Distributed and Parallel Systems — Due on Monday, February, 26, 2024

Assignment 1

CS4402B / CS9635B — *University of Western Ontario*

Submission instructions.

Format: The answers to the problem questions should be typed:

- source programs must be accompanied with input test files and,
- in the case of `Cilk` code, a `Makefile` (for compiling and running) is required, and
- for algorithms or complexity analyzes, LaTeX is highly recommended.

A PDF file (no other format allowed) should gather all the answers to non-programming questions. All the files (the PDF, the source programs, the input test files and Makefiles) should be archived using the UNIX command `tar`.

Submission: The assignment should submitted through the OWL website of the class.

Collaboration. You are expected to do this assignment *on your own* without assistance from anyone else in the class. However, you can use literature and if you do so, briefly list your references in the assignment. Be careful! You might find on the web solutions to our problems which are **not appropriate**. For instance, because the parallelism model is different. So please, avoid those traps and work out the solutions by yourself. You should not hesitate to contact the instructor or the TA if you have any questions regarding this assignment. We will be more than happy to help.

Marking. This assignment will be marked out of 100. A 10 % bonus will be given if your paper is clearly organized, the answers are precise and concise, the typography and the language are in good order. Messy assignments (unclear statements, lack of correctness in the reasoning, many typographical and language mistakes) may yield a 10 % malus.

PROBBLEM 1. [60 points] Let A be a $n \times n$ lower triangular matrix, where every diagonal element is non-zero. Hence, the matrix A is invertible. We assume that n is power of 2. A simple divide-and-conquer strategy to compute the inverse A^{-1} of A is described below. Let A be partitioned into $(n/2) \times (n/2)$ blocks as follows:

$$A = \left[\begin{array}{cc} A_1 & 0 \\ A_2 & A_3 \end{array} \right]. \tag{1}$$

Clearly A_1 and A_3 are invertible lower triangular matrices. The matrix A^{-1} is given by

$$A^{-1} = \left[\begin{array}{cc} A_1^{-1} & 0 \\ -A_3^{-1} A_2 A_1^{-1} & A_3^{-1} \end{array} \right] \tag{2}$$

We assume that we have at our disposal a `Cilk`-code for matrix multiplication, such as the one posted on the course web site based on the DnC algorithm studied in class.

Question 1. [10 points] Write a `Cilk`-like multi-threaded algorithm (that is pseudo-code in the fork-join model) computing A^{-1}.

Question 2. [5 points] Analyze the work and critical path of your multi-threaded algorithm.

Question 3. [25 points] Realize a `Cilk` implementation of your multi-threaded algorithm using matrices with floating point numbers. Your code must use a threshold B such that when the order satisfies $n \leq B$, recursive calls are no longer spawned. For the tests, use matrices with randomly generated coefficients, with absolute value between 1/10 and 10. You must provide two types of tests with your code:

- **correctness tests:** a couple examples with $n = 4$ (with B taking values 1, 2, 4) for which your code verifies that AA^{-1} equals the identity matrix;
- **performance tests:** tests for which n takes successive powers of 2, namely $4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048$ and B varies in the range $32, 64, 128$.

Note that it is possible to avoid recursive calls for $n < B$ by simply writing a for-loop for forward substitution. Here are three matrices A_1, A_2, A_3 with integer coefficients such that the inverse A_I^{-1} has also integer coefficients. These so-called *unimodular matrices* are convenient for testing the correctness of your code and will avoid issues with floating point arithmetic:

$$A_1 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ -1 & 1 & 0 & 0 \\ -1 & -1 & 1 & 0 \\ -1 & -1 & -1 & 1 \end{bmatrix}, \quad A_2 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ -1 & 1 & 0 & 0 \\ 1 & -1 & 1 & 0 \\ 1 & 1 & -1 & 1 \end{bmatrix}, \quad A_3 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix},$$

and we have:

$$A_1^{-1} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 2 & 1 & 1 & 0 \\ 4 & 2 & 1 & 1 \end{bmatrix}, \quad A_2^{-1} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ -2 & 0 & 1 & 1 \end{bmatrix}, \quad A_3^{-1} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ -1 & 1 & 0 & 0 \\ 0 & -1 & 1 & 0 \\ 0 & 0 & -1 & 1 \end{bmatrix}.$$

Note that the patterns in the matrices A_1, A_2, A_3 are easy to generalize to arbitray n so that $A_1^{-1}, A_2^{-1}, A_3^{-1}$ still have integer coefficients.

Question 4. [5 points] The best choice for B depends on various factors, in particular cache sizes, parallelization overheads. Determine experimentally (reporting your experimental data) what is the best choice for B, for

1. the serial elision of your code that is when `cilk_spawn` and `cilk_sync` are erased.
2. the multi-threaded version of your code run on a multi-core processor with 4 cores (or more).

Question 5. [5 points] Collect running times for the performance tests on a multi-core processor with 4 cores (or more) comparing the serial elision of your code against the multi-threaded version of your code. You should report running times using plots. Please indicate the type (brand, model, cache size) of processor you are using. If this processor uses https://en.wikipedia.org/wiki/Hyper-threading/hyper-threading technology, please check whether this has been turned on or not, and report the result in your assignment.

Question 6. [10 points] Using the ideal-cache model, analyze the cache complexity of the serial elision of the algorithm proposed in Question 1.

PROBBLEM 2. [40 points]

In this problem, we develop a divide-and-conquer algorithm for the following geometric task, called the *CLOSEST PAIR PROBLEM* (CSP):

Input: A set of n points in the plane

$$\{p_1 = (x_1, y_1), p_2 = (x_2, y_2), \ldots, p_n = (x_n, y_n)\},$$

whose coordinates are floating point numbers (positive, null or negative).

Output: The closest pair of points, that is, the pair $\{p_i, p_j\}$ with $p_i \neq p_j$ for which the distance between p_i and p_j, that is,

$$\sqrt{(x_j - x_i)^2 + (y_j - y_i)^2}$$

is minimized.

For simplicity, we assume that n is a power of 2 and that all the x-coordinates x_i are pairwise distinct, as well are the y-coordinates y_i. Here's a high-level overview of the proposed algorithm:

1. Find a value x for which exactly half of the points satisfy $x_i < x$ and the other half satisfy $x_i > x$, thus splitting the points into groups L and R.

2. Recursively find the closest pair in L and the closest pair in R. Let us call these pairs $\{p_L, q_L\}$, with $p_L, q_L \in L$, and $\{p_R, q_R\}$, with $p_R, q_R \in R$; we denote by d_L (resp. d_R) the distance between p_L and q_L (resp. p_R and q_R). Let d be the smallest of these two distances.

3. It remains to be seen whether or not there is a point in L and a point in R that are less than distance d apart from each other. To this end, discard all points with $x_i < x - d$ or $x_i > x + d$. Then, sort the remaining points by y-coordinate.

4. Now, go through this sorted list, and for each point, compute its distance to the *six subsequent points* in the list. Let p_M, q_M be the closest pair found in that way.

5. The answer is $\{p_L, q_L\}$, $\{p_R, q_R\}$ or $\{p_M, q_M\}$, whichever is closest.

Why are six subsequent points sufficient in the algorithm above? In fact this follows from an elementary geometrical argument the proof of which can be found at https://sites.cs.ucsb.edu/ suri/cs235 and which states that: A rectangle of width d and height $2d$ can contain at most six points such that any two points are at distance at least d.

Question 1. [10 points] Write down pseudo-code (using the syntax of the C language) for the algorithm, and show that its work is given by the recurrence:

$$W(n) = 2W(n/2) + O(n\log(n))$$

Deduce that $W(n) \in O(n\log^2(n))$.

Question 2. [5 points] Propose a multi-threaded version of the above algorithm (for the fork-join model) and show that its parallelism is limited to $O(\log(n))$.

Question 3. [10 points] Propose an improved multi-threaded algorithm (for the fork-join model) with a parallelism of $O(n/\log(n))$.

Question 4. [15 points] Realize a either a `Cilk` implementation or a `Julia` implementation of the multi-threaded algorithm proposed in Question 3. In any case, provide test cases and study experimentally how your program scales with larger and larger input data sets. If you use `Cilk`, provide test cases as part of your `cpp` file, following the style of the examples studied in class. If you use `Julia`, provide a `Jupyter` notebook.