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, and Gladys Utrera

Spring 2024-25

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Departament d'Arquitectura de Computadors

Index

1

Laboratory Deliverable Information

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 these command lines in your home directory:

```
cd
tar -zxvf ~par0/sessions/lab3.tar.gz
```

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.

For each strategy you are going to analyse in this laboratory, Table 2.1 explains what should be included in the report, in addition of a summary of results filling Table 2.2. Only PDF format for this document will be accepted. An entry for the submission will be created in *Atenea* and the proper submission deadlines will be set. Additionaly, you are required to submit complete C source codes with the Tareador instrumentation that you have to develop. Please refrain from including the entire code in the document, except for fragments of codes that you consider necessary to explain your work. 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.

Comparison: Table 2.2 should be included after the description of all parallel strategies. Run Arguments in the table refers to the optional arguments to be passed to `run-tareador.sh strategy_code arguments`. Based on your parallel strategies results, reason which will be the best parallel strategy for iterative and for recursive implementations.

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.

2

Report Structure

For each strategy (and argument of execution in case of the `Original` strategy), fill in table 2.1.

Section	**Description**
Task Decomposition	Name of the parallel strategy
Arguments	Arguments used in the execution
Code	Add the source code to the zip. Indicate the name of the source code in the pdf. Please refrain from including the entire code in the document, except for fragments of codes that you consider necessary to explain your work.
Tareador TDG (with dependences) :	Include the TDG image for the strategy (run arguments) with all the dependences.
Dependence Analysis:	Explain the dependences found indicating the name of the variable/s. Reason if they force the order of the tasks or if you may be able to use data sharing synchronizations instead
Tareador TDG (without "data sharing" 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:**	Assuming you have disabled only data sharing dependences, perform simulations to describe if there may be load unbalance, and compute T_1 and T_∞. T_1 and T_∞ can be computed using `View Simulation` of the *Tareador* window with 1 and 128 (maximum number of threads allowed in the *Tareador* window), respectively.

Table 2.1: Analysis to be included in the pdf report

Finally, your report should also include the summary of all the strategies, fill in table 2.2:

<table>
<tr><th>Task Decomposition</th><th>Strategy</th><th>Run Arguments</th><th>T_1</th><th>T_∞</th><th>Parallelism</th><th>Load Unbalance (Yes/No) Why?</th></tr>
<tr><td rowspan="5">Iterative</td><td>Original</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>Original</td><td>-d</td><td></td><td></td><td></td><td></td></tr>
<tr><td>Original</td><td>-h</td><td></td><td></td><td></td><td></td></tr>
<tr><td>Finer grain</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>Column</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td rowspan="2">Recursive</td><td>Leaf</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>Tree</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><th rowspan="2">Best Task Decomposition</th><th>Strategy</th><th colspan="5">Reason why</th></tr>
<tr><td></td><td colspan="5"></td></tr>
</table>

Table 2.2: Summary of the parallelism performance of each of the versions

3

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 3.1).

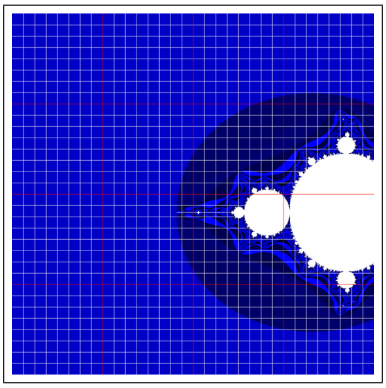

Figure 3.1: Grid Fractal shape

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 the same value. Figure 3.1 shows a tiled version of the mandelbrot picture. Those tiles that fall in the an area with the same color can be computed only copying the same value.

[1] Website last visited on March 13, 2025

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

4

Experimental Setup

Laboratory files

You will find the following files:

- Makefile
- **Source code**
  - 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 iterative original** instrumentation
  - mandel-seq-rec-tar.c : recursive sequential code **with essential** instrumentation
- **Scripts**
  - submit-seq.sh : script to execute the sequential code. Useful to obtain the expected outputs of Mandelbrot.
  - run-tareador.sh : script to execute the *Tareador* instrumented sequential code.

Mandelbrot parameters

The arguments of the programs can be seen running `mandel -help`. For instance:

```
./mandel-seq-iter -help
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
       -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

IMPORTANT: 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.

5

Iterative task decomposition analysis

The iterative TILE pseudocode of the Mandelbrot set computation is shown in Figure 5.1.

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

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

Figure 5.1: Iterative Tile version of the sequential code to compute the Mandelbrot set.

5.1 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.

5.2 Analysis of the 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 the following parallel strategies:

1. Original parallel strategy. You need to perform three analyses:

   - Run with no arguments.

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

     Use `tareador_disable_object(address of variable)` to disable dependences due to data sharing to exploit parallelism among tasks if possible.

   - 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.

     Use `tareador_disable_object(address of variable)` to disable dependences due to data sharing to exploit parallelism among tasks if possible.

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

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

     Use `tareador_disable_object(address of variable)` to disable dependences due to data sharing to exploit parallelism among tasks if possible.

   The analysis done in this part running with "-d" and "-h" is already useful for the rest of strategies and future parallelizations. Therefore, we don't ask you and you don't need to run the following strategies with `-d` and `-h`. However you should consider the obtained conclusions above.

2. Finer grain parallel strategy. You only need to perform the analysis running the strategy with no arguments. Remove *Tareador* instrumentation of the original strategy and add the following new instrumentation (the comments within the code indicate each part):

   (a) Check vertical borders is a task

   (b) Check horizontal borders is a task

   (c) Entire if-else conditional is another task and this includes two other tasks:

      i. Each `py` iteration of fill all with the same value is a task and

      ii. Each `py` iteration of computation of a tile is a task

   Use `tareador_disable_object(address of variable)` to disable dependences due to data sharing to exploit parallelism among previous tasks if possible.

3. Column of tiles parallel strategy. You only need to perform the analysis running the strategy with no arguments. Assume the original strategy (first strategy) to be your baseline source code. Remove the `mandelbrot` task from the original strategy and interchange the for x and for y loops. Then, instrument the modified code so that each `x` iteration is a task. Use `tareador_disable_object(address of variable)` to disable dependences due to data sharing to exploit parallelism among tasks if possible.

6

Recursive task decomposition analysis

The recursive pseudocode of the Mandelbrot set computation is shown in Figure 6.1.

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

Figure 6.1: Tile version of the recursive sequential code to compute the Mandelbrot set.

6.1 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 (DON'T DO IT until you add tasks! - there are not tasks and the execution will take a while).

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

6.2 Analysis of the strategies

Code `mandel-seq-rec-tar` has been only instrumented with the necessary `tareador_ON` and `tareador_OFF` calls.

We ask you to instrument the code `mandel-seq-rec-tar` with *Tareador* to evaluate and analyze a leaf and a tree parallel strategies.

In particular we ask you to perform and analyze two independent parallel strategies:

- Leaf Recursive Task Decomposition strategy. Remember that only base cases of the recursive program are tasks. Use `tareador_disable_object(address of variable)` to disable dependences due to data sharing to exploit parallelism among tasks if possible.
- Tree Recursive Task Decomposition strategy. In this case we ask you to create tasks following a recursive task decomposition, but also to exploit parallelism between both check vertical and horizontal border tasks. Use `tareador_disable_object(address of variable)` to disable dependences due to data sharing to exploit parallelism among tasks if possible.