UNIVERSITÀ DEGLI STUDI DI SALERNO

FACOLTÀ DI SCIENZE NATURALI FISICHE E MATEMATICHE

Corso di laurea in Informatica

Tesina di Programmazione Concorrente e Parallela

L'INSIEME DI MANDELBROT

Professori

Prof. Scarano Vittorio

Dott. De Chiara Rosario

Studente

Costante Luca

Matr. 0521000838

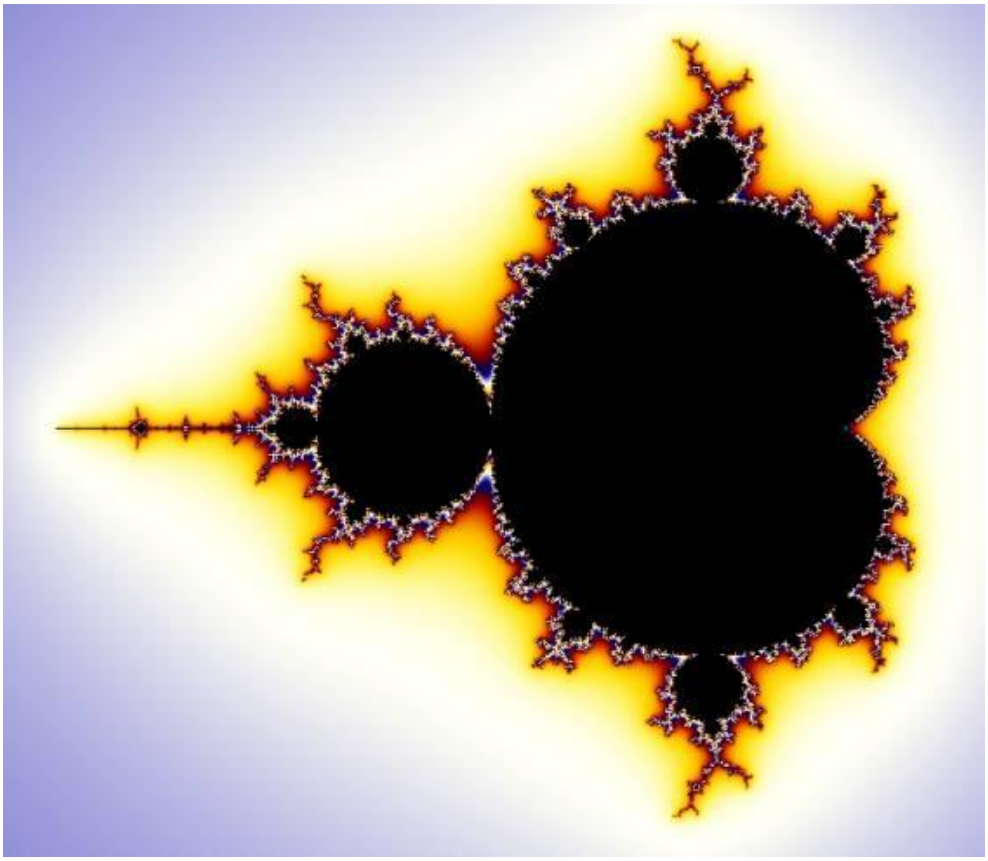
INDICE

INTRODUZIONE

"Why is geometry often described as 'cold' and 'dry'? One reason lies in its inability to describe the shape of a cloud, a mountain, a coastiline or a tree. Clouds are not spheres, mountains are not cones, coastlines are not circles, and bark is not smooth, nor does lightning travel in a straight line"

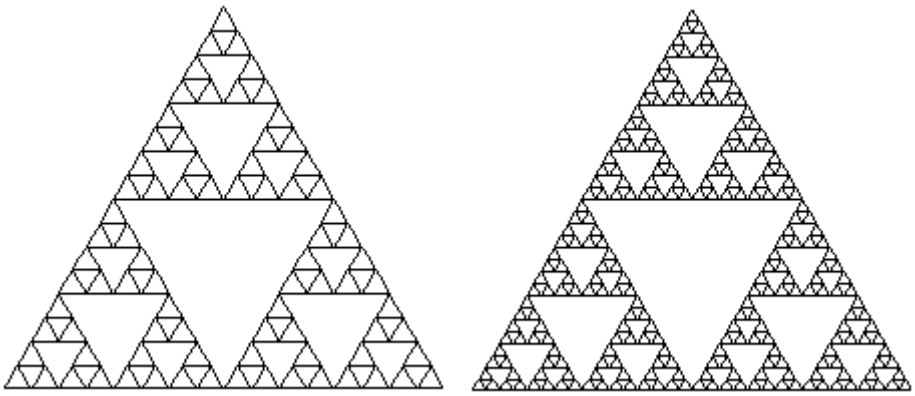
-- Benoit B. Mandelbrot --

Così Mandelbrot nel suo libro The Fractal Geometry of Nature descrive l'inadeguatezza della geometria euclidea nella descrizione della natura. Mandelbrot è il padre fondatore della teoria dei frattali e inventore del famoso insieme che porta il suo nome.



Cos'è un frattale?

La definizione più semplice e intuitiva lo descrive come una figura geometrica in cui un motivo identico si ripete su scala continuamente ridotta. Questo significa che ingrandendo la figura si otterranno forme ricorrenti e ad ogni ingrandimento essa rivelerà nuovi dettagli. Contrariamente a qualsiasi altra figura geometrica un frattale invece di perdere dettaglio quando è ingrandito, si arricchisce di nuovi particolari.



Il termine frattale fu coniato da Mandelbrot e ha origine nel termine latino fractus, poichè la dimensione di un frattale non è intera.

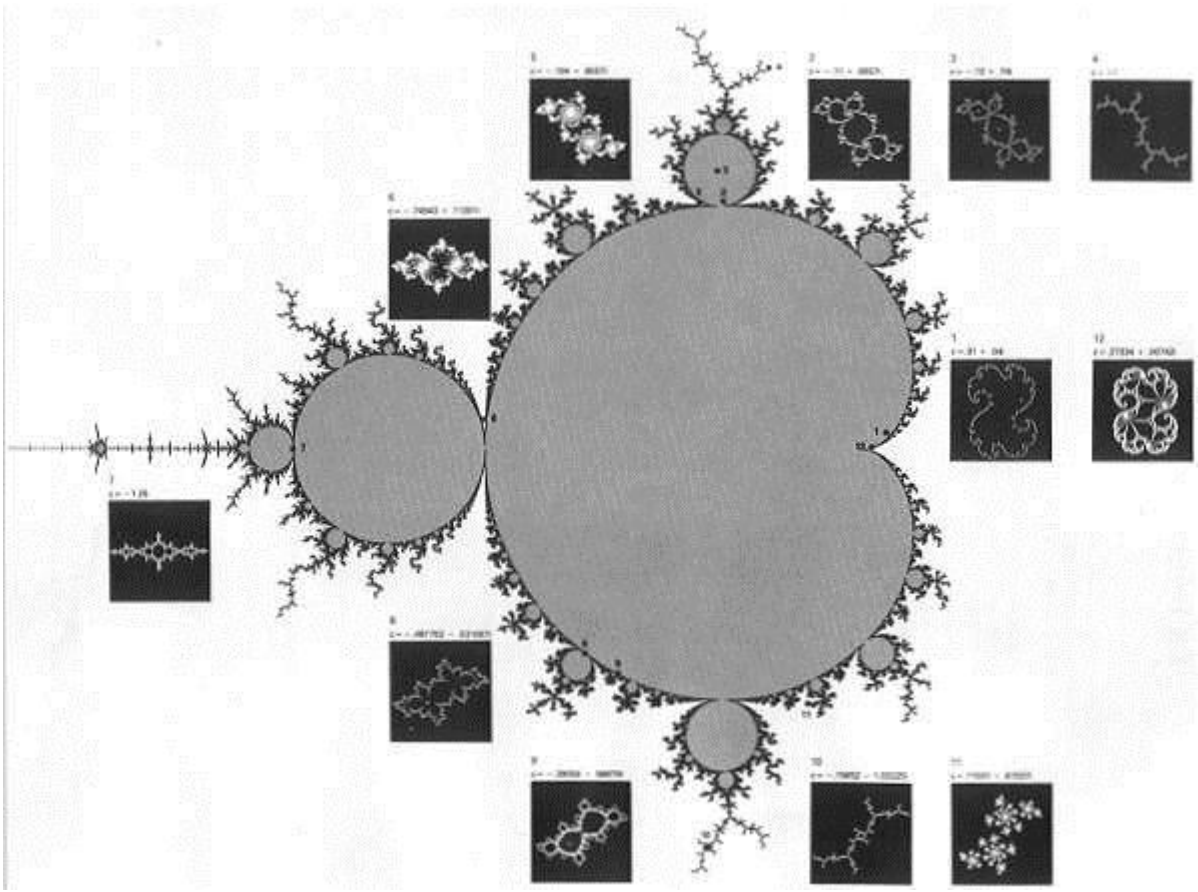
1.1. **Matematica dei frattali**

[1] I frattali sono figure geometriche caratterizzate dal ripetersi sino all'infinito di uno stesso motivo su scala sempre più ridotta. Questa è la definizione più intuitiva che si possa dare di figure che in natura si presentano con una frequenza impressionante ma che non hanno ancora una definizione matematica precisa: l'atteggiamento corrente è quello di considerare frattale un insieme F che abbia proprietà simili alle quattro elencate qui di seguito:

- ✓ **Autosimilarità**: F è unione di un numero di parti che, ingrandite di un certo fattore, riproducono tutto F; in altri termini F è unione di copie di se stesso a scale differenti.
- ✓ **Struttura fine:** F rivela dettagli ad ogni ingrandimento.

- ✓ **Irregolarità:** F non si può descrivere come luogo di punti che soddisfano semplici condizioni geometriche o analitiche. La funzione è ricorsiva: $F = \{Z \mid Z = f(f(f(\ldots)))\}$
- ✓ **Dimensioni di autosimilarità > della dimensione topologica:** La caratteristica di queste figure, caratteristica dalla quale deriva il loro nome, è che, sebbene esse possano essere rappresentate (se non si pretende di rappresentare infinite iterazioni, cioè trasformazioni per le quali si conserva il particolare motivo geometrico) in uno spazio convenzionale a due o tre dimensioni, la loro dimensione non è intera. In effetti la lunghezza di un frattale "piano" non può essere misurata definitamene, ma dipende strettamente dal numero di iterazioni al quale si sottopone la figura iniziale.

1.1.1. L'insieme di Mandelbrot

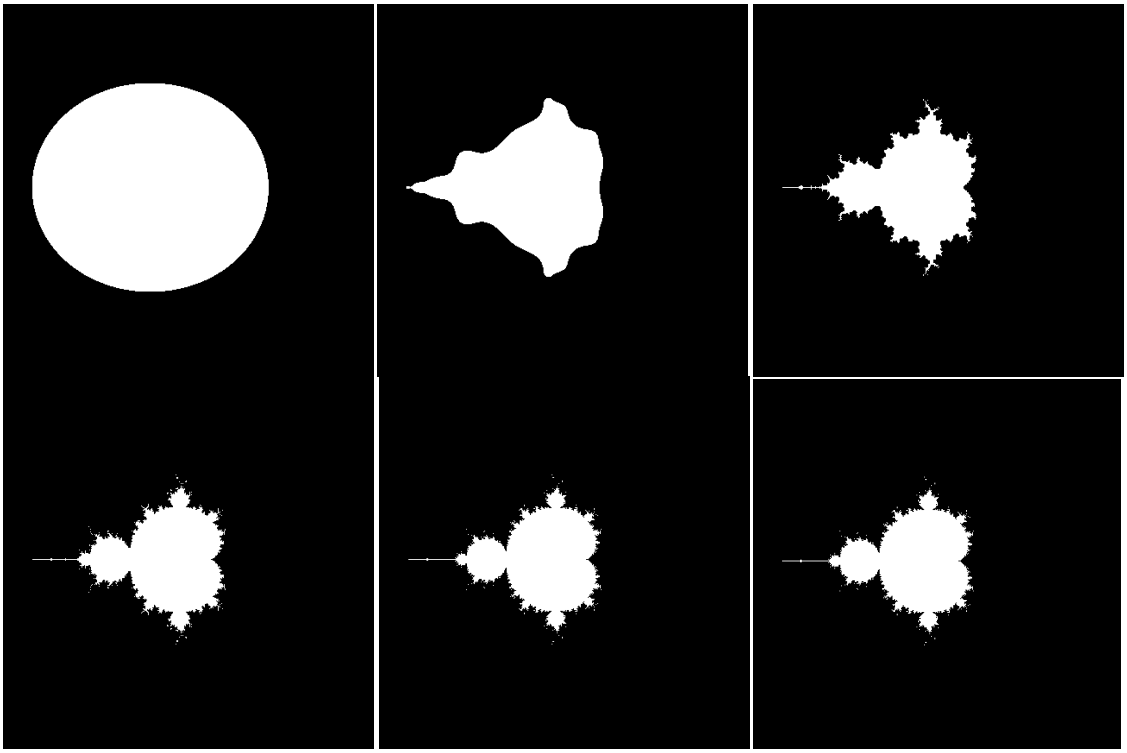


L'insieme di Mandelbrot si presenta come un otto disposto in orizzontale sfrangiato e simmetrico rispetto all'asse delle ascisse.

1. Si supponga di considerare una piccola porzione attorno all'origine di un piano complesso di intervallo (-2, -2) - (2, 2).
2. Si sostituiscano, per ogni punto considerato, le corrispondenti coordinate complesse al termine noto c, nell'equazione $Z = z^2 + c$, ponendo inizialmente $z = 0 + 0i$.
3. Si calcoli il valore di Z.
4. Se si trova che la distanza di Z dall'origine è maggiore di due, si salti al passo 6. Per calcolare tale distanza, si adotta il teorema di pitagora nel seguente modo: dato $Z = a + bi$, il quadrato della distanza d dall'origine sarà $d^2 = a^2 + b^2$
5. Se no, si incrementi di 1 un contatore e si torni al passo 3 se il contatore ha un valore inferiore al numero di iterazioni massime prefissato, dopo aver posto $z = Z$.
6. Si colori il punto di un colore diverso a seconda del valore del contatore.
7. Si azzeri il contatore e si ritorni al passo 2, per calcolare il colore del prossimo punto.
8. Il procedimento avrà termine quando tutti i punti interessati saranno stati processati nel suddetto modo.

N. B. Il numero di iterazioni massime determina la precisione da adottare per la rappresentazione dell'insieme e per ottenere migliori risultati, è consigliabile che sia uguale o comunque inferiore al numero di colori disponibili. Buoni risultati si ottengono assegnando a c dei coefficienti reali ed immaginari compresi fra 0 ed 1.

Questa immagine mostra l'insieme di Mandelbrot ottenuta con un numero crescente di iterazioni massime: come si può notare, la precisione del disegno dei confini diventa sempre più accurata.



1.2. **Implementazione dell'insieme di Mandelbrot**

[2] L'insieme di Mandelbrot o frattale di Mandelbrot è uno dei frattali più popolari, conosciuto anche al di fuori dell'ambito matematico per le suggestive immagini multicolori che ne sono state divulgate.

È l'insieme dei numeri complessi c per i quali è limitata la successione definita da:
$z_{n+1} = {z_n}^2 + c$
con
$z_0 = 0$.

1.2.1. Implementazione sequenziale

La seguente implementazione sequenziale è stata realizzata in c++.

```
/*
L'insieme di Mandelbrot è definito come l'insieme dei numeri complessi
c tale per cui
non è divergente  la successione definita da:
    z_n+1 = z_n^2 + c
con
    z_0 = 0
```

```
L'insieme è un frattale e, nonostante la semplicità della definizione,
ha una forma non banale.

Utilizzata la libreria EasyMBP per il salvataggio dell'immagine BMP
(per maggiori info visitare http://easybmp.sourceforge.net/)
*/

#include "EasyBMP.h"
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>

//prototipo della funzione che restituisce il colore del pixel di una
determinata
//coordinata
RGBApixel mandel(int, int);

//variabili globali
int maxiteration=1000;
double cx;                          /* i rispettivi valori di c
(x,y)*/
double cy;
double magnify = 1.0;         /* no magnification     (zoom)*/
int rows=1600;
int cols=1600;
const double EscapeRadius=6.0;

int main(){
      //inizializzazione oggetto immagine
      BMP Bmp;
      Bmp.SetSize( rows, cols);
      Bmp.SetBitDepth(24);
      clock_t start,end;

      double tempo;
      start=clock();
      for (int y0=0;y0<rows;y0++){
            for (int x0=0;x0<cols;x0++){
                  cx=(((float)x0)/((float)cols)-0.5)/magnify*3.0-0.7;
                  cy=(((float)y0)/((float)rows)-0.5)/magnify*3.0;

                  RGBApixel px=mandel(x0,y0);
                  Bmp.SetPixel(x0,y0,px);
            }
      }
      Bmp.WriteToFile( "mandelbrot.bmp" );
      end=clock();
      tempo=((double)(end-start))/CLOCKS_PER_SEC;
      printf("%f",tempo);
}

RGBApixel mandel(int xpt, int ypt){

      RGBApixel pixel;
      double xtemp=0;
      double x=0;
      double y=0;

      int iteration=0;
```

```
	while ((x*x+y*y<=(EscapeRadius*EscapeRadius)) &&
iteration<maxiteration){
		xtemp=x*x-y*y + cx;
		y=2.0*x*y+cy;
		x=xtemp;
		iteration++;
	}

	if (iteration == maxiteration){
		//color black
		pixel.Red = 0;
		pixel.Green = 0;
		pixel.Blue = 0;
	}
	else{
		//color = iteration
		double  mu = iteration+1-((log(log(x*x+y*y)))/log(2.0));
		mu=mu*EscapeRadius;
		pixel.Red=53+mu;
		pixel.Green=37+mu;
		pixel.Blue=208+mu;
	}
	return pixel;
}
```

Per la compilazione è stato utilizzato Visual Studio 2008. Una volta ottenuto il file .exe, lanciamo l'esecuzione. Otteniamo un file ".bmp" contenente l'immagine dell'insieme di Mandelbrot.

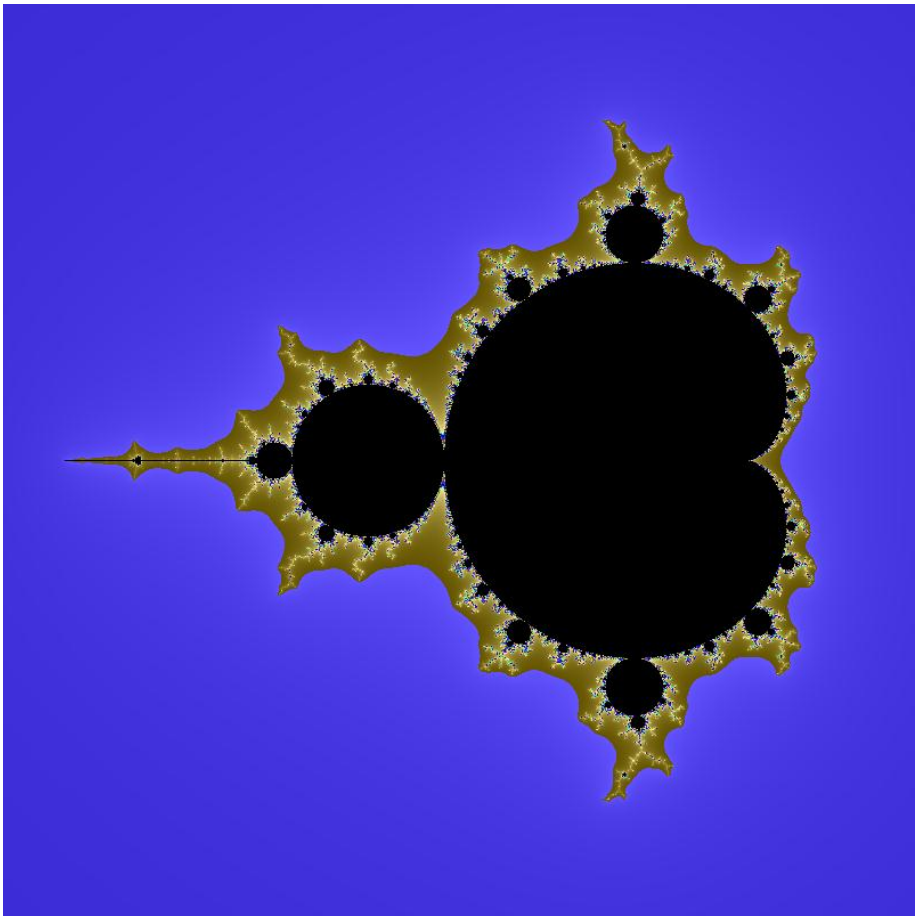
Per la memorizzazione dell'immagine, è stata utilizzata una libreria [3] EasyMBP la quale offre dei semplici metodi per la manipolazione di file BMP. Per l'utilizzo della libreria, bisogna copiare i sorgenti di essa nel progetto. Fatto ciò, basta specificare il nome della libreria `#include <EasyBMP.h>` nel nostro sorgente per poterla utilizzare. Bisogna dichiarare un oggetto di tipo `BMP Bmp;` e successivamente impostare la dimensione in pixel dell'immagine (nel nostro esempio l'immagine è 1600x1600 px). Ogni pixel viene rappresentato attraverso l'oggetto `RGBApixel pixel;` dove bisogna specificare i rispettivi valori di Red, Green e Blue. Alla fine, basta richiamare il metodo `Output.WriteToFile( "mandelbrot.bmp" );` dell'oggetto BMP ottenendo così la nostra immagine dell'insieme di Mandelbrot.

Il main (all'interno dei 2 cicli for) calcola il colore di ogni pixel della funzione di Mandelbrot richiamando la funzione mandel(int,int) e come parametro impostiamo le coordinate del punto x,y da calcolare.

Sono stati utilizzati i seguenti parametri per disegnare l'insieme:

- Numero massimo di iterazioni: 1000;
- Magnify (parametro che rappresenta lo zoom rispetto alla coordinata 0,0): l'oggetto ossia senza zoom;
- Numero di righe: 10000;
- Numero di colonne: 10000;
- EscapeRadius (più questo valore viene aumentato, più il raggio di fuga è ampio): 6.0;

L'immagine ottenuta è la seguente ed il tempo per calcolare è stato di 204.962 secondi. Questo tempo è relativo solo al calcolo sequenziale dell'immagine.



1.2.2. Implementazione parallela tramite MPI

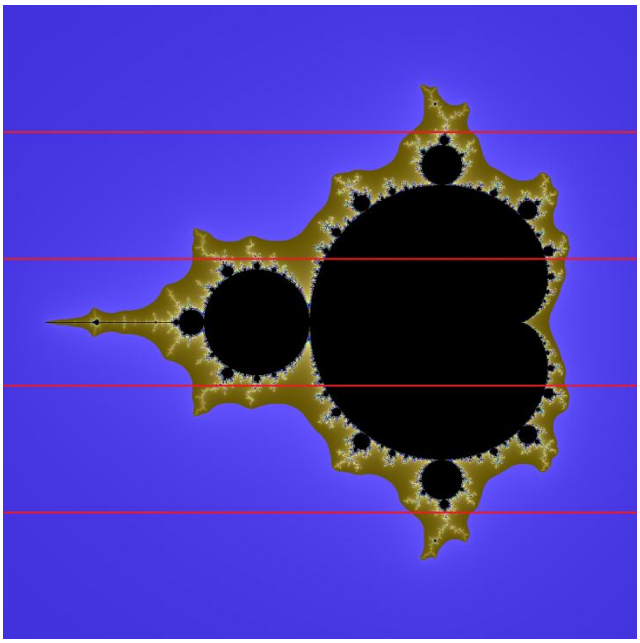
Il problema sopra considerato, risulta essere parallelizzabile. Per far ciò è stato utilizzato il modello di programmazione basato sul Message Passing dove i task si scambiano dati con messaggi di send e receive. A tale scopo, è stato utilizzato DeinoMPI [4], una implementazione del Message Passing Interface per Windows.

Come è possibile facilmente capire dal codice, la parte computazionale più esigente è la parte dell'immagine in nero poiché per ottenere un pixel di colore nero, bisogna effettuare il numero massimo di iterazioni (nel nostro caso, ne sono state previste 1000).

Effettuando un'analisi del problema, si nota facilmente che il calcolo di ogni pixel è indipendente dagli altri. Si può quindi decidere di parallelizzare il problema in 2 modi:

a) Dividere l'immagine in n parti di dimensione (cols, rows:n), dove n è il numero di task utilizzati per la computazione. Vi è un task master che invia informazioni ai worker, controlla l'ubicazione e colleziona i risultati. In questo modo si ha un bilanciamento statico e con processori eterogenei questo può essere un limite. È facile intuire che il task che calcola il maggior numero di pixel neri è il task che impiega più tempo.
b) Dividere l'immagine in un determinato numero di fasce abbastanza grande ed assegnare una fascia ad un task libero. In questo modo si effettua un bilanciamento dinamico del carico introducendo un maggior overhead per la comunicazione.

Il problema sopra esposto è stato implementato nel primo modo, ossia ogni task computata una parte di immagine ed infine il task con rank 0 assembla il tutto.



Nella figura di sopra, viene mostrato un esempio di partizionamento dell'immagine in base alla parte che ogni task ha computato (nell'immagine di sopra sono stati utilizzati 5 task).

```
/*
L'insieme di Mandelbrot è definito come l'insieme dei numeri complessi
c tale per cui
non è divergente  la successione definita da:
    z_n+1 = z_n^2 + c
con
    z_0 = 0
L'insieme è un frattale e, nonostante la semplicità della definizione,
ha una forma non banale.

Utilizzata la libreria EasyMBP per il salvataggio dell'immagine BMP
(per maggiori info visitare http://easybmp.sourceforge.net/)
*/

#include "EasyBMP.h"
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>
#include <mpi.h>

//prototipo della funzione che restituisce il colore del pixel di una
determinata
//coordinata
RGBApixel mandel(int, int);
RGBApixel** mandelRank(int, int);

//variabili globali
int maxiteration=1000;
double cx;                          /* i rispettivi valori di c
(x,y)*/
double cy;
double magnify = 1.0;         /* no magnification     (zoom)*/
int rows=10000;
int cols=10000;
const double EscapeRadius=6.0;

int main(int argc, wchar_t **argv){
      clock_t start,end;

      double tempo;
      start=clock();

      int numtasks,rank;

      MPI_Init ( &argc,  &argv);
      MPI_Comm_rank( MPI_COMM_WORLD, &rank );
      MPI_Comm_size( MPI_COMM_WORLD, &numtasks );

      //Poichè l'oggetto pixel ha 4 campi da 1 byte, quindi il nuovo
      //oggetto contigous avrà 4 byte contigui
      MPI_Datatype pixel;
      MPI_Type_contiguous (4,                   //replication count
            MPI_BYTE,                                 //old datatype
(handle)
            &pixel                                          //new
datatype (handle)
            );

      //sottomettiamo il tipi appena creto per poterlo utilizzare
```

```
      MPI_Type_commit(&pixel);

      //inizializzazione oggetto immagine
      RGBApixel* Picture1D=NULL;
      BMP Bmp;
      if (rank==0){
            //allocazione della picture
            Picture1D=(RGBApixel*)malloc(rows*cols*sizeof(RGBApixel));
      }

      int numRows=rows/numtasks;
      int myStartRow=rank*numRows;
      int myEndRow=(rank+1)*numRows;

      if(rank==numtasks-1){
            myEndRow=rows;
            numRows=numRows+(rows%numtasks);
            //l'ultimo task devo contenere il restante numero di righe
      }
      RGBApixel** subPicture=mandelRank(myStartRow, myEndRow);
      int outsize=numRows*cols*sizeof(RGBApixel);
      RGBApixel* subPicuture1D=(RGBApixel*)malloc(outsize);
      int position=0;

      /* ho la mia sottofigura espressa in una matrice di RGBApixel.
      Prima di fare la gather, quindi, ho bisgno di raggruppare i
      dati in un unico buffer monodimensionale.
      Per fare ciò utilizzo quindi l'MPI_Pack.
      */
      for (int i=0; i<numRows;i++){
            MPI_Pack(&subPicture[i][0],         //input buffer start
                  cols,                               //number of input
data items (1 riga)
                  pixel,                                    //datatype
of each input data item
                  subPicuture1D,                      //output buffer
start
                  outsize,                            //output buffer
size, in bytes
                  &position,                          //current
position in buffer
                  MPI_COMM_WORLD                      //communicator
for packed message
                  );
      }

      MPI_Gather(subPicuture1D,           //indirizzo di partenza del
buffer
            numRows*cols,                       //numero di elemnti da
inviare
            pixel,                                    //tipo di dato da
inviare
            Picture1D,                          //indirizzo di
ricezione
            numRows*cols,                       // numero di elementi
di ogni singolo receive
            pixel,                                    //tipo di dati
nel buffer di ricezione
            0,                                        //rank del
receive
```

```
            MPI_COMM_WORLD                        //communicator
            );

      //ora bisogna che il rank 0 deve spacchettare tutto l'array
monodimensionale
      //per fare ciò, grazie al puntatore position, prendo dall'array
una porzione
      //di dati pari ad una riga. questo verrà inserito nell'oggetto
Bmp
      if (rank == 0){
            Bmp.SetSize( cols, rows);
            Bmp.SetBitDepth(24);
            position=0;
            for (int y=0;y<rows;y++){
                  RGBApixel *riga=(RGBApixel*)
malloc(cols*sizeof(RGBApixel));
                  MPI_Unpack(Picture1D,         //inbuf input buffer
start (choice)
                        cols*rows,              //size of input buffer
                        &position,              //position current
position in bytes
                        riga,                   //output buffer start
(choice)
                        cols,                   //number of items to be
unpacked (integer)
                        pixel,                        //datatype of
each output data item (handle)
                        MPI_COMM_WORLD          //communicator for
packed message (handle)
                        );
                  for (int x=0;x<cols;x++){
                        Bmp.SetPixel(x,y,riga[x]);
                  }
                  free(riga);
            }
            Bmp.WriteToFile( "mandelbrot-parallelo.bmp" );
            end=clock();
            tempo=((double)(end-start))/CLOCKS_PER_SEC;
            printf("%f",tempo);
      }
      free(Picture1D);
      free(subPicture);
      free(subPicuture1D);
      MPI_Type_free(&pixel);
      MPI_Finalize();
      exit(0);
}

RGBApixel mandel(int xpt, int ypt){
      RGBApixel pixel;
      double xtemp=0;
      double x=0;
      double y=0;

      int iteration=1;
      while ((x*x+y*y<=(EscapeRadius*EscapeRadius)) &&
iteration<maxiteration){
            xtemp=x*x-y*y + cx;
            y=2.0*x*y+cy;
            x=xtemp;
```

```
            iteration++;
      }

      if (iteration == maxiteration){
            //color black
            pixel.Red = 0;
            pixel.Green = 0;
            pixel.Blue = 0;
      }
      else{
            //color = iteration
            double  mu = iteration+1-((log(log(x*x+y*y)))/log(2.0));
            mu=mu*EscapeRadius;
            pixel.Red=53+mu;
            pixel.Green=37+mu;
            pixel.Blue=208+mu;
      }
      return pixel;
}

RGBApixel** mandelRank(int myStartRow, int myEndRow){
      //allocazione della subpicture
      RGBApixel** subPicture=(RGBApixel**)malloc((myEndRow-
myStartRow)*sizeof(RGBApixel*));
      for (int y=0;y<(myEndRow-myStartRow); y++){
            subPicture[y]=(RGBApixel*)malloc(cols*sizeof(RGBApixel));
      }

      for ( int y0=myStartRow;y0<myEndRow;y0++){
            for (int x0=0;x0<cols;x0++){
                  cx=(((float)x0)/((float)cols)-0.5)/magnify*3.0-0.7;
                  cy=(((float)y0)/((float)rows)-0.5)/magnify*3.0;

                  RGBApixel px=mandel(x0,y0);
                  subPicture[y0-myStartRow][x0]=px;
            }
      }
      return subPicture;
}
```

Come per l'algoritmo sequenziale, anche questa versione parallela utilizza la libreria EasyBMP per la memorizzazione dell'immagine. Come possiamo vedere nel main, ogni task stabilisce la porzione di dati da calcolare in base al numero di rank che ha. Effettuata la computazione, si ottiene una sottomatrice di pixel. Questi dati vengono impacchettati in un buffer attraverso la funzione MPI_Pack specificando come tipo di dato un oggetto MPI_DataType pixel. Questo oggetto, dichiarato nelle prime linee di codice del main, rappresenta l'oggetto RGBAPixel ed è necessario la sua dichiarazione per lo scambio di dati tra i task.

Ogni task infine chiama la MPI_Gather, una funzione che collezione tutti i dati dei vari task in un unico task. Nel nostro caso, il task con rank 0 colleziona i dati in un buffer

monodimensionale, organizzati in ordine di rank. Dal buffer, attraverso la funzione MPI_Unpack vengono estrapolati i dati riga per riga e si provvede a memorizzarli nell'oggetto BMP bmp.

La figura ottenuta è analoga a quella presentata nel paragrafo precedente.

I test sono stati effettuati su un computer con:

- Processore Intel Core 2 Duo T 7300 a 2.00 Ghz;
- 4 GB RAM DDR2;
- Sistema Operativo: Windows 7 64 Bit
- DeinoMPI 2.0.1

Nella tabella e nella figura successiva, sarà mostrato il tempo di esecuzione della versione parallela incrementando il numero di task.

Numero di processi	Tempo di esecuzione (sec)
1	318.835
2	220.311
3	311.518
4	243.716
5	268.892

Bibliografia

[1] Frattali, sito web http://www.miorelli.net/frattali/matematica.html, ultimo accesso 15/05/2010

[2] Insieme di Mandelbrot, http://it.wikipedia.org/wiki/Insieme_di_Mandelbrot, ultimo accesso 15/05/2010

[3] EasyBMP, http://easybmp.sourceforge.net/, ultimo accesso 15/05/2010

[4] DeinoMPI, http://mpi.deino.net/, ultimo accesso 15/05/2010