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,
Jordi Tubella and Gladys Utrera

Spring 2023-24

UNIVERSITAT POLITÈCNICA
DE CATALUNYA
BARCELONATECH

Index

Note:

- 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 this command line:
`"tar -zxvf ~par0/sessions/lab4.tar.gz"`.

Laboratory Deliverable Information

IMPORTANT NOTE: 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.

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 developed. 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.

Section	Description
Code	Add the parallel source code of each implementation to the zip. Indicate the name of the source code in the pdf.
Modelfactor Analysis:	For each parallel code, you should include the *Modelfactors* tables to the report. In the case of the incompleted implementation we have provided and, once it has been completed, identify the main issues you observe. In case of the new implementation, compare the results of the new implementation to those of the completed provided version and remark the improvements achieved. Remember that the *Modelfactors* analysis generates a directory with a set of execution traces.
Paraver Analysis:	For each parallel code, you should 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`. For both iterative and recursive version, compare the completed implementation provided and your new implementation.
Strong Scalability:	For each parallel code, you should include the scalability plot to the report and an explanation of the main aspects of the results. For both iterative and recursive, compare the results of the new implementation to the completed implementation provided.

Table 1: Analysis to be included in the pdf report and codes to be added into the zip file

Comparison: Table 2 should be included after the previous analysis. Based on your performance results, explain which implementation you consider is the best one for the iterative and for the recursive 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

	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						

Table 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

the third language competence evaluation" document to know the *R*ubric that will be used.

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.
  - 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.

Madelbrot parameters

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 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 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.

1

Iterative task decomposition

The parallel iterative task decomposition TILE pseudocode of the Mandelbrot set computation is shown in Figure 1.1.

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

We ask you

1. **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 previous laboratory to avoid concurrency problems.
2. **Implement a new iterative task decomposition code**. This should be the *OpenMP* version of the Finer grain parallel strategy you analysed with *Tareador* in the previous laboratory, which exploited parallelism both during the check of the borders and during the full computation of the tile or fill it with the same color. We advise you to use the completed version of previous question as baseline of this new version.

1.1 Execution and Check the Correctness

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
  ```

- Compare the output files.

Note: elapsed execution time for only `mandel` function can be found in `sumbit-???.sh.oXXXXX` files.

1.2 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.

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
   ```

2

Recursive task decomposition

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

We ask you

1. **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 previous laboratory to avoid concurrency problems.
2. **Implement a new recursive task decomposition code**. This should be 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.

2.1 Execution and Check the Correctness

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
  ```

- Compare the output files.

2.2 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.