departamento de informática
FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Concurrency and Parallelism 2018-19

Project — Parallel Patterns (v1.2)

João Lourenço

October 31, 2018

Abstract

This document describes the project assignment for the course of Concurrency and Parallelism.

1 Introduction

Up to last week we have been learning and discussing about Patterns for Parallel Programming. In this project you are asked to collaborate with another colleague and implement a set of Patterns for Parallel Programming using Cilk+. You are supposed to implement at least the following patterns: `Map`, `Reduce`, `Scan`, `Pack`, `Gather`, `Scatter`, `Pipeline`, `Farm`, and optimize your code to be as fast and as scalable as possible.

2 Important dates

Date	**Action**
October 31, 2018	Publication of the Project Assignment.
November 28, 2018	Last commit for code.
December 2, 2018	Last commit for report.

Pease note that you have a few extra days for submitting the report. Your (performance) tests SHOULD have be ran before November 28, and your text SHOULD have been witten before as well. Use these days, from Nov 28 to Dec 2, only for refinement and control quality of the report.

2.1 Working rules

The following working rules apply. This list is not exhaustive, so in case of doubt, do not hesitate in asking.

- The project work will be done in grops of 3 people (unless explicitly authorized otherwise).
- Each group will fork the given main repository and work on their own private repository (remember to keep your repository private *and not public*).

- Each student will commit his/her own work into the shared repository. I expect at least one commit on each day that you work on the project.
- Commit messages must describe clearly what was changed/added in that commit (and why).
- Remember to push your commits to your group repository.
- The project report must have the look and feel of a research paper, using the IEEE template for Computer Society Journals (`https://goo.gl/Xtjdh4`), with a maximum of 5 pages (references may be on the 6th page).

2.2 Project grading aspects

The Project's grade [$\mathcal{P}$] will take into consideration the following aspects. The order is not relevant and the list is not exhaustive. In case of doubt, do not hesitate in asking.

- The project will be graded according to the following criteria:
  - Projects that fail to compile and execute with the following procedure will receive the final grade of 0 points:

    ```
    clone <your_repository>
    cd  <your_repository>
    make
    ./main NUMBER
    ```
  - The quality of the work as perceived from project report;
  - The quality of the text in the project report (well organized, complete, the text/message is clear, no misspellings, etc)
  - The project's code look and feel;
  - A perfect project which implements all (and only) the above listed patterns will have a top grande of 17 (over 20) points.
- Some points are reserved for complementary work/achievements, such as:
  - Some points will be given to the implementation of more patterns (unlisted above). Please discuss with me the *signature* of the patterns you want to implement.
  - Some points will be given to the implementation of more unit/integration ing functions, if shared publicly with the colleagues and used and acknowledged by them in their reports.
  - Some points will be given to the project that exhibits the best (timed) speedup (compiling and running the s in the *server node9*).
  - Some points will be given to the project that exhibits the best scaled (data size) speedup for 16 processors (compiling and running the s in the *server node9*).

2.3 Student grading aspects

The Students's grade [$\mathcal{S}$] will take into consideration the following aspects. The order is not relevant and the list is not exhaustive. In case of doubt, do not hesitate in asking.

- Relevance of the student's commit messages.
- Relevance of the code committed by the student.
- Students that do not exhibit evidences of working on the project (e.g., with no commits, or with nearly empty/non-meaningful commits, or with insufficient/unclear commit messages) will fail the course.

2.4 Student/Project Final grading rule

- Formula: $\mathcal{G} = 0.7 * \mathcal{P} + 0.3 * \mathcal{S}$

3 Project Description

In this lab work you are given a working version of the projet. It will compile and run and produce results, although the execution will be slow as all the given implementations for the patterns are sequential. You are asked to study the given code a create parallel versions of the patterns using Cilk+.

3.1 Given Version

You are given a working (sequential) version of the project at

`https://bitbucket.org/cp201819/project_parallel_patterns.git`

This repository contains two directories/folders: `src` and `doc`. The former contains the base source code, and the latter will contain your Project Report as a PDF file. Please name your report as `report_AAAAA_BBBBB_CCCCC.pdf`, where `AAAAA`, `BBBBB` and `CCCCC` are the numbers of the group members sorted in increasing order (lowest to highest).

Fork the above repository and name your groups repository as

`cp2018-19_project_AAAAA_BBBBB_CCCCC`

where `AAAAA`, `BBBBB` and `CCCCC` are the numbers of the group members sorted in increasing order (lowest to highest).

In your Linux device (own laptop, lab workstation, or as a last resort, in the "node9" server used in the last lab class) clone your new repository and then try compile your code using the command `make` in the `src` directory. It must compile with no errors nor warnings.

3.2 Code Structure

debug.c debug.h main.c patterns.c patterns.h unit.c unit.h

The project include the following source files:

Files	Description
`debug.c debug.h`	Functions for printing the contents of the array(s), useful for debugging (activated with the option "-d").
`patterns.c patterns.h`	The patterns to be implemented. The ".c" file contains sequential implementations of the patterns.
`unit.c unit.h`	Functions for unit testing of each pattern.
`main.c`	The main program.

3.3 Work plan

Your job is to make an optimized parallel version (using Cilk+) of all the patterns listed in the files `patterns.c/patterns.h`!.

Please note that to implement the parallel patterns, you may use Cilk's *parallel for* and *fork/join*, but you cannot use the reducers available in the Cilk library.

I suggest you follow these steps:

1. Clone/fork the given project.
2. Compile the given version. Study the source code and understand how it works.
3. Discuss with your colleagues how to split the work.
4. Check the provided sequential implementation of each pattern.
5. Compile and run the tests and confirm the results.
6. Implement a parallel version of each pattern.
7. Compile and run the tests and confirm the results.
8. For each pattern, measure its perfocrmance/scalability/scaled scalability. You may experiment with different numbers of processors (by setting the environment variable `CILK_NWORKERS`).
9. Optimize the given code.
10. Go back to item 7. until satisfied.
11. Write the report. Revise the report. Please put your report in the `doc` directory and name it as `report_AAAAA_BBBBB_CCCCC.pdf`, where `AAAAA`, `BBBBB` and `CCCCC` are the numbers of the group members sorted in increasing order (lowest to highest).
12. *Optional:* implement and optimize some more parallel patterns.
13. *Optional:* implement and share some more tests (unit or integration tests).
14. *Optional:* complete the report and revise again.

Please remember to commit regularly your changes, and please always write meaningful commit messages.

4 Project Report — What to write and how?

In your project, I'm expecting to see evidence of work done by (candidates to) engineers and not by programmers. This means I'm not interested to read in the report that you implemented de class X with methods A and B. I'm interested in you decisions and why you made then, and whether they did or did not resulted as you expected. And if they didn't, why was that?

I'm expecting the report to have the look and feel of a research paper. In the following list of sections, some are mandatory, others are optional and must be managed with *good sense*. You may merge two of the proposed sections into a single one, or split on into more sections/subsections and you feel adequate.

Abstract — A very short summary of your work. In these two or three paragraphs you should be able to answer the following questions:

1. What is the problem?
2. Why is it interesting/challenging?
3. What is your approach?
4. What were the results? What did you learn?

Introduction — Extended version of the abstract (don't copy&paste text from/to introduction to/from abstract.)

Architecture / Model / Solution — Describe your approach/solution in a way that is as independent as possible form your implementation. What you describe here should be (as much as possible) neutral to the programmin language and to the parallelization framework being used.

Implementation — Showcase for very relevant details in your implementation.

Experimental evaluation — How did you make your functional/correctness and performance evaluation? Were there are relevant results? How do you explain them?

Conclusions — Lessons learned!

Acknowledgments — Please acknowledge the colleagues (external to your group) that directly or indirectly (e.g., on Piazza) helped you during the development of your project.

References — List your sources of information (books, papers, web sites, …) here, but also be sure you cite them in the main text at least once. *This section does not count for the page limit!*

Comments — I will appreciate any (constructive — which does not necessarily mean positive) feedback you may think relevant to give about this project. Possible topics include dates and timing, difficulty level, work required, interest, challenging, adequacy, integration into the course schedule (contents of the previous lectures and labs), …. Please feel free to talk about anything you think will help any future edition of this course. *This section does not count for the page limit!*

5 Questions/Discussion

Please ask your questions using the Piazza system. Either public (if possible) or private (if really necessary).