PAR Laboratory Assignment
Lab 3: Analysis of parallel strategies: the computation of the Mandelbrot set

Mario Acosta, Eduard Ayguadé, Rosa M. Badia, Jesus Labarta (Q1),
Josep Ramon Herrero, Daniel Jiménez-González, Pedro Martínez-Ferrer,
Jordi Tubella and Gladys Utrera

Spring 2023-24

UNIVERSITAT POLITÈCNICA
DE CATALUNYA
BARCELONATECH

Index

Index **1**

1 Iterative task decomposition analysis **6**
1.1 Sequential execution 6
1.2 Analysis of strategies 6

2 Recursive task decomposition analysis **8**
2.1 Sequential execution 8
2.2 Analysis of strategies 9

Note:

- This laboratory assignment will be done in two sessions (2 hours each).
- All files necessary to do this laboratory assignment are available in a compressed tar file available from the following location: `/scratch/nas/1/par0/sessions/lab3.tar.gz` in `boada.ac.upc.edu`. Uncompress it with this command line:
  `"tar -zxvf ~par0/sessions/lab3.tar.gz"`.

Laboratory Deliverable Information

IMPORTANT NOTE: In this laboratory you will analyze different parallel strategies. Your objective is to determine the dependences between the different tasks and analyse the potential concurrency problems due to data sharing, synchronization and possible load unbalance. Table 1 explains what should be included in the report for each parallel strategy you try. Only PDF format for this document will be accepted. A deliverable for the assignment will be opened in *Atenea* and the appropriate dates for the delivery will be set. You also have to deliver the complete C source codes for Tareador instrumentation strategies that you have done. DON'T include code in the document except for the fragment modified with respect to the original one: just provide a code except. You will have to **deliver TWO files**, one with the document in PDF format and one compressed file (`tgz`, `.gz` or `.zip`) with the requested C source codes.

Section	**Description**
Code	Not necessary for the original strategy we provide you. Add the source code to the zip. Indicate the name of the source code in the pdf.
Tareador TDG:	Include the TDG image.
Dependence Analysis:	Explain the dependences found. Reason if they force the order of the tasks or if you may be able to use data sharing synchronizations instead
Tareador TDG (without dependences):	Include a new TDG image obtained when dependences that can be removed using data sharing synchronizations are deactivated using tareador (`tareador_disable_object(&name_var)`).
T_∞ **Analysis:**	For each parallel strategy, with its dependences due to data sharing disabled, perform simulations to describe if there may be load unbalance, and compute T_1 and T_∞.

Table 1: Analysis to be included in the pdf report

Comparison: Table 2 should be included after the description of all parallel strategies. Based on your parallel strategies results, reason which will be the best parallel strategy for iterative and for recursive implementations.

Version	T_1	T_∞	**Parallelism**
Iterative: original strategy (no parameters) Iterative: original strategy (-d) Iterative: original strategy (-h)			
Iterative: Finer grain			
Iterative: Column of tiles			
Recursive: Leaf			
Recursive: Tree			

Table 2: Summary of the parallelism performance of each of the versions

As you know, this course contributes to the **transversal competence "Tercera llengua"**. Deliver your material in English if you want this competence to be evaluated. Please refer to the "Rubrics for the third language competence evaluation" document to know the *R*ubric that will be used.

Introduction: The Mandelbrot set

In this laboratory assignment you are going to explore differents parallel strategies of **iterative and recursive task decompositions**. The program that will be used is the computation of the *Mandelbrot set*, a particular set of points, in the complex domain, whose boundary generates a distinctive and easily recognisable two-dimensional fractal shape (Figure 1).

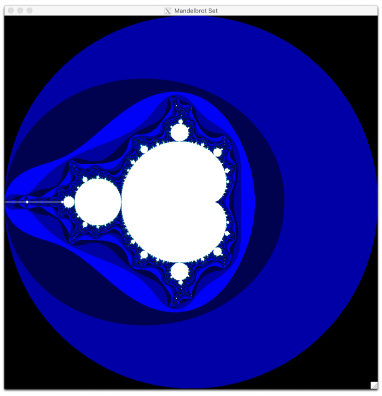

Figure 1: Fractal shape

For each point c in a delimited two-dimensional space, the complex quadratic polynomial recurrence $z_{n+1} = z_n^2 + c$ is iteratively applied n to determine if it belongs or not to the Mandelbrot set. The point is part of the Mandelbrot set if, when starting with $z_0 = 0$ and applying the iteration repeatedly, the absolute value of z_n never exceeds a certain number however large n gets.

A plot of the Mandelbrot set is created by colouring each point c in the complex plane with the number of steps max for which $|z_{max}| \geq 2$ (or simply $|z_{max}|^2 \geq 2 * 2$ to avoid the computation of the square root in the modulus of a complex number). In order to make the problem doable, the maximum number of steps is also limited: if that number of steps is reached, then the point c is said to belong to the Mandelbrot set.

If you want to know more about the Mandelbrot set, we recommend that you take a look at the following Wikipedia page:

`http://en.wikipedia.org/wiki/Mandelbrot_set`[1]

In the *Computer drawings* section of that page you will find different ways to render the Mandelbrot set. In particular, we will using the idea of Mariani's algorithm[2] so that we can check the border of rectangles (tiles) to avoid the computation of full tiles since we know all of them have *max* steps. Figure 2 shows a tiled version of the mandelbrot picture. Those tiles that fall in the white area can be computed only copying the white value.

Laboratory files

You will find the following sequential codes without and with *Tareador* instrumentation:

[1]Website last visited on August 31, 2022

[2]Dewdney, A. K. (1989). "Computer Recreations, February 1989; A tour of the Mandelbrot set aboard the Mandelbus". Scientific American. p. 111. JSTOR 24987149

- mandel-seq-iter.c : iterative sequential code without instrumentation
- mandel-seq-rec.c : recursive sequential code without instrumentation
- mandel-seq-iter-tar.c : iterative sequential code with instrumentation
- mandel-seq-rec-tar.c : recursive sequential code with instrumentation

The parameters of the programs can be seen running `mandel -help`. For instance `mandel-seq-rec -help` shows:

```
Usage: ./mandel-seq-iter [-o -h -d -i maxiter -c x0 y0 -s size]
       -o to write computed image (mandel_image.jpg) and histogram (mandel_histogram.out if -h indicated) \
                 to disk (default no file generated)
       -h to produce histogram of values in computed image (default no histogram)
       -d to display computed image (default no display)
       -i to specify maximum number of iterations at each point (default 100)
       -c to specify the center x0+iy0 of the square to compute (default origin)
       -s to specify the size of the square to compute (default 2, i.e. size 4 by 4)
```

Warning: `-o` option makes the program to write the image and histogram to disk, **always with the same name**. Rename them if you want to keep them to check the results of your future parallel programs. Those both files are in binary format and you will need to use **cmp or diff** commands to compare results.

Compilation

Look at the Makefile to see how to compile each of the files.

Execution

We have provided you with the tareador scripts to analyse your paralell strategies.

Analysis

We suggest you to review the `lab1` sections to remember the functionality of tareador and how to use it to generate and analyze your strategies. Remember to look for how to disable the analysis of variables to analyse how this affects to the parallelism.

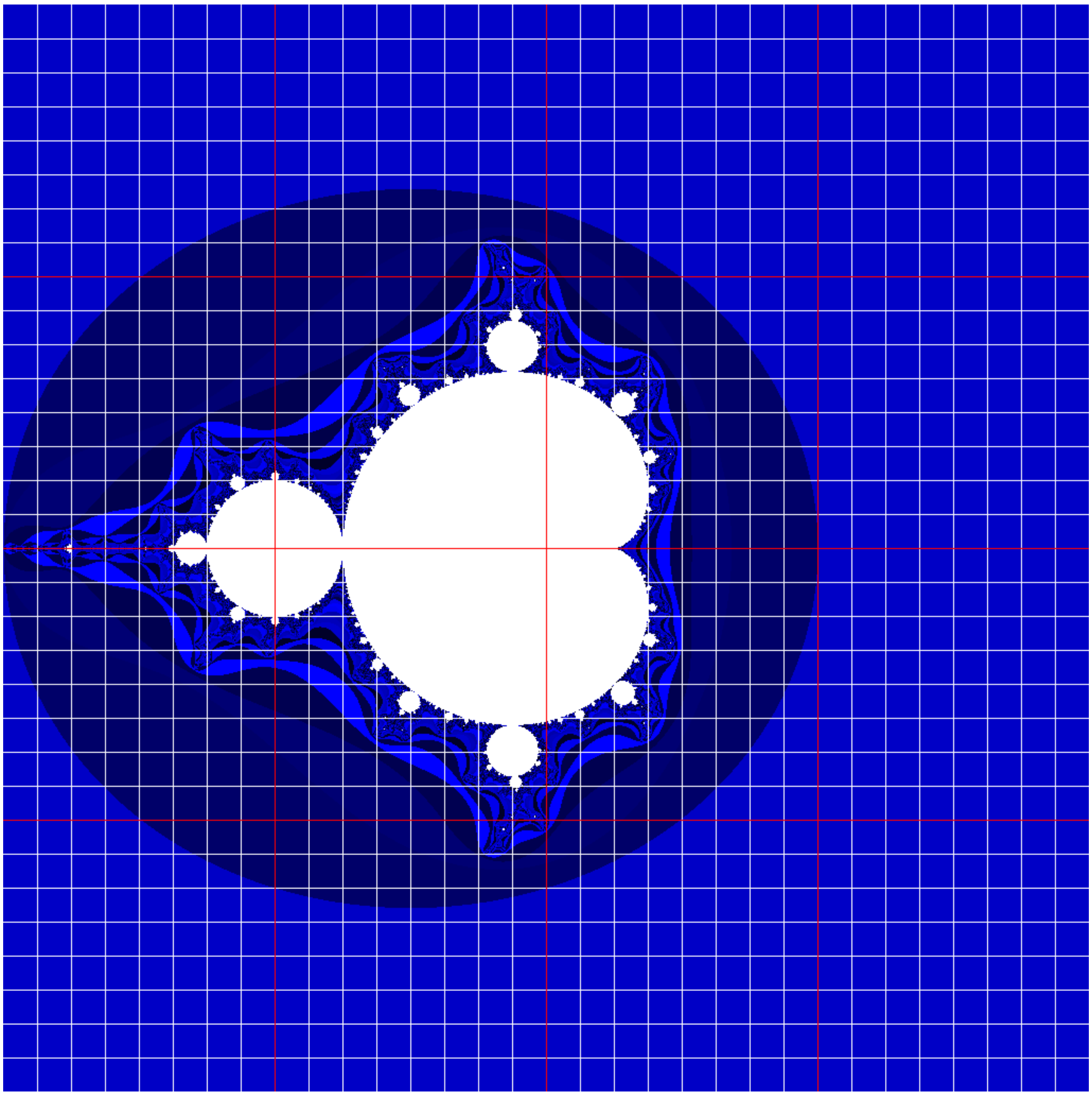
Figure 2: Grid fractal shape

1

Iterative task decomposition analysis

The iterative TILE pseudocode of the Mandelbrot set computation is shown in 1.1.

```
for (int y= 0; y<numROW ; y+=TILE) {
    for (int x= 0; x<numCOL; x+=TILE) {
        equal = // check if horizontal TILE borders have same color
        equal = equal && // check if vertical TILE borders have same color

        if (equal && reach maxiter)
             // fill full TILE with the same color maxiter (if -d, display, if -h, update histogram)
        else // computation of a TILE in the Mandelbrot set (if -d, display, if -h, update histogram)
} }
```

Figure 1.1: Iterative Tile version of the sequential code to compute the Mandelbrot set.

1.1 Sequential execution

Run the sequential code to see what it produces:

- Run mandelbrot to only measure its execution time:

```
sbatch submit-seq.sh ./mandel-seq-iter -i 10000
```

- Run to generate histogram (`mandel_histogram.out`) and image (`mandel_image.jpg`) output:

```
sbatch submit-seq.sh ./mandel-seq-iter -h -o -i 10000
```

- Run it interactively with display:

```
./mandel-seq-iter -d -i 10000
```

- Run tareador analysis. See below.

1.2 Analysis of strategies

Code `mandel-seq-iter-tar` has been already instrumented with *Tareador* to implement a straighforward parallel strategy: one iteration of loop `x` is a task. We ask you to analyze it and create and analyze new parallel strategies:

1. Original parallel strategy. You need to perform three analyses:
    - Run with no parameters.

```
./run-tareador.sh mandel-seq-iter-tar
```

- Run with only "-d" to display the mandelbrot picture.

  ```
  ./run-tareador.sh mandel-seq-iter-tar -d
  ```

  Note that you have to close (or press any key to close) the window displaying the mandelbrot set in order to allow tareador to finish its work.

- Run with only "-h" to keep the histogram of colors.

  ```
  ./run-tareador.sh mandel-seq-iter-tar -h
  ```

The analysis done in this part should be useful for the rest of strategies and future parallelizations. Therefore, we don't ask you to run all them with `-d` and `-h` but you should consider the obtained conclusions here.

2. Finer grain parallel strategy. Remove *Tareador* instrumentation of the original strategy and add the following new instrumentation: in this case check horizontal borders is a task, check vertical borders is a task and the entire if-else conditional is another task and includes two other tasks: each `py` iteration of fill all with the same value is a task and each `py` iteration of computation of a tile is a task (the comments within the code indicate each part).

3. Column of tiles parallel strategy. A task is the computation of a full column of tiles. It is better if you interchange the for x and for y loops in the code before doing so. You can keep the tasks of previous parallel strategy.

2

Recursive task decomposition analysis

The recursive pseudocode of the Mandelbrot set computation is shown in 2.1.

```
recursive_call(matrix)
      equal = // check if horizontal borders of matrix have same color
      equal = equal && // check if vertical matrix borders have same color
      if (equal && reach maxiter)
         // fill full matrix with the same color maxiter (if -d, display, if -h, update histogram)
      else
         if (matrix size < TILE)
            // computation of matrix in the Mandelbrot set (if -d, display, if -h, update histogram)
         else
             recursive_call(block matrix left-top)
             recursive_call(block matrix right-top)
             recursive_call(block matrix left-bottom)
             recursive_call(block matrix right-bottom)
```

Figure 2.1: Tile version of the recursive sequential code to compute the Mandelbrot set.

2.1 Sequential execution

Run the sequential code to see what it produces:

- Run mandelbrot to only measure its execution time:

```
sbatch submit-seq.sh ./mandel-seq-rec -i 10000
```

- Run to generate histogram (`mandel_histogram.out`) and image (`mandel_image.jpg`) output:

```
sbatch submit-seq.sh ./mandel-seq-rec -h -o -i 10000
```

- Run it interactively with display:

```
./mandel-seq-rec -d -i 10000
```

- Run tareador.

```
./run-tareador.sh mandel-seq-rec-tar
./run-tareador.sh mandel-seq-rec-tar -d
./run-tareador.sh mandel-seq-rec-tar -h
```

2.2 Analysis of strategies

Code `mandel-seq-rec-tar` has been already instrumented. In this instrumentation a task is `mandel_tiled_rec` function. We ask you to instrument the code `mandel-seq-rec-tar` with *Tareador* to evaluate and analyze a leaf and a tree parallel strategies. First, remove the original instrumentation.

In particular you should create two indepenent parallel strategies:

- Leaf Recursive Task Decomposition strategy (remember: only base cases are tasks).
- Tree Recursive Task Decomposition strategy. In this case we ask you to create tasks to exploit any possible parallelism in each recursive level.