Using Matlab and MIT's MatlabMPI Toolbox for the creation of The Multi-Dimensional Fractals

J. Nejedlý, J. Daněk

Department of Mathematics, University of West Bohemia

Abstract

The algorithm for generating a representation of the 2D fractal is deceptively simple. A repeating calculation is performed for each x,y point in the plot area and based on the behaviour of that calculation, a colour is chosen for pixel representing selected point.

The same way is used for generating multi-dimensional fractals, in our case 4D fractals. To generate famous Mandelbrot set we use a simple iteration process represented by the equation $z = z^2 + constant$, where $z \in \mathbf{C}$. For 4D fractals we will use $z = z^x + constant$, where $z, x \in \mathbf{C}$.

The generating process of the fractal consist of many iterations, thus we need a great deal of the computer time. However, these iterations do not depend on each other, so we can generate fractal representation by more than one processor very easily. This is the right point for utilizing multiprocessor computers by MPI library, especialy with MatlabMPI toolbox developed by Dr. J. Kepner in Lincoln Laboratory, MIT.

1 Matlab and Parallel Computing

Matlab is simple but powerful language for implementing numerical computations and we use it for algorithm development, simulation and testing. Many of these computations could benefit from faster execution on a parallel computer. However, standard usage of the parallel computers assumes good knowledge of C or Fortran programming language and complicated usage of libraries for many standard algebra or mathematical analysis operations. If we could combine the simplicity of Matlab language with the ability to run on multiple processors, we could obtain a very powerful tool for solving large-scale problems.

The Message Passing Interface (MPI) is the standard for implementing programs on multiple processors, independently on parallel computer architecture. MPI defines C and Fortran language functions for doing interprocess communication in a parallel program.

MatlabMPI[1] is a set of Matlab scripts that implement a subset of MPI and allow any Matlab program to be run on a parallel computer. This package can be freely downloaded from *http://www.ll.mit.edu/MatlabMPI*.

The main benefit is simple use but the greatest disadvantage is the fact that MatlabMPI uses disk I/O for simulating message passing through interprocessor interface. This problem can be eliminated by the use of SMP computers with working directory located on the ramdisk. As modern computer processors are based on more than one core so even in laptop computer we have small parallel computer and if we use two-processor computers we have four processors to utilize with shared memory. We have tested MatlabMPI under Linux with great success.

We will try to subscribe use of the MatlabMPI with very simple problem - to generate fractals.

2 2D Fractals

The algorithm[2] for generating a representation of the 2D fractal is simple. A repeating calculation - iteration - is performed for each x, y point in the plot area and based on the behavior of that calculation, a color is chosen for pixel representing selected point.

To generate famous Mandelbrot set we use a simple iteration process represented by the equation

$$z_{n+1} = z_n^2 + c,$$

where $z, c \in \mathbf{C}, z_0 = 0, Re(c) = x, Im(c) = y$. The stop condition is $|z_n - z_{n-1}| < \varepsilon$ and color of the point x, y represents number of iterations needed to achieve convergency. If $|z_n| > 2$ then the process diverges. See Fig.1. To obtain this image with resolution 401x401 about 1.5 millions of iterations have been done. However, these iterations do not depend on each other, so we can compute them separately - we can use more than one processor.

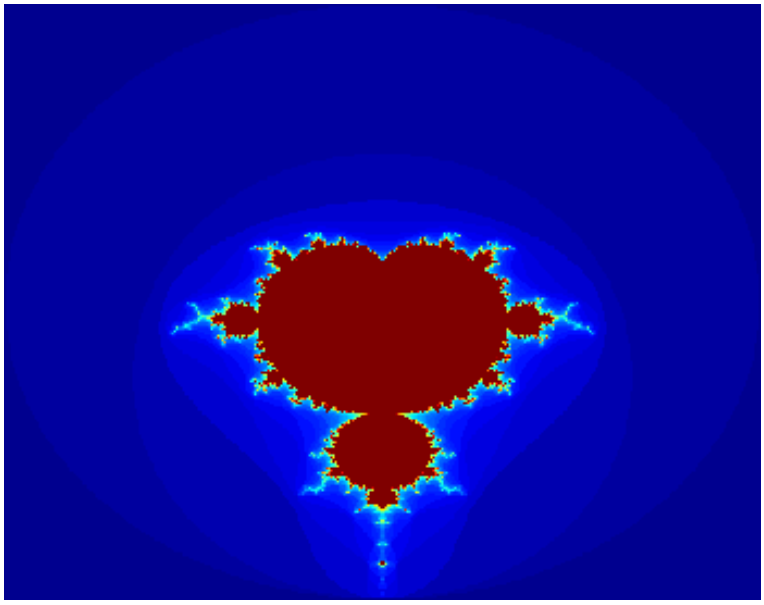

Figure 1: Mandelbrot set computed by 1 processor

This is presented in Fig.2. This fractal was computed by two processors simultaneously, the computational domain was simply divided into two parts. The source code of this example

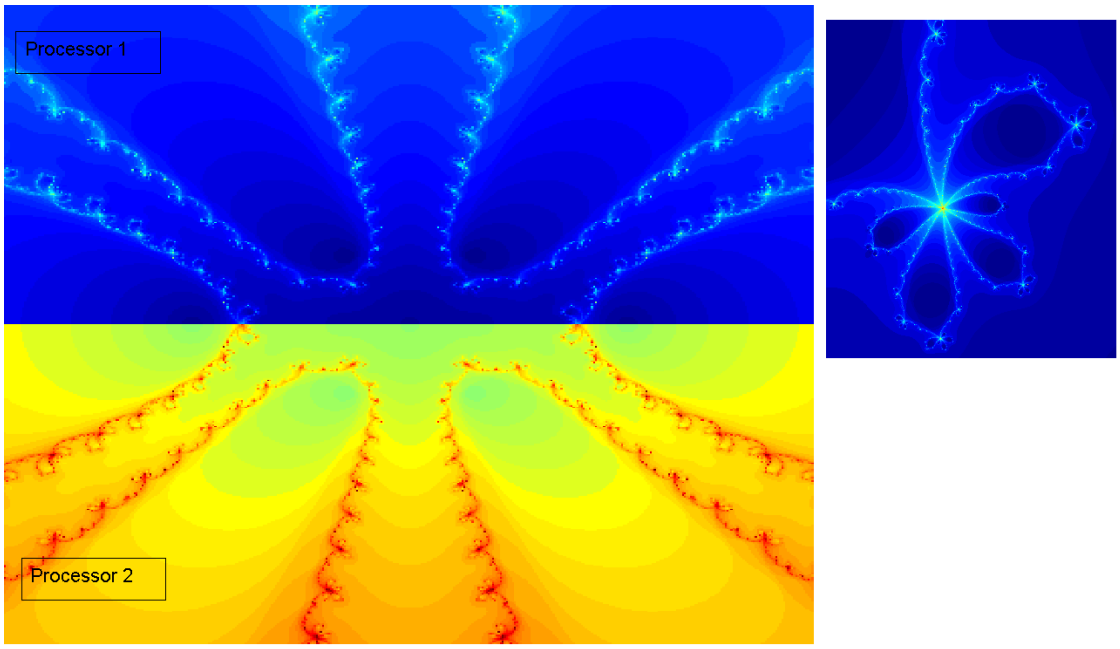

Figure 2: 2D fractal computed by 2 processors; with nice detail

is listed below.

```
% To run, start Matlab and type:
%
%   eval( MPI_Run('fractal',3,{}) );  %one master, two slaves
%                                      master consumes very little CPU time
%
% Or, to run a different machines:
%
%   eval( MPI_Run('fractal',3,{'machine1' 'machine2' 'machine3'}) );
% Initialize MPI.
MPI_Init;
% Create communicator.
comm = MPI_COMM_WORLD;
% Get size and rank.
comm_size = MPI_Comm_size(comm); my_rank = MPI_Comm_rank(comm);

if comm_size<2
    disp(' need more processors');
    break;
end

x=-2:0.01:2;    %whole domain
y=-2:0.01:2;

Z=zeros(length(x),length(y));   %matrix of iterations

%%%%%%%%%%%%%%%%%%%%%
if my_rank=0  %master
%%%%%%%%%%%%%%%%%%%%%

domains=comm_size-1;  %same number of domains as slave processors

%divide vector x to parts
veclen=floor(length(x)/domains);

for i=1:domains-1
   MPI_Send(i,1,comm,(i-1)*veclen+1:i*veclen)  %send "pieces of work"
end;
   MPI_Send(domains,1,comm,(domains-1)*veclen+1:length(x));  %last piece

for i=1:domains
   Z = Z + MPI_Recv( i, 100, comm );  %Collect results
end;

image(x,y,Z);
shading interp;
colormap(jet);

%%%%%%%%%%%%%
end   %master
%%%%%%%%%%%%%
```

```
%%%%%%%%%%%%%%%%%%%%
if my_rank>0 %slave
%%%%%%%%%%%%%%%%%%%%

vektor=MPI_Recv(0,1,comm);  %what to do

koreny=roots([10 0 10 0 -1 0 1 0]);

for l=vektor;
  for k=1:length(y);
    konec=1;
    z=x(l)+i*y(k);
    while konec>0
    zn=z-(10*z^7 +10*z^5 - z^3 + z)/ ...
                          (70*z^6+50*z^4+3*z^2+1);
    %fractal formula, Newton's method
    konec=konec+1;
    if min(abs(z-koreny))<.05;
      konec=-konec;
    end;
    if konec>50
      konec=-konec;
    end;
    z=zn;
    end;
    Z(l,k)=-konec;
  end;
end;

MPI_Send( 0, 100, comm, Z );  %send my results

exit;
%%%%%%%%%%%%%%%
end %slave
%%%%%%%%%%%%%%%
```

If we use more than one processor, we can observe that some of the processors have done their job quickly but some of them are still computing, especially when we strictly divide the computational domain into the same number of parts as the number of processors used.

To avoid this situation we need to apply some "load balancing" - the processors compute approximately the same amount of calculations. An easy way to do that is to divide the computational domain into a greater number of subdomains (much greater than the number of processors) and declare one process as "master". This process assigns subdomains to processors and collects their outputs. This proces is called "Master" and the processes which provide calculations are called "Slaves" - so called Master-Slave model.

3 Multi-dimensional Fractals

A similar procedure is used for generating multi-dimensional fractals. For generating 4D fractals (Fig.4) we will use

$$z_{n+1} = z_n^x + c,$$

where $z, x, c \in \mathbf{C}$. We can choose many different start conditions. If we fix a real or imaginary part of the exponent, we obtain 3D (Fig.3) fractals. For generating these structures we need a great deal of computational time, so it is very useful if we can utilize more processors.

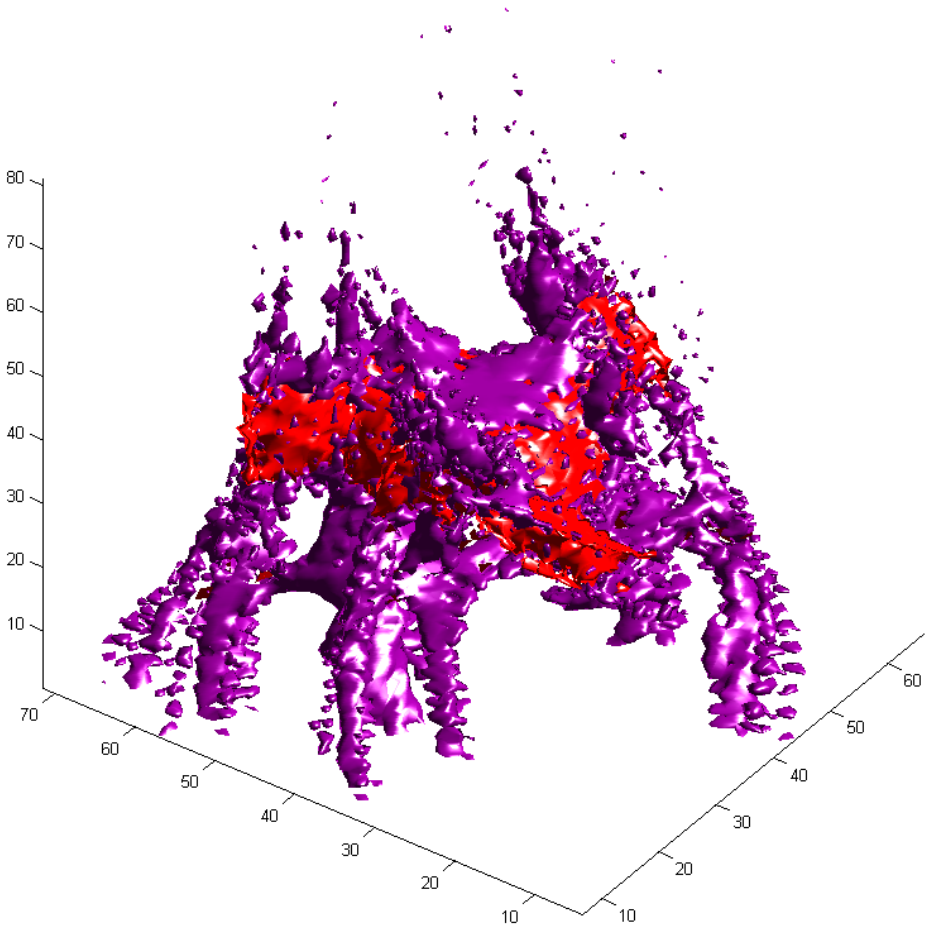

Figure 3: 3D fractal, the red parts represent points which need 5 iterations to converge and the violet parts represent points which need 10 iterations.

To render presended pictures the Matlab's functions *patch* and *isosurface* were used.

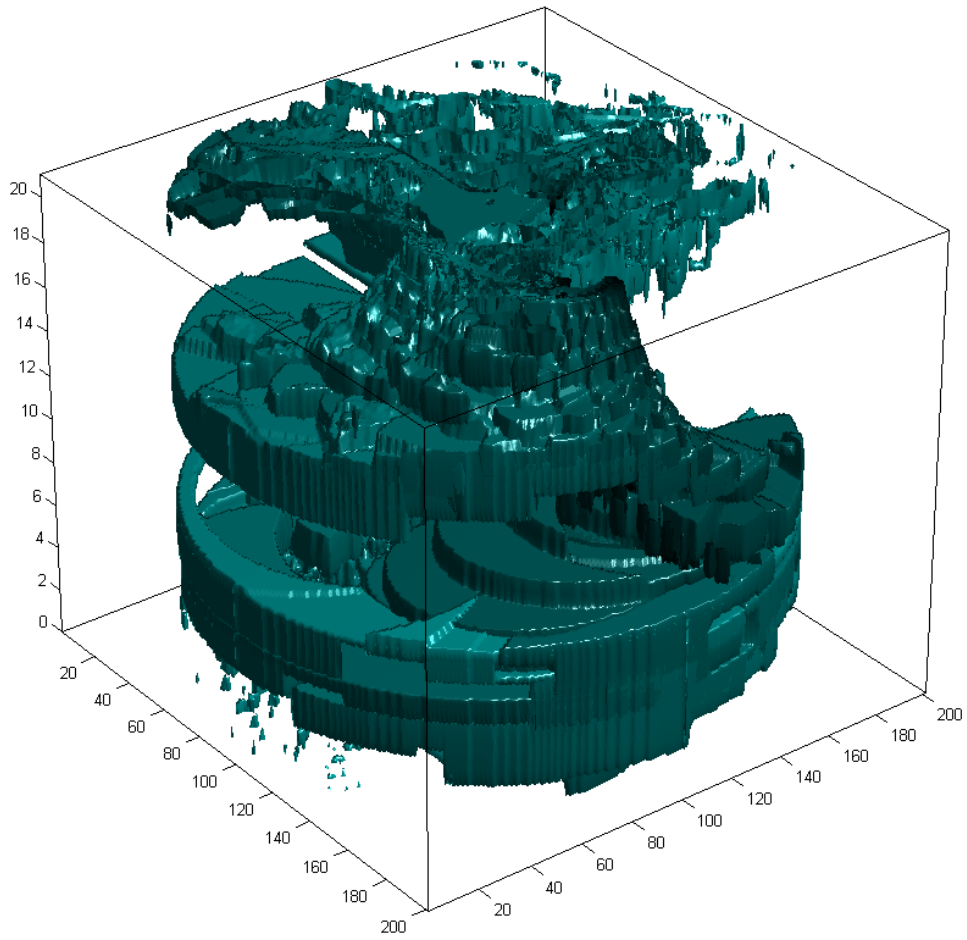

Figure 4: 4D fractal, the structure shown represents points which need 5 iterations to converge; changing color represents an imaginary part of the exponent in the computation formula.

References

[1] J. Kepner. Parallel programming with MatlabMPI. http://www.ll.mit.edu/MatlabMPI/.

[2] P. Tišnovský. Fraktály v počítačové grafice. http://www.root.cz/clanky/fraktaly-v-pocitacove-grafice-i/.

Jan Nejedlý
Department of Mathematics, Faculty of Applied Sciences, University of West Bohemia, Univerzitni 22, 306 14 Pilsen, e-mail: lucifer@kma.zcu.cz

Josef Daněk
Department of Mathematics, Faculty of Applied Sciences, University of West Bohemia, Univerzitni 22, 306 14 Pilsen, e-mail: danek@kma.zcu.cz