Click to show/hide PDF Layer

Lab 1

Ricard Borrell Pol[†], Sergi Laut Turón[‡], Imanol Muñoz Pandiella[§]

Instructions

The first Lab session consists of 4 problems with several questions each. You have to compile your answers to the exercises in a report, explaining how have you solved the problem and the answers to the questions. This lab assignment has a weight of 15% of the total grade of the subject and the deadline for it is April 30th at 23:59.

Each group will have to submit a compressed file named lab1_TxGyy.zip, where TxGyy is your group identifier. A .tar or a .tgz file is also accepted (e.g. lab1_T1G1.zip or lab1_T2G21.zip). The compressed file has to contain four folders (1_cholesky, 2_sudoku, 3_histogram, and 4_product) and all the requested files with the following structure.

```
lab1_TxGyy.zip ─┬─ 1_cholesky ─┬─ cholesky.c
                │              ├─ cholesky.h
                │              ├─ main.c
                │              ├─ Makefile
                │              ├─ job.cmd
                │              └─ output_files (.out .err)
                ├─ 2_sudoku ─┬─ sudoku.c
                │            ├─ Makefile
                │            ├─ job.cmd
                │            └─ output_files (.out .err)
                ├─ 3_histogram ─┬─ histogram.c
                │               ├─ random.c
                │               ├─ random.h
                │               ├─ Makefile
                │               ├─ job.cmd
                │               └─ output_files (.out .err)
                ├─ 4_product ─┬─ product.c
                │             ├─ Makefile
                │             ├─ job.cmd
                │             └─ output_files (.out .err)
                └─ report.pdf
```

A sample file named lab1_TxGyy.zip containing the reference codes has been published in Aula Global. To ease the process we provide sample job files and Makefiles. You may need to modify the execution lines in the job scripts to perform your tests and the batch lines to request more cores. The Makefiles should not be

[†] ricard.borrell@upf.edu.
[‡] sergi.laut@upf.edu.
[§] imanol.munoz@upf.edu.

modified unless the exercise says so. Focus on the code and the work requested for each exercise. In the case the compilation fails, the job will finish and it will not try to run the binary.

For any arising question, please, post it in the Lab class forum in Aula Global. However, do not post your code in the Forum. For other questions regarding the assignment that you consider cannot be posted in the Forum (e.g. personal matters or code), please, contact the responsible for the lab sergi.laut@upf.edu.

Criteria

The codes will be tested and evaluated on the same cluster where you will be working. The maximum grade on each part will only be given to these exercises that solve in the most specific way and that tackle all the functionalities and work requested. All the following criteria will be applied while reviewing your labs in the cluster.

Exercises that will not be evaluated:

- A code that does not compile.
- A code giving wrong results.
- A code that does not adhere to all the input/output requests.
- lab1_TxGyy.zip delivered files not structured or named as described previously.

Exercises with penalty:

- A code with warnings in the compilation.

1. Cholesky Factorization

The Cholesky factorization (or Cholesky decomposition) is a factorization of a symmetric, positive matrix into two matrices. Cholesky generates two matrices, the lower matrix (L) and the upper matrix (U), where these names refer to the lower and upper part of the diagonal of the matrix. In the Cholesky factorization, the generated matrices are the transpose of the other. $L = U^T$, and vice versa.

In this exercise, we will have to finish the sequential version of Cholesky in cholesky() and then parallelize it in cholesky_openmp(). We will calculate U from A and then calculate L by doing the transpose of matrix U. **The matrix transpose operation has to be implemented to optimize the efficiency of L1 cache memory usage.** The formulas used to calculate the elements of U are the following, depending if they are diagonal elements or non-diagonal elements.

$$u_{ij} = \frac{a_{ji} - \sum_{k=0}^{i-1} u_{kj} u_{ki}}{u_{ii}}$$

(a) Off-diagonal entries.

$$u_{ii} = \sqrt{a_{ii} - \sum_{k=0}^{i-1} u_{ki}^2}$$

(b) Diagonal entries.

It is important to notice that we only need to compute the corresponding elements of the diagonal and the lower elements (for L) and upper elements (for U). For the sake of simplicity, L and U may be square matrices, and have the rest of the elements be zero.

Once calculated the Cholesky factorization we want to check that it has been calculated correctly. This is done by performing $B = LU$ and later verifying that $B = A$. Multiply matrices LU and calculate B. Matrix B will have the same size as A. After generating B, compare the elements of B and A and check that the difference between elements is less than 0.001%. In the file cholesky.c there are the two C functions that we will work with. We will need to first finish the sequential implementation and later parallelize all 5 steps for the OpenMP version. The steps of our functions are the following.

- Matrix initialization for A, L, U, and B (already done).
- Compute Cholesky factorization for U.
- Calculate L from U^T: iterate only over the non-zero elements.
- Compute $B = LU$: matrix multiplication. Iterate only over non-zero elements of L.
- Check if all elements of A and B have a difference smaller than 0.001%.

Here there is a sample output of the code (results are not extracted from the cluster). Notice that argv[1] is the value n for an nxn matrix.

```
Sequential Cholesky
Initialization: 0.160174
Cholesky: 52.790788
L=U': 0.051511
B=LU: 88.952372
Matrices are equal
A==B?: 0.015213

OpenMP Cholesky
Initialization: 0.161941
Cholesky: 3.670381
L=U': 0.003441
B=LU: 5.572723
Matrices are equal
A==B?: 0.001657
```

Report Questions 1 — Cholesky

1. Expose your parallelization strategy to divide the work in the Cholesky algorithm and in the matrix multiplication. Justify the selection of the scheduler and chunk size and compare different schedulers with different chunk sizes and show the results.

2. Make two plots: one for the speedup of the Cholesky factorization and another for the matrix multiplication for n=3000. Use 1, 2, 4, 8, and 16 cores for a strong scaling test. Plot the ideal speedup in the figures and use a logarithmic scale to print the results. Discuss the results.

2. Sudoku solver

Sudokus are logic-based combinatorial puzzles. In a typical Sudoku, numbers from 1 to 9 have to be placed in a 9x9 grid in such a way no row, column, or one of the 9 3x3 sub-grid blocks has a duplicate or misses one number.

In its initial conditions, the grid is partially completed and should lead to a unique solution.

In this exercise, we provide a sequential code that solves a Sudoku. Your goal is to parallelize it using tasks.

The code requires no arguments and the output is already coded.

Report Questions 2 — Sudoku

1. Briefly explain how the code works.
2. Explain how the parallelized code distributes the workload among threads.
3. Add the clause $final(level > 1)$ to your task-generating pragma. What does it do?

3. Histogram

In this exercise, we provide a program that will fill an array with pseudo-random values, build a histogram of that array, and then compute statistics. This can be used as a simple test of the quality of a random number generator.

The code is sequential. The goal of this exercise is to parallelize it using three different methods: using critical pragma, locks, and a reduction.

The code requires no arguments.

The output must be like this. It must execute the 4 variants one after the other. This execution has been performed on a laptop. Thus, your times can be different. Remember to reinitialize the histogram after each variant.

```
4 threads
Sequential  histogram for 50 buckets of 1000000 values
ave = 20000.000000, std_dev = 394.372925
in 0.003438 seconds
par with critical  histogram for 50 buckets of 1000000 values
ave = 20000.000000, std_dev = 394.372925
in 0.077274 seconds
par with locks  histogram for 50 buckets of 1000000 values
ave = 20000.000000, std_dev = 394.372925
in 0.058147 seconds
par with reduction  histogram for 50 buckets of 1000000 values
ave = 20000.000000, std_dev = 394.372925
in 0.000601 seconds
```

Report Questions 3 — Histogram

1 Explain how have you solved each of the parallelizations.

2 Explain the time differences between different parallel methods if there are any.

3 Make a speedup plot for the different parallelization methods for 1, 2, 4, 8, and 16 cores. Discuss the results.

4. Product-consumer

In this exercise, we will work with a simple SPMD product/consumer program. In this program, an array (the product) is filled with random numbers, and after, a computation with those numbers is done (the product is consumed).

Your goal in this exercise is to modify the code in such a way the product is performed by one thread and consumed by another thread.

You have to create a parallel region and only modify or extend the code **inside** the region. You have to use sections. The code must work only with 2 threads. If the number of threads is different than 2 it must return the following message and exit:

```
Error: incorrect number of threads, %i
```

Where %i is the type of an integer in the printf() function.

Report Questions 4 — Product-consumer

1 Explain how have you synchronized the threads.