PAR Laboratory Assignment
Lab 4: Parallel Task Decomposition Implementation and Analysis

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 four 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/lab4.tar.gz` in `boada.ac.upc.edu`. Uncompress it with these command lines in your home directory:

```
cd
tar -zxvf ~par0/sessions/lab4.tar.gz
```

Your first objective is to complete, execute and analyse the incomplete implementations of an interactive and a recursive task decomposition of Mandelbrot that we provide you. Those versions try to implement two of the parallel strategies you analysed in the previous laboratory. Once you complete those codes and analyse them, we will ask you to implement and analyse two new implementations (one iterative and one recursive) that overcome the provided versions. Finally, you will compare the best iterative and recursive versions.

The provided incomplete parallel versions: iterative and recursive, correspond to the first iterative parallel strategy and the leaf task decomposition of the recursive version, analysed with *Tareador*. In addition, the sequential iterative and recursive C implementation, a makefile, and a set of scripts to help you with the analysis: submit-seq.sh, submit-omp.sh, submit-strong-omp.sh, submit-strong-extrae.sh, are also provided.

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 elapsed execution times filling Table 2.2. Based on your performance results, explain which implementation you consider is the best one for the iterative and for the recursive versions. 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 OpenMP C source codes 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. Note that you have to fill in this table using the output files generated after the execution of `submit-strong-omp.sh` script. Based on your parallel strategies implementation 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 fill in table 2.1. Remember to complete the incomplete versions we have provided you before filling in their tables.

Section	Description
Task Decomposition	Name of the parallel strategy implementation
Code	Add the parallel source code of each implementation 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.
Modelfactor Analysis:	Include the *Modelfactors* tables to the report. In the case of Tile Iterative and Leaf Recursive Task Decomposition implementations, you should identify the main issues you observe based on the *Modelfactors* results. In the case of the Fine Grain Iterative and the Tree Recursive Task Decomposition Implementations, you should compare them to the Tile Iterative and Leaf Recursive Task Decomposition implementations and remark the improvements achieved. Remember that the *Modelfactors* analysis generates a directory with a set of execution traces, that can be used for the following table entry.
Paraver Analysis:	Include the *Paraver* views of the execution trace of 16 threads using the following hints: `Useful/Instantaneous parallelism` and `OpenMP tasking/explicit tasks function created and executed`, and comment them. In the case of Fine Grain iterative task decompostion, you should compare it to the Tile Iterative Task Decomposition Execution Trace. In the case of Tree Recursive Task Decompostion, you should compare it to the Leaf Recursive Task Decomposition Execution Trace.
Strong Scalability:	Include the scalability plot to the report and an explanation of the main aspects of the results. In the case of the Fine Grain Iterative Task Decomposition you should compare it to the Tile Iterative Task Decomposition. In the case of the Tree Recursive Task Decomposition you should compare it to the Leaf Recursive Task Decomposition.

Table 2.1: Analysis to be included in the pdf report and codes to be added into the zip file

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

	Number of threads					
Version	1	4	8	12	16	20
Iterative: Tile (provided, but should be completed)						
Iterative: Finer grain						
Recursive: Leaf (provided, but should be completed)						
Recursive: Tree						
Best Implementation	Version	**Reason why**				

Table 2.2: Summary of the elapsed execution times for each of the versions, obtained from the output files after the execution of submit-strong-omp.sh script

3

Experimental Setup

Laboratory files

You will find the following files:

- Makefile
- **Source code**
  - mandel-seq-iter.c : iterative sequential code.
  - mandel-seq-rec.c : recursive sequential code.
  - mandel-omp-iter.c : **incomplete** coarse grain iterative *OpenMP* parallel code that tries to implement the first parallel strategy we analysed with *Tareador*.
  - mandel-omp-rec.c : **incomplete** leaf recursive *OpenMP* parallel code that tries to implement the leaf parallel strategy we analysed with *Tareador*.
- **Scripts**
  - submit-seq.sh : script to execute the sequential code. Useful to obtain the expected outputs of Mandelbrot. You should use the outputs of the iterative and recursive sequential code to check the correctness of the outputs of your parallel programs.
  - submit-omp.sh : script to execute an *OpenMP* parallel code. Useful to check the correctness of your programs and find out the performance of your parallel program for a particular number of threads.
  - submit-strong-omp.sh : script to perform the strong scalability an *OpenMP* parallel code. This script will generate two output files with the information of the speedup and elapsed time of the mandelbrot function.
    - Speedup file: `speedup-partial-program-hostname.txt`
    - Elapsed execution times: `elapsed-partial-program-hostname.txt`. This file is used to fill in Table 2.2.
  - submit-strong-extrae.sh : script to perform the *Modelfactors* analysis of an *OpenMP* parallel code. Remember that its execution also generates a set of *Paraver* traces within the *Modelfactors* directory.

Mandelbrot parameters

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

```
./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 after execution the sequential codes to check the results of your parallel programs. Those two files are in binary format so 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 a set of scripts to analyse your parallel implementations.

Analysis

We suggest you to review the `lab1` sections and appendix to remember the functionality of *Modelfactors* and how to use it to generate and analyse the *OpenMP* implementations. Also, remember to look at your `lab3` laboratory to remember all you did.

4

Iterative task decomposition

The parallel iterative task decomposition TILE pseudocode of the Mandelbrot set computation is shown in Figure 4.1.

```
  for (int y= 0; y<numROW ; y+=TILE) {
      for (int x= 0; x<numCOL; x+=TILE) {
      #pragma omp task
      {
          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 4.1: Iterative Tile version of the parallel code to compute the Mandelbrot set.

4.1 Implementation

We ask you

1. Tile Iterative Task Decomposition: **Complete** the iterative parallel code we have provided so that it exploits parallelism and takes into account the dependences/data sharing you analysed in the Lab3 laboratory to avoid concurrency problems. This program corresponds to the *OpenMP* version of the `Original` strategy (Tiled) you analysed with *Tareador* in the previous laboratory.

2. Fine Grain Iterative Task Decomposition: **Implement the *OpenMP* version** of the `Finer` grain parallel strategy you analysed with *Tareador* in the previous laboratory, which exploited parallelism both between the check of the vertical and horizontal borders and during the full computation of the tile or fill it with the same color. Use the completed version of previous question (Tile Iterative Task Decomposition) as baseline of this new version as is; DO NOT remove the task of the previous version.

4.2 Execution

Run the sequential and parallel code to check it works:

- Run the sequential code with some parameters to generate histogram (`mandel_histogram.out`) and image (`mandel_image.jpg`) output:

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

- Rename the output of the sequential code. Then, run a parallel code with 20 threads to generate histogram (`mandel_histogram.out`) and image (`mandel_image.jpg`) output:

  ```
  sbatch submit-omp.sh mandel-omp-iter 20
  ```

4.3 Check the Correctness

Compare the output files: histogram `mandel_histogram.out` and image `mandel_image.jpg` of the iterative sequential and the parallel versions. You can use `cmp` or `diff` linux commands. If they differ you should review your code.

4.4 Performance Analysis

We ask you

Run the *Modelfactors* analysis, analyse the *Paraver* trace for 16 threads, and perform a strong scaling analysis of the completed provided code and the new implementation. It would be good that you also take into account the analysis done in lab3.

Fast Reminder:

1. *Modelfactors* analysis:

   ```
   sbatch submit-strong-extrae.sh mandel-omp-iter
   ```

   Look at `lab1` to remember how to read and understand the *Modelfactors* tables.

2. *Paraver* analysis:

   ```
   wxparaver mandel-omp-iter-strong-extrae/mandel-omp-iter-16-boada-XX-cutter.prv
   ```

   Look at `lab1` to remember how to open hints of *Paraver*.

3. Strong scalability. You should edit `submit-strong-omp.sh` script to set `SEQ` and `PROG` variables.

   ```
   sbatch submit-strong-omp.sh
   ```

5

Recursive task decomposition

The parallel leaf recursive pseudocode of the Mandelbrot set computation is shown in Figure 5.1.

```
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)
         #pragma omp task
         // fill full matrix tile with the same color
         // (if -d, display, if -h, update histogram)
         fill_tile_code
      else
         if (matrix tile size < TILE)
            #pragma omp task
            // 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 5.1: Leaf Recursive version of the recursive sequential code to compute the Mandelbrot set.

5.1 Implementation

We ask you

1. Leaf Recursive Task Decomposition: **Complete** the recursive leaf parallel code we have provided so that it can exploit parallelism and takes into account the dependences/data sharing you analysed in the Lab3 laboratory to avoid concurrency problems. This program corresponds to the *OpenMP* version of the `leaf` strategy you analysed with *Tareador* in the previous laboratory.

2. Tree Recursive Task Decomposition: **Implement the *OpenMP* version** of the `Recursive tree` task decomposition parallel strategy you analysed with *Tareador* in the previous laboratory. Your new implementation should try to maximize the exploited parallelism and consider the task creation and synchronization overheads.

5.2 Execution

Run the sequential and parallel code to check it works:

- Run mandelbrot to only measure its execution time:

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

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

```
sbatch submit-omp.sh mandel-omp-rec 20
```

5.3 Check the Correctness

Compare the output files: histogram `mandel_histogram.out` and image `mandel_image.jpg` of the recursive sequential and the parallel versions. You can use `cmp` or `diff` linux commands. If they differ you should review your code.

5.4 Performance Analysis

We ask you

Run the *Modelfactors* analysis, analyse the *Paraver* trace for 16 threads, and perform a strong scaling analysis of the completed provided code and `Tree` Recursive parallel strategy you analysed in previous laboratory.