VILNIAUS UNIVERSITETAS

MATEMATIKOS IR INFORMATIKOS FAKULTETAS

PROGRAMŲ SISTEMŲ KATEDRA

Concurrent Calculation of Mandelbrot Set.
Using Master/Slave model and MPI

IV laboratory assignment

Author:

Kazimieras Senvaitis

Supervisor:

prof. Rimantas Vaicekauskas

Vilnius, 2018

Introduction

The Mandelbrot set is made up of points plotted on a complex plane to form a fractal: a shape or form in which each part is actually a miniature copy of the whole.

The Mandelbrot set has become popular outside mathematics both for its aesthetic appeal and as an example of a complex structure arising from the application of simple rules. It is one of the best-known examples of mathematical visualization and mathematical beauty (See Figure 1).

Images of the Mandelbrot set exhibit an elaborate and infinitely complicated boundary that reveals progressively ever-finer recursive detail at increasing magnifications.

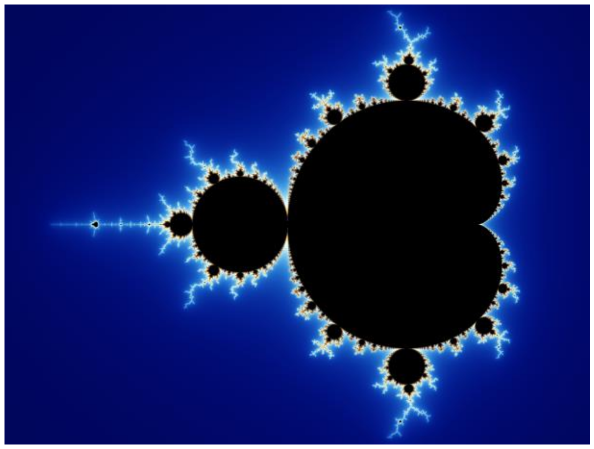

Figure 1. Visualized Mandelbrot set (in black color)

This paper explains how to implement Mandelbrot Set drawing algorithm using C++ programming language and OpenMPI library.

1. Mandelbrot Set

Mathematically, the Mandelbrot set is defined on the plane of complex numbers by picking a starting point c and iterating the formula $z_{k+1} = z_k^2 + c$. The iteration gives you a sequence of numbers that either stays bounded or spirals out of control further and further from the starting point. The complex number c belongs to the Mandelbrot set if the sequence stays within a radius of 2 from the origin.

2. Concurrent algorithm

Calculation of Mandelbrot set is considered to be perfectly parallel computation as it can be divided into a number of completely independent parts (See Figure 2).

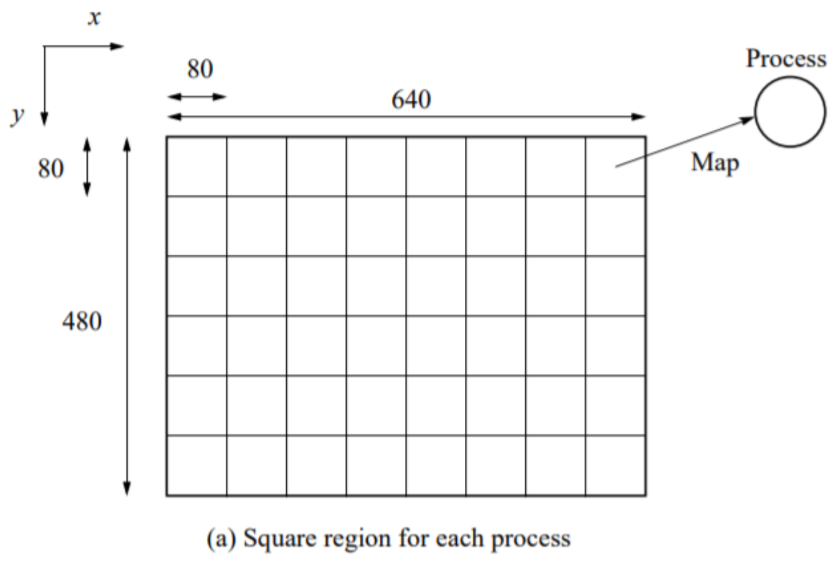

Figure 2. Splitting into completely independent parts

However, calculating the Mandelbrot set is a task of infinite complexity. From the beginning it is not possible to efficiently assign tasks to threads as tasks are of unequal difficulty. Idling of threads should be reduced as it reduces efficiency. Efficiency is calculated as follows:

$$e = \frac{\left(\frac{t_0 + t_1 + \ldots + t_{(p-1)}}{p}\right)}{t_{total}}$$

For example, one image region needs calculations of 10 seconds for each C before it exits the Mandelbrot set, and other needs calculations of 100 seconds before exiting the set. So, thread calculating first region would be idling while second thread still working. Efficiency in this case would be calculated:

$$e = \frac{\left(\frac{10 + 100}{2}\right)}{100} = 55\%$$

For master/slave model, one solution for independent image part assignment for a thread, is to split image into tasks and send tasks to slaves, see UML state diagram in Figure 3.

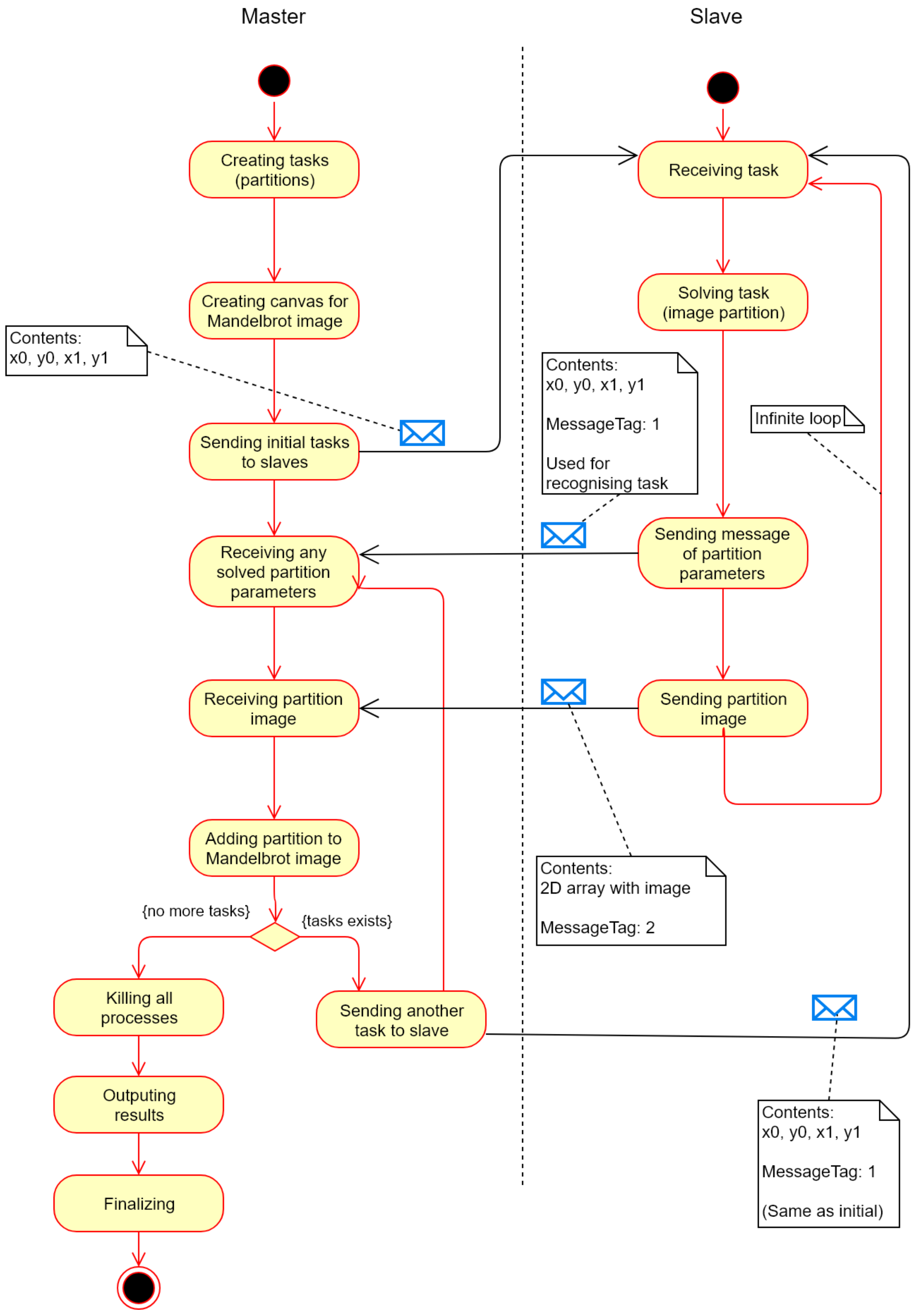
Master
Slave
Creating tasks (partitions)
Creating canvas for Mandelbrot image
Contents: x0, y0, x1, y1
Sending initial tasks to slaves
Receiving any solved partition parameters
Receiving partition image
Adding partition to Mandelbrot image
{no more tasks}
{tasks exists}
Killing all processes
Sending another task to slave
Outputing results
Finalizing
Receiving task
Solving task (image partition)
Contents: x0, y0, x1, y1 MessageTag: 1 Used for recognising task
Infinite loop
Sending message of partition parameters
Sending partition image
Contents: 2D array with image MessageTag: 2
Contents: x0, y0, x1, y1 MessageTag: 1 (Same as initial)

Figure 3. Master/slave algorithm for concurrently calculating Mandelbrot Set

3. Computer program

Computer program that calculates and draws Mandelbrot Set is written using C++ programming language and uses OpenMPI implementation of Message Passing Interface (MPI) library. For drawing the fractal, it uses simple console output. Additionally, it uses complex number library for calculating Mandelbrot set values.

Table below describes input parameters for the OpenMPI program. The latter table describes input parameters for the Mandelbrot Set calculation program:

No.	**Parameter**	**Allowed values**	**Description**
1.	np	2, 3, …, N	Determines number of threads in the C++ program. Works only when runing with OpenMPI runner command: mpirun.

No.	**Parameter**	**Allowed values**	**Description**
1.	debug	0, 1	Debug mode enables verbose in the program: logs debugging information.
2.	GUI	0, 1	Draws Mandelbrot set in terminal.
3.	Thread Pool Size	1, 2, …, N	Number of threads in thread pool.
4.	Max Iterations	1, 2, …, N	Maximum number of iterations in Mandelbrot Set algorithm. In other words, depth of calculation.
5.	Image Width	1, 2, …, N	Image width in pixels
6.	Image Height	1, 2, …, N	Image height in pixels
7.	Image Width Step	1, 2, …, N	Step by which image width is partitioned to allow creation of smaller tasks. In pixels.
8.	Image Height Step	1, 2, …, N	Step by which image height is partitioned to allow creation of smaller tasks. In pixels.

To compile this program run: `mpiCC mpi-mandelbrot.cpp`

It is important to note, that mpicc is for C programs, whereas mpiCC for C++ programs.

4. Program run example

For example, command `mpirun -np 9 a.out 0 0 1000 500 500 100 100` run from terminal would produce following a picture (See Figure 4) and following console output:

Notes about MPI_ABORT should be ignored. Termination of slave processes was simplified, this inducing these notes.

```
Available CPU cores: 8, Available workers (threads): 8
Program starting:
debug: 0; gui: 0; maxIter: 1000; imgWidth: 500; imgHeight: 500; imgWidthStep: 100;
imgHeightStep: 100;
Time: 1.634948
--------------------------------------------------------------------------
MPI_ABORT was invoked on rank 0 in communicator MPI_COMM_WORLD
with errorcode 1.

NOTE: invoking MPI_ABORT causes Open MPI to kill all MPI processes.
You may or may not see output from other processes, depending on
exactly when Open MPI kills them.
--------------------------------------------------------------------------
```

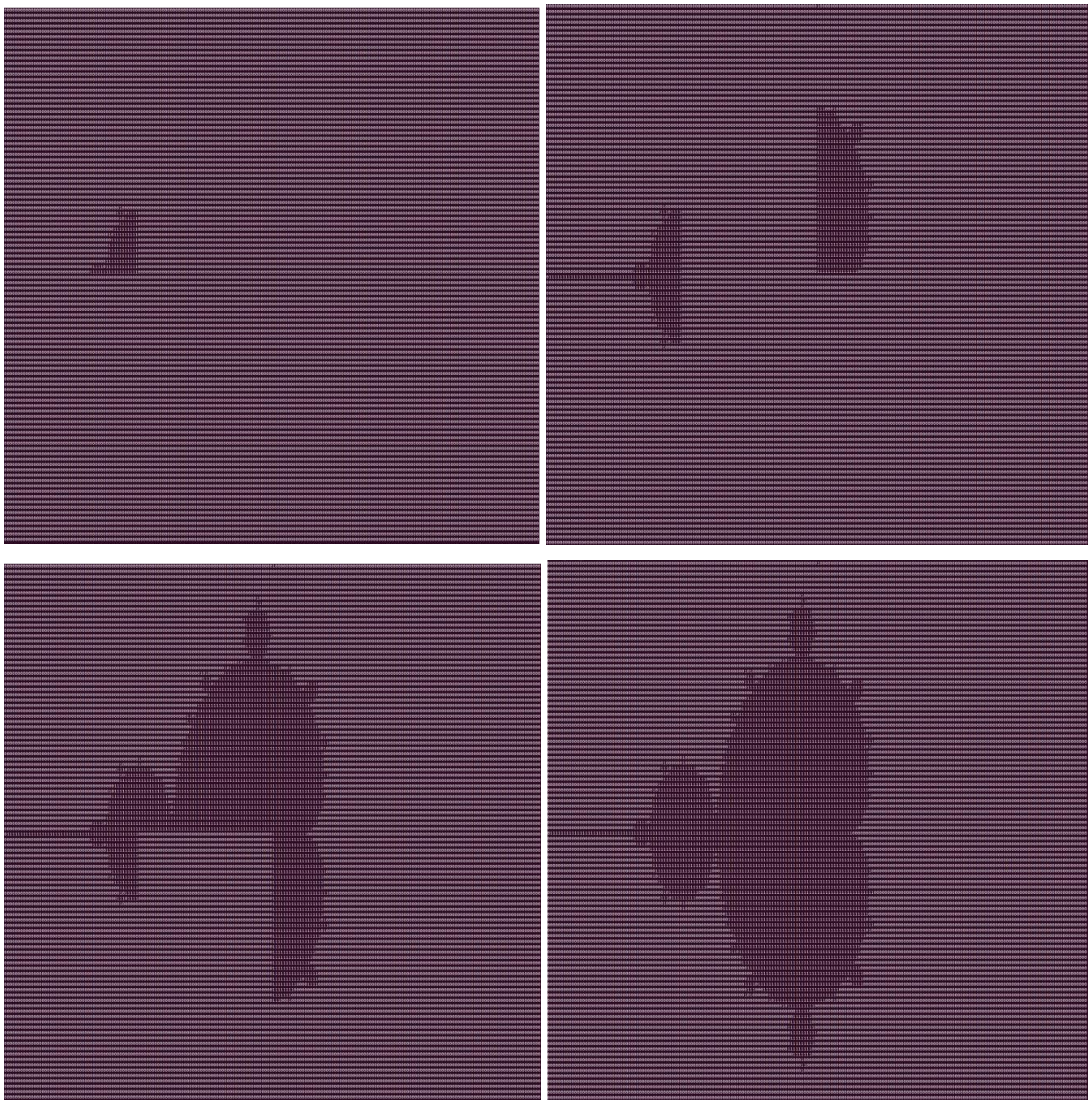
Figure 4. Picture being drawn by 8 workers