Sa.net - Computer Science dicecca.net

GIOVANNI DI CECCA

Calcolo

Parallelo e Distribuito

Matrice per Matrice Strategia





© 2009 – Giovanni Di Cecca

 $@\ 2021-MONITORE\ NAPOLETANO-\underline{www.monitorenapoletano.it}\\$

Direttore Responsabile: Giovanni Di Cecca

 ${\bf Collana} \; {\bf dicecca.net} \; - \; {\bf Computer} \; \; {\bf Science}$

Anno II - N
º20 — Supplemento al Numero 162 — Agosto
 2021

Periodico Mensile Registrato presso il Tribunale di Napoli № 45 dell'8 giugno 2011

ISSN: 2239-7035

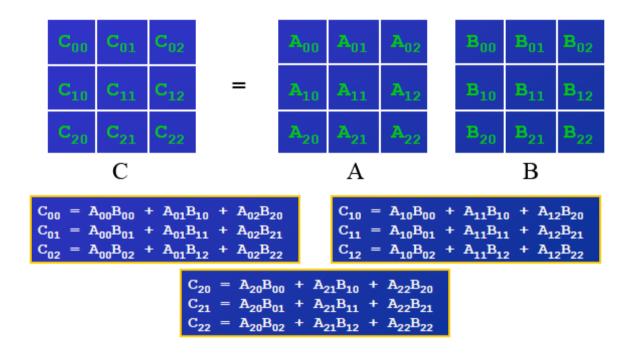
Indice

Definizione ed analisi del Problema.	5
Descrizione dell'algoritmo	7
Introduzione	
Inizializzazione dell'ambiente di calcolo	
Inserimento dei dati	
Distribuzione dei dati	
Calcolo del prodotto matriciale parziale e totale	
Compilazione ed esecuzione	15
Compilazione sotto Linux	
Esecuzione del programma sotto Linux	15
Compilazione sotto Microsoft Windows XP	
Esecuzione del programma sotto Microsoft Windows XP	
Indicatori di errore	17
Funzioni utilzzate	19
Esempi d'uso	25
Analisi dei tempi	27
Introduzione	
Tabella dei tempi di esecuzione	
Tabella di Speed – Up	
Tabella dell'Efficienza	
Commenti ai tempi	
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 n² processi disposto secondo una griglia a topologia bidimensionale, con le seguenti caratteristiche:

- 1.il numero di processi concorrenti è del tipo n²;
- 2.1'ordine delle due matrici è proporzionale al numero di processi



Descrizione dell'Algoritmo

Introduzione

La strategia usata per risolvere il calcolo del prodotto Matrice Matrice è quella della Broadcast Multiply Rolling (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);
```

- <u>Inserimento dei dati</u>

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).

```
{
         // 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("- Acquisizione elementi di B matrice %dx%d -\n",n, k );
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++)</pre>
      {
            //Gli offset sono necessari a PO 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++)</pre>
                  tag=10+i;
                  //Spedisce gli elementi in vettori di dimensione n/p
                  MPI Send(A+offsetC*n+offsetR,
                                                  n/p, MPI FLOAT,
                                                                             tag,
MPI COMM WORLD);
                  offsetC++; //l'offset di colonna viene aggiornato
            }// end for
      }//end for
      for(i=1;i<nproc;i++)</pre>
            //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++)</pre>
            {
                  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,
MPI COMM WORLD);
                  offsetC++; //l'offset di colonna viene aggiornato
            }//end for interno
      }// end for esterno
      //PO inizializza il suo blocco subA
      for(j=0;j<m/p;j++)</pre>
      {
            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</pre>
```

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</pre>
```

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

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.shx

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:

Successivamente sarà generato un file a.exe

- Esecuzione del programma sotto Microsoft Windows²

Per eseguire il programma si deve seguire la seguente sintassi:

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 compilatore 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

```
LI1569@node02:~/prj3

[LI1569@node02 prj3]$ go.sh 2

ERRORE:Impossibile applicare la strategia...

Il numero di processori deve essere tale da generare una griglia quadrata.

Ad esempio 1,4,9,16,25,...

[LI1569@node02 prj3]$
```

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

 θ : 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.

<u>Funzione di creazione e gestione della topologia a griglia bidimensionale in</u> 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

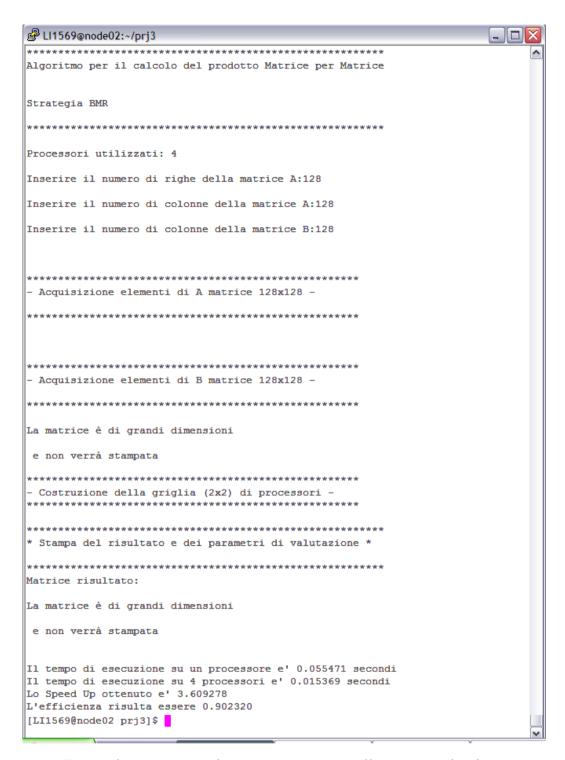
C: matrice risultato

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.



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 <u>efficienza</u> 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:

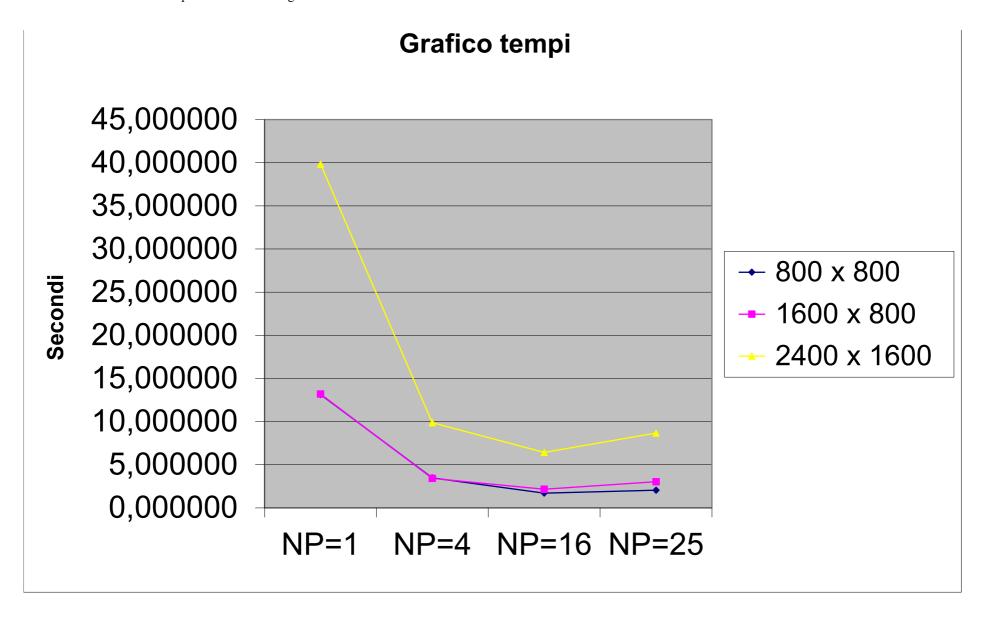
$$E_p = 1$$

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.

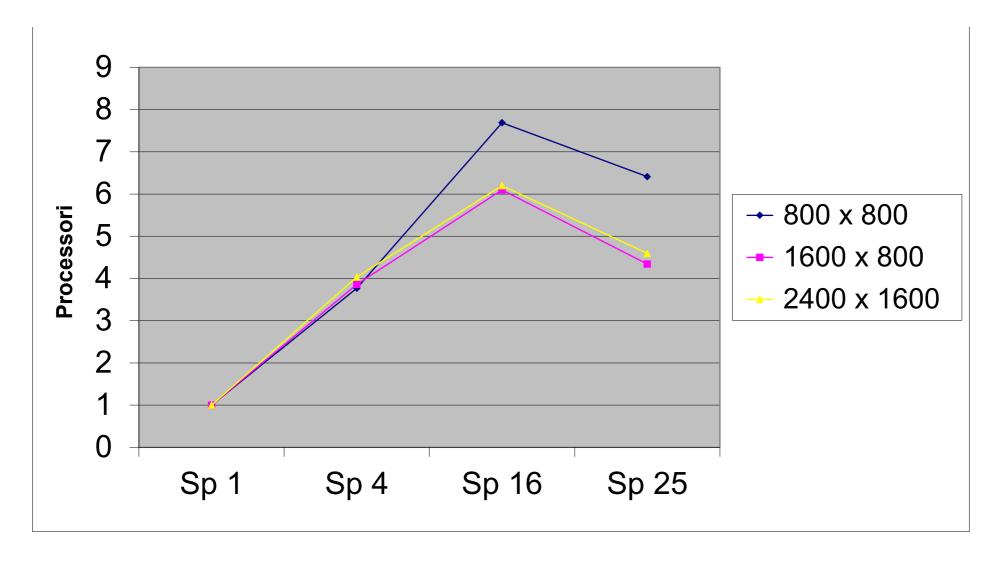
Tempi delle elaborazioni

	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



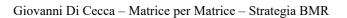
	800 x 800	1600 x 800	2400 x 1600
Sp 1	1	1	1
Sp 4	3,770226869	3,858704325	4,033889252
Sp 16	7,687550404	6,097084042	6,197485082
Sp 25	6,413432655	4,341598978	4,596428049

Grafico Speed - Up



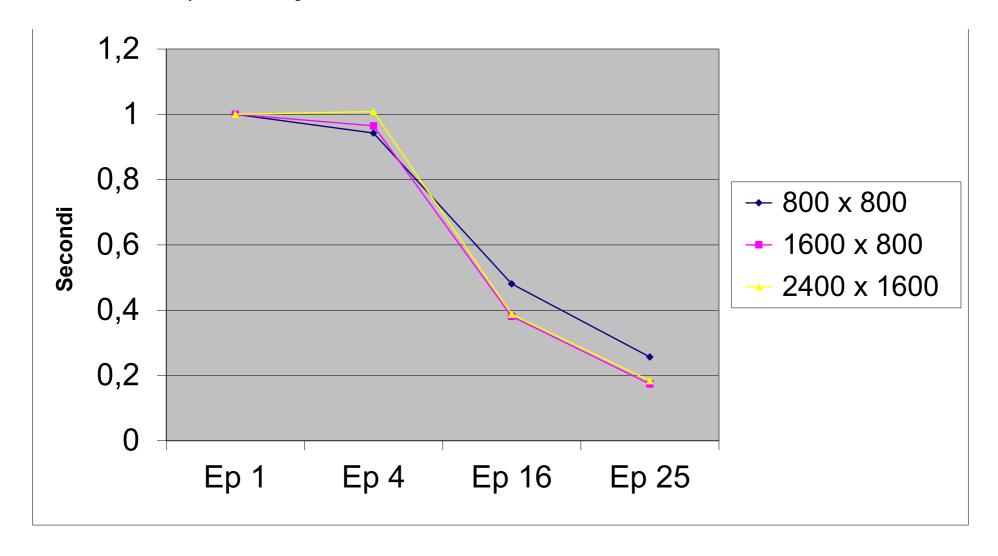
Efficienza

	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



38

Grafico Efficienza



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.** <u>www.mat.uniroma1.it/centro-calcolo/HPC/materiale-corso/sliMPI.pdf</u> (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
9
10
11
      Web Site: http://www.dicecca.net
12
13
14
      ***********
15
    */
16
17
    // Preprocessore
18
    #include <stdio.h>
19
    #include <stdlib.h>
   #include <math.h>
20
21
    #include "mpi.h"
22
2.3
    // Libreria di funzioni
24
    #include "mxm-aux.c"
25
26
    // Funzione Principale
27
    int main(int argc, char *argv[])
28
        //**************
29
        // 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;
        int N, i, j, z, y, tag, mittente, mittenter, destinatarior;
36
37
        float *A, *B, *C, *subA, *subB, *subC, *Temp, *printTemp;
38
39
40
        // Parametri per valutare le prestazioni
        double t inizio, t fine, T1, Tp=0.F, speedup, Ep;
41
42
43
        // Variabili di MPI
44
        MPI Status info;
45
        MPI Request rqst;
        MPI Comm griglia, grigliar, grigliac;
46
47
48
49
        int coordinate[2];
50
        int mittBcast[2];
51
        int destRoll[2];
52
        int mittRoll[2];
53
54
        //Inizializzazione dell'ambiente MPI
        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
        // Se il numero di processori inseriti non è un quadrato perfetto, il
65
     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\ 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
84
            printf("Algoritmo per il calcolo del prodotto Matrice per Matrice\n");
            printf("\n\nStrategia BMR\n");
8.5
            86
87
            printf("\nProcessori utilizzati: %d\n", nproc);
88
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
                //Calcola il valore di p. Esso verrà utilizzato per verificare se
97
     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)
                   printf("ATTENZIONE: Numero di righe non divisibile per p=%d!\n",
103
      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);
                 scanf("%d", &n);
113
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);
                 scanf("%d", &k);
127
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 PO
138
             //Il numero di righe di B è pari al numero di colonne di A
139
             A=(float *)malloc(m*n*sizeof(float));
             B=(float *)malloc(n*k*sizeof(float));
140
141
             C=(float *)calloc(m*k, sizeof(float));
142
             143
144
             printf("- Acquisizione elementi di A matrice %dx%d -\n", m, n );
             145
146
             for (i=0;i<m;i++)</pre>
147
             1
148
                 for (j=0;j<n;j++)</pre>
149
                 {
150
151
                     // Comandi per l'inserimento dei dati a mano
                     //printf("Digita elemento A[%d][%d]: ", i, j);
152
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
161
             printf("- Acquisizione elementi di B matrice %dx%d -\n",n, k );
             162
163
             for(i=0;i<n;i++)</pre>
164
             {
165
                 for (j=0;j<k;j++)</pre>
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
             if (m<100 && n<100)
179
180
181
                printf("\nMatrice A acquisita:");
182
                for (i=0;i<m;i++)</pre>
183
184
                    printf("\n");
185
                    for (j=0; j<n; j++)</pre>
186
                        printf("%.2f\t", *(A+i*n+j));
187
                 }
188
             } // end if
189
             else
190
                printf("\nLa matrice è di grandi dimensioni \n");
191
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
             {
                printf("\nMatrice B acquisita:");
198
199
                for (i=0;i<n;i++)</pre>
200
                 {
201
                    printf("\n");
202
                    for (j=0; j<k; j++)</pre>
                        printf("%.2f\t", *(B+i*k+j));
203
204
                 }
205
             } // end if
206
             else
207
             {
                printf("\nLa matrice è di grandi dimensioni \n");
208
209
                printf("\n e non verrà stampata \n");
210
             }
211
             //*******************
212
213
             // Inizio fase della creazione della struttura griglia dei dati
```

```
//**********************
214
215
             216
217
             printf("- Costruzione della griglia (%dx%d) di processori -", p, p);
             218
219
         } // end if di riga 81
220
221
         //Il processore PO esegue un Broadcast del numero di righe di A: m
222
         MPI Bcast(&m, 1, MPI INT, 0, MPI COMM WORLD);
223
         //Il processore PO eseque un Broadcast del numero di colonne di A: n
224
225
         MPI Bcast(&n, 1, MPI INT, 0, MPI COMM WORLD);
226
227
         //Il processore PO esegue un Broadcast del numero di colonne di B: k
         MPI Bcast(&k, 1, MPI INT, 0, MPI COMM WORLD);
228
229
230
         //Il processore PO esegue un broadcast del valore p
         MPI Bcast(&p, 1, MPI INT, 0, MPI COMM WORLD);
231
232
         // Carica la funzione crea griglia (dal file mxm-aux.c) per creare la
233
     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
         subA=(float *)malloc(m/p*n/p*sizeof(float));
237
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
         printTemp=(float *)malloc(k/p*sizeof(float));
248
249
250
251
         if (menum==0)
252
253
             for(i=1;i<nproc;i++)</pre>
254
             {
255
256
                 //Gli offset sono necessari a PO per individuare in A il blocco da
     spedire
257
                 offsetC=m/p*(i/p); //Individua la riga da cui partire
                 offsetR=n/p*(i%p); //Individua la colonna da cui partire
258
259
260
                 for (j=0;j<m/p;j++)</pre>
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++)</pre>
271
               {
                    //Stessa cosa avviene per spedire sottoblocchi di B
272
273
                   offsetC=n/p*(i/p);
274
                   offsetR=k/p*(i%p);
2.75
276
                   for (j=0;j<n/p;j++)</pre>
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
               }// end for esterno
283
284
285
286
               //PO inizializza il suo blocco subA
287
               for (j=0;j<m/p;j++)</pre>
288
               {
289
                   for(z=0;z<n/p;z++)</pre>
290
                        *(subA+n/p*j+z) = *(A+n*j+z);
291
               }
292
293
294
               //PO inizializza il suo blocco subB
295
               for (j=0;j<n/p;j++)</pre>
296
               {
297
                   for (z=0; z<k/p; z++)</pre>
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++)</pre>
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
               for (j=0;j<n/p;j++)</pre>
311
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++)</pre>
321
          {
322
              //al primo passo fa broadcast e prodotto
323
              if(i==0)
324
              {
325
                  //Calcola le coordinate del processore che invierà in broadcast
                  //la propria matrice subA ai processori sulla stessa riga
326
327
                  mittBcast[0]=coordinate[0];
328
                  mittBcast[1]=(i+coordinate[0])%p;
329
330
                  //Calcola le coordinate del processore a cui inviare subB
                  destRoll[0] = (coordinate[0] +p-1) %p;
331
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
                  MPI Cart rank (grigliac, mittRoll, &mittenter);
342
343
                  //Ricava il rango del processore che effettua il Broadcast sulla
344
      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;
                 t_inizio=MPI_Wtime();
375
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
                 MPI Isend(subB, n/p*k/p, MPI FLOAT, destinatarior, tag, grigliac, &
382
     rqst);
383
                 //E la ricezione del nuovo blocco subB dalla riga inferiore
384
385
                 taq=30;
                 MPI Recv(subB, n/p*k/p, MPI FLOAT, mittenter, tag, grigliac, &info);
386
                 t fine=MPI Wtime();
387
388
                 //Calcola il prodotto parziale e il tempo impiegato per eseguirlo
389
390
                 Tp+=mat mat righe (Temp, m/p, n/p, subB, k/p, subC)+t fine-t inizio;
391
             }// end else di riga 364
392
393
394
         }//end for di riga 321
395
396
397
         //Tutti i processori in ordine inviano a PO la propria porzione di C
398
         if (menum==0)
399
400
             //PO stampa così come le riceve le porzioni ricevute
             401
402
             printf("* Stampa del risultato e dei parametri di valutazione *\n");
             403
404
405
             printf("Matrice risultato:\n");
406
             if (m<100 && n<100)
407
408
409
                 for (i=0;i<p;i++)</pre>
410
                 { //per quante sono le righe di C
411
                     for(z=0;z<m/p;z++)</pre>
                     { //per quante sono le righe di subC
412
413
                        for (j=0;j<p;j++)</pre>
414
                         { //per quanti sono i processori per riga
415
                            if(i*p+j!=0)
416
                            {
417
                                taq=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++)</pre>
422
                                    //Stampa la porzione di riga di C ricevuta
```

```
423
                                       printf(" %.2f\t", *(printTemp+y));
424
                               }// end if
425
                               else
426
                                   //PO stampa le righe della propria matrice subC
427
                                   for (y=0;y<k/p;y++)</pre>
                                       printf(" %.2f\t", *(subC+k/p*z+y));
428
                           }// end for riga 414
429
430
                           printf("\n");
431
                       }// end for riga 412
                   }// end for riga 410
432
433
              }// end if riga 408
              else
434
435
436
                  printf("\nLa matrice è di grandi dimensioni \n");
437
                  printf("\n e non verrà stampata \n");
438
              } // end else
439
              //Calcolo del prodotto con singolo processore e del tempo necessario
440
      ad eseguirlo
441
              T1=mat mat righe(A, m, n, B, k, C);
442
              speedup=T1/Tp; //Calcola lo speed up
443
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);
              printf("L'efficienza risulta essere %f\n\n", Ep);
451
452
453
              //Deallocazione della memoria allocata da PO dinamicamente
454
              free(A);
455
              free (B);
456
          }// end if riga 399
457
          else
458
          {
459
              for (i=0;i<m/p;i++)</pre>
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
      Matr. 108/1569
10
11
      E-Mail: giovanni.dicecca@gmail.com
12
13
14
      Web Site: http://www.dicecca.net
15
      ***********
16
17
    */
18
19
20
    // PROTOTIPI DELLE FUNZIONI
    double mat_mat_righe(float *, int, int, float *, int, float *);
21
22
23
    void crea griglia(MPI Comm *, MPI Comm *, MPI Comm *, int, int *);
24
25
26
    //********
27
28
    // Funzioni ausiliarie
29
    //*********
30
31
    // FUNZIONE crea griglia
32
    // Crea una griglia bidimensionale periodica utile a combinare i
33
34
    // risultati parziali ottenuti da tutti i processori
35
    void crea griglia (MPI Comm *griglia, MPI Comm *grigliar, MPI_Comm *grigliac,
36
    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
        period=(int*)calloc(dim, sizeof(int));
48
        period[0]=period[1]=1; //La griglia è periodica
49
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...
          // ..nel comunicatore griglia creato
 59
          MPI Cart coords (*griglia, menum griglia, dim, coordinate);
 60
 61
 62
          vc[0]=0;
 63
          vc[1]=1;
          MPI Cart sub(*griglia, vc, grigliar); //Partiziona il communicator nel
      sottogruppo grigliar
 65
 66
          vc[0]=1;
 67
          vc[1]=0;
          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
      double mat mat righe (float *A, int m, int n, float *B, int k, float *C) {
 77
 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++)</pre>
 87
          {
 88
              for (z=0; z<n; z++)</pre>
 89
 90
                  for (j=0;j<k;j++)</pre>
 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
```

rem Script Windows che manda in esecuzione
rem un file MPI inserendo solo il numero di processori
rem
rem Esempio d'uso
rem
rem c:\> go 4

8

mpirun -np <mark>%1</mark> a

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

LIBERTÀ EGUAGLIANZA

MONITORE NAPOLETANO

Fondato nel 1799 da Carlo Lauberg ed Eleonora de Fonseca Pimentel Rifondato nel 2010 Direttore: Giovanni Di Cecca

Anno CCXXII

Contatti



C.Ph.: +39 392 842 76 67



www.monitorenapoletano.it



info@monitorenapoletano.it