	800 x 800	1600 x 800	2400 x 1600
Sp 1	1	1	1
Sp 4	3,770226869	3,858704325	4,033889252
Sp 16	7,687550404	6,097084042	6,197485082
Sp 25	6,413432655	4,341598978	4,596428049

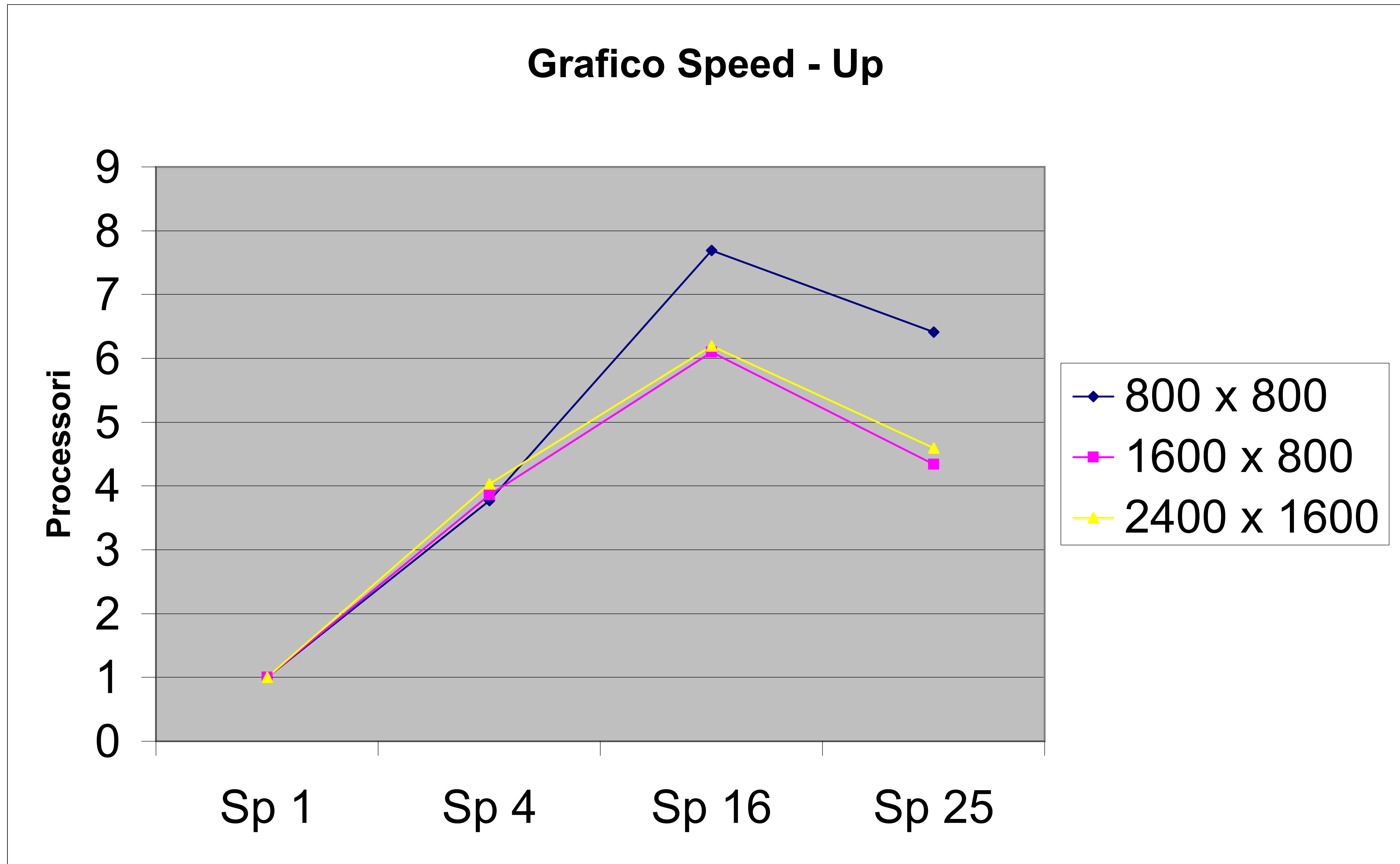


Grafico Speed - Up
Processori
9
8
7
6
5
4
3
2
1
0
Sp 1
Sp 4
Sp 16
Sp 25
800 x 800
1600 x 800
2400 x 1600

Efficienza

	800 x 800	1600 x 800	2400 x 1600
Ep 1	1	1	1
Ep 4	0,942556717	0,964676081	1,008472313
Ep 16	0,4804719	0,381067753	0,387342818
Ep 25	0,256537306	0,173663959	0,183857122

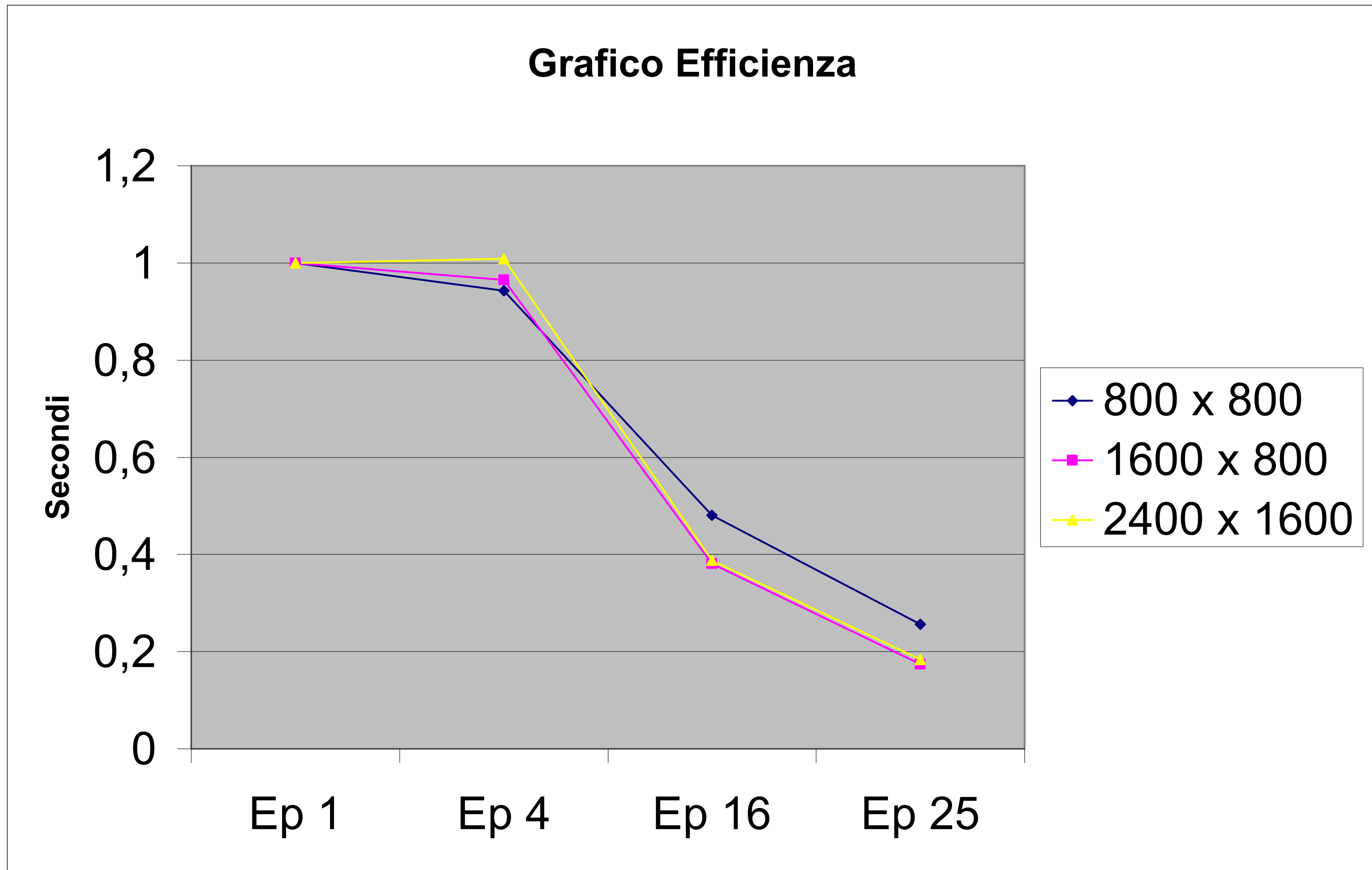


Grafico Efficienza
Secondi
1,2
1
0,8
0,6
0,4
0,2
0
Ep 1
Ep 4
Ep 16
Ep 25
800 x 800
1600 x 800
2400 x 1600

Come è possibile osservare dai test effettuati (pag. 29) per questo tipo di problema la parallellizzazione porta degli oggettivi incrementi di prestazioni nell'ambito dell'efficienza delle prestazioni.

Le curve discendeti di pag. 31 mostrano, soprattutto con valori elevati un miglioramento effettivo delle prestazioni.

Per matrice a di 2400 x 1600 elementi floating point si passa dai 39,845088 secondi di un singolo processore ai 6,429235 secondi di 16 processori.

Può sembrare strano che, improvvisamente si passi a valori più alti pari a 8,668707 secondi di 25 processori.

Il problema è dovuto alla discordanza tra il numero di processori cui si è ripartito il calcolo ed il numero di processori reali dove il calcolo è stato effettuato, che è un valore di molto inferiore a 25.

Ma i dati dimostrano che in linea teorica più il numero di processori aumenta, più aumentano lo speedup e l'efficienza del calcolo.

Bibliografia

1. A.Murli – Lezioni di Calcolo Parallelo – Ed. Liguori

2. Slide del corso di Calcolo Parallelo e distribuito
   http://www.dma.unina.it/~murli/didattica/mat_didattico_nav_cp0708.html

3. www.mat.uniroma1.it/centro-**calcolo**/HPC/materiale-corso/sliMPI.pdf
   (consultato per interessanti approfondimenti su MPI)

4. www.orebla.it/module.php?n=**c**_num_**casuali**
   (consultato per approfondimenti sulla funzione rand per generare numeri casuali)

5. Aitken Peter G., Jones Bradley J. - Programmare in C – Apogeo
   (consultato per ripasso di alcune funzioni)

6. http://www.mcs.anl.gov/research/projects/mpi/www/www3/
   (consultato per alcune funzioni di MPI)

Codice Sorgente

Di seguito viene proposto il codice sorgente del programma sin qui descritto, nella visualizzazione dell’editor di testo Notepad++ Portable per Windows

```
1    /*
2      ***********************************************
3      Algoritmo per il calcolo del Prodotto Matrice Matrice,
4
5      Progetto Realizzato da
6
7      Giovanni Di Cecca
8      Matr. 108/1569
9
10     E-Mail: giovanni.dicecca@gmail.com
11
12     Web Site: http://www.dicecca.net
13
14     ***********************************************
15   */
16
17   // Preprocessore
18   #include <stdio.h>
19   #include <stdlib.h>
20   #include <math.h>
21   #include "mpi.h"
22
23   // Libreria di funzioni
24   #include "mxm-aux.c"
25
26   // Funzione Principale
27   int main(int argc, char *argv[])
28   {
29       //********************************************************
30       // Inizio fase di inizializzazione delle variabili
31       //********************************************************
32
33       int menum, nproc;
34       int m, n, k; // numero di righe e colonne delle matrici
35       int p, q, flag=0, sinc=1, offsetR=0, offsetC=0, dim_prod_par;
36       int N, i, j, z, y, tag, mittente, mittenter, destinatarior;
37       float *A, *B, *C, *subA, *subB, *subC, *Temp, *printTemp;
38
39
40       // Parametri per valutare le prestazioni
41       double t_inizio, t_fine, T1, Tp=0.F, speedup, Ep;
42
43       // Variabili di MPI
44       MPI_Status info;
45       MPI_Request rqst;
46       MPI_Comm griglia, grigliar, grigliac;
47
48
49       int coordinate[2];
50       int mittBcast[2];
51       int destRoll[2];
52       int mittRoll[2];
53
54       //Inizializzazione dell'ambiente MPI
55       MPI_Init(&argc, &argv);
```

```
56
57      //Calcolo dell'identificativo in MPI_COMM_WORLD
58      MPI_Comm_rank(MPI_COMM_WORLD, &menum);
59
60      //Calcolo del numero di processori
61      MPI_Comm_size(MPI_COMM_WORLD, &nproc);
62
63
64      // Controllo sul numero dei processori per garantire l'applicabilità della
    BMR
65      // Se il numero di processori inseriti non è un quadrato perfetto, il
    programma abortisce.
66      // Quindi il processori devono essere 1,4,9,16,25,...
67      if(sqrt(nproc)*sqrt(nproc)!=nproc)
68      {
69          if(menum==0)
70          {
71              printf("ERRORE:Impossibile applicare la strategia...\n");
72              printf("Il numero di processori deve essere tale da generare una
    griglia quadrata.\n\n Ad esempio 1,4,9,16,25,...\n");
73          }
74          MPI_Finalize();
75          return 0;
76      }
77
78      //****************************************************
79      // Inizio fase inserimento dati nelle matrici
80      //****************************************************
81      if(menum==0)
82      {
83          printf("\n*****************************************************\n");
84          printf("Algoritmo per il calcolo del prodotto Matrice per Matrice\n");
85          printf("\n\nStrategia BMR\n");
86          printf("\n*****************************************************\n");
87
88          printf("\nProcessori utilizzati: %d\n", nproc);
89
90          // Procedura di inserimento dati nella Matrice A
91          while(flag==0)
92          {
93              printf("\nInserire il numero di righe della matrice A:");
94              fflush(stdin);
95              scanf("%d", &m);
96
97              //Calcola il valore di p. Esso verrà utilizzato per verificare se
    le dimensioni delle matrici sono divisibili...
98              //...per il numero di processori
99              p=sqrt(nproc);
100
101             //Si richiede che il numero di righe di A sia multiplo di p
102             if(m%p!=0)
103                 printf("ATTENZIONE:Numero di righe non divisibile per p=%d!\n",
     p);
104             else
105                 flag=1;
```

```
106             }
107
108             while(flag==1)
109             {
110                 //Si richiede che il numero di colonne di A sia multiplo di p
111                 printf("\nInserire il numero di colonne della matrice A:");
112                 fflush(stdin);
113                 scanf("%d", &n);
114
115                 //Si richiede che il numero di colonne di A sia multiplo di p
116                 if(n%p!=0)
117                     printf("ATTENZIONE:Numero di colonne non divisibile per
        p=%d!\n", p);
118                 else
119                     flag=0;
120             }
121
122             // Procedura di inserimento dati nella Matrice A
123             while(flag==0)
124             {
125                 printf("\nInserire il numero di colonne della matrice B:");
126                 fflush(stdin);
127                 scanf("%d", &k);
128
129                 //Si richiede che il numero di colonne di B sia multiplo di p
130                 if(k%p!=0)
131                     printf("ATTENZIONE:Numero di colonne non divisibile per
        p=%d!\n", p);
132                 else
133                     flag=1;
134             }
135
136
137             //Allocazione dinamica delle matrici A, B e C solo in P0
138             //Il numero di righe di B è pari al numero di colonne di A
139             A=(float *)malloc(m*n*sizeof(float));
140             B=(float *)malloc(n*k*sizeof(float));
141             C=(float *)calloc(m*k, sizeof(float));
142
143             printf("\n\n\n****************************************************\n");
144             printf("- Acquisizione elementi di A matrice %dx%d -\n", m, n );
145             printf("\n****************************************************\n");
146             for(i=0;i<m;i++)
147             {
148                 for(j=0;j<n;j++)
149                 {
150
151                     // Comandi per l'inserimento dei dati a mano
152                     //printf("Digita elemento A[%d][%d]: ", i, j);
153                     //scanf("%f", A+i*n+j);
154
155                     //per valutare la prestazioni si attiva la riga seguente
156                     *(A+i*n+j)=(float)rand()/((float)RAND_MAX+(float)1);
157                 }
158             }
```

```
159
160         printf("\n\n\n*****************************************************\n");
161         printf("- Acquisizione elementi di B matrice %dx%d -\n",n, k );
162         printf("\n*****************************************************\n");
163         for(i=0;i<n;i++)
164         {
165             for(j=0;j<k;j++)
166             {
167                 // Comandi per l'inserimento dei dati a mano
168                 //printf("Digita elemento B[%d][%d]: ", i, j);
169                 //scanf("%f", B+i*k+j);
170 
171                 //per valutare la prestazioni si attiva la riga seguente
172                 *(B+i*k+j)=(float)rand()/((float)RAND_MAX+(float)1);
173             }
174         }
175 
176 
177 
178         // Stampa della Matrice A solo se sono meno di 100 dati
179         if (m<100 && n<100)
180         {
181             printf("\nMatrice A acquisita:");
182             for(i=0;i<m;i++)
183             {
184                 printf("\n");
185                 for(j=0;j<n;j++)
186                     printf("%.2f\t", *(A+i*n+j));
187             }
188         } // end if
189         else
190         {
191             printf("\nLa matrice è di grandi dimensioni \n");
192             printf("\n e non verrà stampata \n");
193         } // end else
194 
195         // Stampa della Matrice B solo se sono meno di 100 dati
196         if (m<100 && n<100)
197         {
198             printf("\nMatrice B acquisita:");
199             for(i=0;i<n;i++)
200             {
201                 printf("\n");
202                 for(j=0;j<k;j++)
203                     printf("%.2f\t", *(B+i*k+j));
204             }
205         } // end if
206         else
207         {
208             printf("\nLa matrice è di grandi dimensioni \n");
209             printf("\n e non verrà stampata \n");
210         }
211 
212         //*****************************************************************
213         // Inizio fase della creazione della struttura griglia dei dati
```

```
214             //************************************************************
215 
216             printf("\n\n\n*****************************************************\n");
217             printf("- Costruzione della griglia (%dx%d) di processori -", p, p);
218             printf("\n*****************************************************\n");
219         } // end if di riga 81
220 
221         //Il processore P0 esegue un Broadcast del numero di righe di A: m
222         MPI_Bcast(&m, 1, MPI_INT, 0, MPI_COMM_WORLD);
223 
224         //Il processore P0 esegue un Broadcast del numero di colonne di A: n
225         MPI_Bcast(&n, 1, MPI_INT, 0, MPI_COMM_WORLD);
226 
227         //Il processore P0 esegue un Broadcast del numero di colonne di B: k
228         MPI_Bcast(&k, 1, MPI_INT, 0, MPI_COMM_WORLD);
229 
230         //Il processore P0 esegue un broadcast del valore p
231         MPI_Bcast(&p, 1, MPI_INT, 0, MPI_COMM_WORLD);
232 
233         // Carica la funzione crea_griglia (dal file mxm-aux.c)  per creare la
    griglia  in cui ripartire i dati
234         crea_griglia(&griglia, &grigliar, &grigliac, menum, p, coordinate);
235 
236         //Tutti i processori allocano spazio per il blocco subA
237         subA=(float *)malloc(m/p*n/p*sizeof(float));
238 
239         //Tutti allocano spazio per un blocco d'appoggio
240         Temp=(float *)malloc(m/p*n/p*sizeof(float));
241 
242         //Tutti allocano spazio per il blocco subB
243         subB=(float *)malloc(n/p*k/p*sizeof(float));
244 
245         //Tutti allocano spazio per il blocco subC
246         subC=(float *)calloc(m/p*k/p, sizeof(float));
247 
248         printTemp=(float *)malloc(k/p*sizeof(float));
249 
250 
251         if(menum==0)
252         {
253             for(i=1;i<nproc;i++)
254             {
255 
256                 //Gli offset sono necessari a P0 per individuare in A il blocco da
    spedire
257                 offsetC=m/p*(i/p); //Individua la riga da cui partire
258                 offsetR=n/p*(i%p); //Individua la colonna da cui partire
259 
260                 for(j=0;j<m/p;j++)
261                 {
262                     tag=10+i;
263                     //Spedisce gli elementi in vettori di dimensione n/p
264                     MPI_Send(A+offsetC*n+offsetR, n/p, MPI_FLOAT, i, tag,
    MPI_COMM_WORLD);
265                     offsetC++; //l'offset di colonna viene aggiornato
```

```
266                 }// end for
267             }//end for
268 
269 
270             for(i=1;i<nproc;i++)
271             {
272                 //Stessa cosa avviene per spedire sottoblocchi di B
273                 offsetC=n/p*(i/p);
274                 offsetR=k/p*(i%p);
275 
276                 for(j=0;j<n/p;j++)
277                 {
278                     tag=20+i;
279                     //Spedisce gli elementi di B in vettori di dimensione k/p
280                     MPI_Send(B+offsetC*k+offsetR, k/p, MPI_FLOAT, i, tag,
    MPI_COMM_WORLD);
281                     offsetC++; //l'offset di colonna viene aggiornato
282                 }//end for interno
283             }// end for esterno
284 
285 
286             //P0 inizializza il suo blocco subA
287             for(j=0;j<m/p;j++)
288             {
289                 for(z=0;z<n/p;z++)
290                     *(subA+n/p*j+z) = *(A+n*j+z);
291             }
292 
293 
294             //P0 inizializza il suo blocco subB
295             for(j=0;j<n/p;j++)
296             {
297                 for(z=0;z<k/p;z++)
298                     *(subB+k/p*j+z) = *(B+k*j+z);
299             }
300         } //end if di riga 252
301         else
302         {
303             //Tutti i processori ricevono il rispettivo subA
304             for(j=0;j<m/p;j++)
305             {
306                 tag=10+menum;
307                 MPI_Recv(subA+n/p*j, n/p, MPI_FLOAT, 0, tag, MPI_COMM_WORLD, &info);
308             }
309 
310             //Ricevono subB
311             for(j=0;j<n/p;j++)
312             {
313                 tag=20+menum;
314                 MPI_Recv(subB+k/p*j, k/p, MPI_FLOAT, 0, tag, MPI_COMM_WORLD, &info);
315             }
316         }// end else di riga 302
317 
318 
319         //Tutti i processori hanno i rispettivi blocchi subA e subB
```

```
320         for(i=0;i<p;i++)
321         {
322             //al primo passo fa broadcast e prodotto
323             if(i==0)
324             {
325                 //Calcola le coordinate del processore che invierà in broadcast
326                 //la propria matrice subA ai processori sulla stessa riga
327                 mittBcast[0]=coordinate[0];
328                 mittBcast[1]=(i+coordinate[0])%p;
329 
330                 //Calcola le coordinate del processore a cui inviare subB
331                 destRoll[0]=(coordinate[0]+p-1)%p;
332                 destRoll[1]=coordinate[1];
333 
334                 //Calcola le coordinate del processore da cui ricevere la nuova subB
335                 mittRoll[0]=(coordinate[0]+1)%p;
336                 mittRoll[1]=coordinate[1];
337 
338                 //Ricava il rango del destinatario sulla propria colonna
339                 MPI_Cart_rank(grigliac, destRoll, &destinatarior);
340 
341                 //Ricava il rango del mittente sulla propria colonna
342                 MPI_Cart_rank(grigliac, mittRoll, &mittenter);
343 
344                 //Ricava il rango del processore che effettua il Broadcast sulla
    propria riga
345                 MPI_Cart_rank(grigliar, mittBcast, &mittente);
346 
347                 if(coordinate[0] == mittBcast[0] && coordinate[1]==mittBcast[1])
348                 {
349                     //L'esecutore del broadcast copia subA nel blocco temporaneo
    Temp
350                     memcpy(Temp, subA, n/p*m/p*sizeof(float));
351                 }
352 
353                 t_inizio=MPI_Wtime();
354 
355                 MPI_Bcast(Temp, n/p*m/p, MPI_FLOAT, mittente, grigliar);
356 
357                 t_fine=MPI_Wtime();
358                 Tp+=t_fine-t_inizio;
359 
360                 Tp+=mat_mat_righe(Temp, m/p, n/p, subB, k/p, subC);
361             }// end if di riga 324
362 
363             else //Se non è il primo passo esegue Broadcast, Rolling e Prodotto
364             {
365                 mittBcast[0]=coordinate[0];
366                 mittBcast[1]=(i+coordinate[0])%p;
367 
368                 if(coordinate[0] == mittBcast[0] && coordinate[1]==mittBcast[1])
369                 {
370                     //L'esecutore del broadcast copia subA nel blocco temporaneo
    Temp
371                     memcpy(Temp, subA, n/p*m/p*sizeof(float));
```

```
372                 }
373 
374                 mittente=(mittente+1)%p;
375                 t_inizio=MPI_Wtime();
376 
377                 //Broadcast di Temp
378                 MPI_Bcast(Temp, n/p*m/p, MPI_FLOAT, mittente, grigliar);
379 
380                 //Il rolling vede l'invio del blocco subB al processore della riga 
    superiore
381                 tag=30; //La spedizione è non bloccante mentre la ricezione si
382                 MPI_Isend(subB, n/p*k/p, MPI_FLOAT, destinatarior, tag, grigliac, &
    rqst);
383 
384                 //E la ricezione del nuovo blocco subB dalla riga inferiore
385                 tag=30;
386                 MPI_Recv(subB, n/p*k/p, MPI_FLOAT, mittenter, tag, grigliac, &info);
387                 t_fine=MPI_Wtime();
388 
389                 //Calcola il prodotto parziale e il tempo impiegato per eseguirlo
390                 Tp+=mat_mat_righe(Temp, m/p, n/p, subB, k/p, subC)+t_fine-t_inizio;
391 
392             }// end else di riga 364
393 
394         }//end for di riga 321
395 
396 
397         //Tutti i processori in ordine inviano a P0 la propria porzione di C
398         if(menum==0)
399         {
400             //P0 stampa così come le riceve le porzioni ricevute
401             printf("\n***********************************************************\n");
402             printf("* Stampa del risultato e dei parametri di valutazione *\n");
403             printf("\n***********************************************************\n");
404 
405             printf("Matrice risultato:\n");
406 
407             if (m<100 && n<100)
408             {
409                 for(i=0;i<p;i++)
410                 { //per quante sono le righe di C
411                     for(z=0;z<m/p;z++)
412                     { //per quante sono le righe di subC
413                         for(j=0;j<p;j++)
414                         { //per quanti sono i processori per riga
415                             if(i*p+j!=0)
416                             {
417                                 tag=70;
418                                 //Riceve le righe delle varie matrici subC 
    nell'ordine di stampa
419                                 MPI_Recv(printTemp, k/p, MPI_FLOAT, i*p+j, tag, 
    MPI_COMM_WORLD, &info);
420 
421                                 for(y=0;y<k/p;y++)
422                                     //Stampa la porzione di riga di C ricevuta
```

```
423                                 printf(" %.2f\t", *(printTemp+y));
424                         }// end if
425                         else
426                             //P0 stampa le righe della propria matrice subC
427                             for(y=0;y<k/p;y++)
428                                 printf(" %.2f\t", *(subC+k/p*z+y));
429                     }// end for riga 414
430                     printf("\n");
431                 }// end for riga 412
432             }// end for riga 410
433         }// end if riga 408
434         else
435         {
436             printf("\nLa matrice è di grandi dimensioni \n");
437             printf("\n e non verrà stampata \n");
438         } // end else
439 
440         //Calcolo del prodotto con singolo processore e del tempo necessario
    ad eseguirlo
441         T1=mat_mat_righe(A, m, n, B, k, C);
442 
443         speedup=T1/Tp; //Calcola lo speed up
444 
445         Ep=speedup/nproc; //Calcola l'efficienza
446 
447         //Stampa dei risultati ottenuti
448         printf("\n\nIl tempo di esecuzione su un processore e' %f secondi\n",
    T1);
449         printf("Il tempo di esecuzione su %d processori e' %f secondi\n", nproc
    , Tp);
450         printf("Lo Speed Up ottenuto e' %f\n", speedup);
451         printf("L'efficienza risulta essere %f\n\n", Ep);
452 
453         //Deallocazione della memoria allocata da P0 dinamicamente
454         free(A);
455         free(B);
456     }// end if riga 399
457     else
458     {
459         for(i=0;i<m/p;i++)
460         {
461             tag=70;
462             MPI_Send(subC+i*k/p, k/p, MPI_FLOAT, 0, tag, MPI_COMM_WORLD);
463         }
464     }
465 
466     //Deallocazione della memoria allocata dinamicamente da tutti i processori
467     free(Temp);
468     free(printTemp);
469     free(subA);
470     free(subB);
471 
472     MPI_Finalize();
473 
474     return(0);
```

```
475
476     } // end main
477
478
479
```

```
1   /*
2     ***********************************************
3     Algoritmo per il calcolo del Prodotto Matrice Matrice
4
5     Libreria di funzioni ausiliarie
6
7     Progetto Realizzato da
8
9     Giovanni Di Cecca
10    Matr. 108/1569
11
12    E-Mail: giovanni.dicecca@gmail.com
13
14    Web Site: http://www.dicecca.net
15
16    ***********************************************
17  */
18
19
20  // PROTOTIPI DELLE FUNZIONI
21  double mat_mat_righe(float *, int, int, float *, int, float *);
22
23  void crea_griglia(MPI_Comm *, MPI_Comm *, MPI_Comm *, int, int, int *);
24
25
26
27  //**************************
28  // Funzioni ausiliarie
29  //**************************
30
31  // FUNZIONE crea_griglia
32
33  // Crea una griglia bidimensionale periodica utile a combinare i
34  // risultati parziali ottenuti da tutti i processori
35
36  void crea_griglia(MPI_Comm *griglia, MPI_Comm *grigliar, MPI_Comm *grigliac,
    int menum, int lato, int *coordinate){
37
38      int menum_griglia, reorder;
39      int *ndim, *period, dim=2; //Le dimensioni della griglia sono 2
40      int vc[2];
41
42      //Specifica il numero di processori in ogni dimensione
43      //la griglia risulta quadrata,poichè entrambe le componenti di ndim hanno
    lo stesso valore
44      ndim=(int*)calloc(dim, sizeof(int));
45      ndim[0]=lato;
46      ndim[1]=lato;
47
48      period=(int*)calloc(dim, sizeof(int));
49      period[0]=period[1]=1; //La griglia è periodica
50      reorder=0; //la griglia non ha un ordine particolare
51
52      //funzuine MPI che crea la griglia
53      MPI_Cart_create(MPI_COMM_WORLD, dim, ndim, period, reorder, griglia);
```

```
 54
 55     //Ricava il rango del processore all'interno della griglia
 56     MPI_Comm_rank(*griglia, &menum_griglia);
 57
 58     // Il processore di id menum calcola le proprie coordinate...
 59     // ..nel comunicatore griglia creato
 60     MPI_Cart_coords(*griglia, menum_griglia, dim, coordinate);
 61
 62     vc[0]=0;
 63     vc[1]=1;
 64     MPI_Cart_sub(*griglia, vc, grigliar); //Partiziona il communicator nel
    sottogruppo grigliar
 65
 66     vc[0]=1;
 67     vc[1]=0;
 68     MPI_Cart_sub(*griglia, vc, grigliac); //Partiziona il communicator nel
    sottogruppo grigliac
 69
 70 }
 71
 72
 73
 74
 75 //Function per il CALCOLO DEL PRODOTTO MATRICE MATRICE
 76
 77 double mat_mat_righe(float *A, int m, int n, float *B, int k, float *C){
 78
 79     short i, j, z;
 80     double t_inizio, t_fine, tempo=0.F;
 81
 82     t_inizio=MPI_Wtime();
 83     //L'azzeramento di C non viene effettuato  per consentire che chiamate
 84     //successive della function aggiornino la matrice C e non sovrascrivano
 85
 86     for(i=0;i<m;i++)
 87     {
 88         for(z=0;z<n;z++)
 89         {
 90             for(j=0;j<k;j++)
 91             {
 92                 //l'accesso ad A, B e C è per righe
 93                 //calcolo del prodotto scalare tra la riga i-sima di A e la
    colonna j-sima di B
 94                 *(C+i*k+j)+=*(A+i*n+z)*(*(B+k*z+j));
 95             }
 96         }
 97     }
 98
 99     t_fine=MPI_Wtime();
100
101     tempo+=t_fine-t_inizio;
102
103     return tempo;
104 }
105
```

```
1  rem Script Windows che manda in esecuzione
2  rem un file MPI inserendo solo il numero di processori
3  rem
4  rem Esempio d'uso
5  rem
6  rem c:\> go 4
7
8  mpirun -np %1 a
```

```
1   # Questo script serve a mandare in esecuzione il programma inserendo
2   # solo il numero di processori da usare
3
4   # Esempio d'uso in ambiente Linux
5
6   # $ go.sh 4
7
8   mpirun -np $1 -machinefile lista a.out
```