PAR Laboratory Assignment
Lab 5: Parallel Data 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 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/lab5.tar.gz` in `boada.ac.upc.edu`. Uncompress it with these command lines in your home directory:

```
cd
tar -zxvf ~par0/sessions/lab5.tar.gz
```

IMPORTANT NOTE: We have simplified the sequential mandel source code. However, the data dependency analysis that you did in lab3 is still valid. The new version is now more simple and there are not tiles neither checking of the border tiles. The computation is always done for the full matrix. Your main objective is to analyse three different geometric data decompositions and compare them each other. Table 2.1 explains what should be included in the report, in addition of a summary of elapsed execution times and cache misses information filling Table 2.2. Based on your performance results, explain which implementation you consider is the best one for the 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.
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

You should fill in table 2.1.

Section	Description
Codes	Add the parallel source code of each implementation (Block Geometric Data Decomposition by columns, Block-Cyclic Geometric Data Decomposition by columns and Cyclic Geometric Data Decomposition by rows) 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 for each of the implementations and compare the results of the three implementations. Remember that the *Modelfactors* analysis generates a directory with a set of execution traces, and a summary file: modelfactors.out or modelfactor-tables.pdf.
Paraver Analysis:	Include the *Paraver* views of the execution traces of 16 threads using the following hints: `Useful/Instantaneous parallelism` and `OpenMP/implicit tasks in parallel constructs` of the three implementations. Make all the traces have the same duration and size than the longest execution trace, to easily compare them.
Memory Analysis:	For each parallel code, you should submit submit-strong-memory.sh script. This will generate several traces in a directory called nameprogram-strong-memory and a file: `l2_misses_...txt`. Those files contain the overall number of cache misses (the accumulation for all the threads) and the average number of misses per thread of the second level of cache. Include this information in the report. For an specific number of threads, you can also open the trace and use the *Paraver* configuration file that we have provided you: `l2d_implicit.cfg`. Include the profile information of the *Paraver* view of the execution trace of 16 threads using the *Paraver* configuration provided. Compare the three implementations based on the number of `l2_misses` and try to reason the differences among them.
Strong Scalability:	For each data decomposition parallel code, you should include the scalability plot to the report. Compare all three implementations based on the scalability, modelfactors and memory results.

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. 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` and `submit-strong-memory.sh` scripts. Based on your parallel strategies implementation results, reason which will be the best parallel strategy implementation.

	Number of threads (elapsed)					
Version	1	4	8	12	16	20
1D Block Geometric Data Decomposition by columns						
1D Block-Cyclic Geometric Data Decomposition by columns						
1D Cyclic Geometric Data Decomposition by rows						
	Number of threads (L2 Cache Misses per thread)					
1D Block Geometric Data Decomposition by columns						
1D Block-Cyclic Geometric Data Decomposition by columns						
1D Cyclic Geometric Data Decomposition by rows						
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. Average L2 cache misses per thread are obtained from the memory analysis script execution.

3

Experimental Setup

Laboratory files

You will find the following files:

- Makefile
- **Source code**
  - mandel-seq-iter-simple.c : simple (no tiled) iterative sequential code.
  - mandel-omp-iter-simple.c : simple (no tiled) iterative baseline *OpenMP* code. There is only the necessary code to allow a correct Modelfactor analysis. You have to implement the *OpenMP* codes from this baseline code.
- **Scripts**
  - submit-seq.sh : script to execute the sequential code. Useful to obtain the expected outputs of Mandelbrot. You should use the output of the iterative 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 and the tables that you can analyze.
  - submit-strong-memory.sh : script to perform the memory hierarchy analysis of an *OpenMP* parallel code. Its execution generates a set of *Paraver* traces within a directory, and a file with the information of the second level of cache misses.

Mandelbrot parameters

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

```
./mandel-seq-iter-simple -help
Usage: ./mandel-seq-iter-simple [-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 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

Geometric Data Decomposition Strategies

4.1 Implementation

Important: You are NOT allowed to use the `for` worksharing clause in OpenMP.
We ask you to implement:

1. **An iterative data decomposition code** using a 1D Block Geometric Data Decomposition by columns. As much as possible, your implementation should minimize the load unbalance due to the data decomposition and exploit data locality accessing matrix `M`.
2. **An iterative data decomposition code** using a 1D Block-Cyclic Geometric Data Decomposition by columns. As much as possible, your implementation should reduce cache coherence protocol overheads and exploit data locality accessing matrix `M`.
3. **An iterative data decomposition code** using a 1D Cyclic Geometric Data Decomposition by rows. As much as possible, your implementation should minimize the load unbalance due to the data decomposition, reduce cache coherence protocol overheads and exploit data locality accessing matrix `M`.

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-simple -h -o -i 256
```

- 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-simple 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 to:

1. Run the *Modelfactors* analysis.
2. Complete the analysis of *Modelfactors* with a memory hierarchy analysis. We have provided you with a new script to perform a memory hierarchy analysis. Submit your parallel codes with this new script. You may want to look at the output of `lstopo` at lab1 to remember the memory hierarchy in the NUMAnodes.
3. Perform a strong scalability analysis of the *OpenMP* codes. It would be good that you also take into account the analysis done in lab3.

Fast Reminder:
Some hints/reminders to be able to perform the previous analysis:

- *Modelfactors* analysis:

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

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

- Memory hierarchy analysis:

  ```
  sbatch submit-strong-memory.sh mandel-omp-iter-simple
  ```

  It will generate execution traces and two files with the overall number of L2 misses.

- *Paraver* analysis:

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

  Look at `lab1` to remember how to use *Paraver*. We have provided you with to *Paraver* configuration files that you can load to see the profiling amount of L2 misses per thread execution.

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

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