ECEC-622: Introduction to Parallel Computer Architecture Midterm Exam

Prof. Naga Kandasamy, ECE Department, Drexel University

May 9, 2013

The exam is due May 15, 2013, by midnight. You may work on this exam in a team of up to two people.

1. **The Cholesky Decomposition.** This problem asks you to develop multi-core implementations of the Cholesky decomposition method. A brief description of Cholesky decomposition follows.

Consider the problem of solving a system of linear equations of the form $Ax = b$ where A is a $n \times n$ coefficient matrix, and x and b are $n \times 1$ vectors. Here, A and b are known and we seek to determine the solution vector x. If the matrix A is symmetric and positive definite, that is $x^{\mathrm{T}}Ax > 0$ for all non-zero vectors x[1], then Cholesky decomposition is an efficient method of computing an upper triangular matrix U with positive diagonal elements such that $U^{\mathrm{T}}U = A$. So, to solve $Ax = b$, one solves first $U^{\mathrm{T}}y = b$ for y and then $Ux = y$ for x.

Consider a simple example of how to obtain the Cholesky decomposition of a 4×4 symmetric matrix A. We would like to obtain a corresponding 4×4 upper triangular matrix U such that $A = U^TU$. So,

$$\begin{aligned}
\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix} &= \begin{pmatrix} u_{11} & 0 & 0 & 0 \\ u_{12} & u_{22} & 0 & 0 \\ u_{13} & u_{23} & u_{33} & 0 \\ u_{14} & u_{24} & u_{34} & u_{44} \end{pmatrix} \times \begin{pmatrix} u_{11} & u_{12} & u_{13} & u_{14} \\ 0 & u_{22} & u_{23} & u_{24} \\ 0 & 0 & u_{33} & u_{34} \\ 0 & 0 & 0 & u_{44} \end{pmatrix} \\
&= \begin{pmatrix} u_{11}^2 & u_{11}u_{12} & u_{11}u_{13} & u_{11}u_{14} \\ u_{11}u_{12} & u_{12}^2 + u_{22}^2 & u_{12}u_{13} + u_{22}u_{23} & u_{12}u_{14} + u_{22}u_{24} \\ u_{11}u_{13} & u_{12}u_{13} + u_{22}u_{23} & u_{13}^2 + u_{23}^2 + u_{33}^2 & u_{13}u_{14} + u_{23}u_{24} + u_{33}u_{34} \\ u_{11}u_{14} & u_{12}u_{14} + u_{22}u_{24} & u_{13}u_{14} + u_{23}u_{24} + u_{33}u_{34} & u_{14}^2 + u_{24}^2 + u_{34}^2 + u_{44}^2 \end{pmatrix}
\end{aligned}$$

Equating the left and the right-hand sides, we obtain

$$\begin{aligned}
u_{11} &= \sqrt{a_{11}}, u_{12} = \frac{a_{12}}{u_{11}}, u_{13} = \frac{a_{13}}{u_{11}}, u_{14} = \frac{a_{14}}{u_{11}} \\
u_{22} &= \sqrt{a_{22} - u_{12}^2}, u_{23} = \frac{a_{23} - u_{12}u_{13}}{u_{22}}, u_{24} = \frac{a_{24} - u_{12}u_{14}}{u_{22}} \\
u_{33} &= \sqrt{a_{33} - u_{13}^2 - u_{23}^2}, u_{34} = \frac{a_{34} - u_{13}u_{14} - u_{23}u_{24}}{u_{33}} \\
u_{44} &= \sqrt{a_{44} - u_{14}^2 - u_{24}^2 - u_{34}^2}
\end{aligned}$$

Examining the above equations, u_{12} can be computed easily since the key variable on the right-hand side, u_{11} is already known in the previous step. Similarly, u_{22} can be easily computed since u_{12} is known during the previous step, and so on. A serial implementation of a row-oriented Cholesky decomposition algorithm is shown in the next page. Unlike Gaussian elimination, Cholesky decomposition does not require pivoting since the diagonal element $A[k, k] > 0$ if the matrix A is positive definite. So, our implementation is numerically stable.

[1]A matrix is positive definite if and only if the determinant of each of the principal sub-matrices is positive.

```
procedure CHOLESKY($A$)
int $i$, $j$, $k$;
for $k$ := 0 to $n-1$ do
    $A[k,k] := \sqrt{A[k,k]}$;    /* Obtain the square root of the diagonal element. */
    for $j := k+1$ to $n-1$ do
        $A[k,j] := A[k,j]/A[k,k]$;    /* The division step. */
    end for
    for $i := k+1$ to $n-1$ do
        for $j := i$ to $n-1$ do
            $A[i,j] := A[i,j]$ - $A[k,i] \times A[k,j]$;    /* The elimination step. */
        end for
    end for
end for
```

Identify the parallelism available within the above `CHOLESKY` algorithm and develop a parallel formulation for multi-core CPUs using both pthreads and OpenMP. Before tackling this problem, I suggest that you work out the above algorithm on a small matrix to identify potential sources of parallelism. The program given to you accepts no arguments. The CPU computes the reference solution which is checked for correctness: if U is the upper-triangular matrix obtained by the implementation, then U^TU should be equal to A. Edit the `chol_using_pthreads()` and `chol_using_openmp()` functions in `chol.c` to complete the multi-core functionality. Do not change the source code elsewhere (except for adding timing-related code). The source files for this question are available in a zip file called `problem_1.zip`.

E-mail me all of the files needed to run your code as a single zip file called `solution_1.zip`. The code must compile and run on the xunil cluster.

This question will be graded as follows:

- **(25 points)** Multi-core version of the Cholesky decomposition using pthreads. I will focus on both the correctness of the operation as well as the speedup achieved over the single-threaded version. Ensure that your multi-threaded implementations are correct using small matrix sizes before obtaining the timing results for larger matrices.
- **(25 points)** Multi-core version of the Cholesky decomposition using OpenMP. I will focus on both the correctness of the operation as well as the speedup achieved over the single-threaded version.

Provide a two/three page report describing how you designed your program (use code or pseudocode if that helps the discussion) and the amount of speedup obtained for 2, 4, 8, and 16 threads over the serial version, for matrices of various sizes, including 2048×2048 elements.

2. Consider the Gauss-Seidel equation solver discussed within the lecture notes on how to write parallel programs (see the file called `parallelization_process.pdf` on webCT).

Recall that the order in which the grid points are updated in the sequential algorithm is not fundamental to the Gauss-Seidel solution method; it is simply one possible ordering that is convenient to program sequentially. Since the Gauss-Seidel method is not an exact solution method but rather iterates until convergence, we can update the grid points in a different order as long as we use updated values for grid points frequently enough, a technique called the *Jacobi method* where we don't use updated values from the current iteration for any grid points but always use the values as they were at the end of the previous iteration. Using the sequential program as a starting point, develop parallel versions of the Jacobi method using the following decomposition strategies.

- **(25 points)** *Red-black decomposition.* Implement a pthread program using *red-black ordering* where grid points are decomposed into alternating red points and black points so that no red point is adjacent to a black point, or vice versa. Since each point reads only its four nearest neighbors, to compute a given red point, we do not need the updated value of any other red point; we need only the updated values of the black points above it and to its left and vice versa for computing black points.
- **(25 points)** *Element-based decomposition.* Implement a pthread program for the element-based decomposition method. Do not create one thread for each element. Rather, create just a few threads (for example, 2, 4, 18, or 16) and have each thread process multiple elements.

The program provided to you accepts no arguments. It creates a randomly initialized grid of 8192×8192 elements and applies the update rule to each element within the grid until the specified convergence criteria is satisfied. The solution provided by the pthread implementations are compared to that generated by the reference code. The source files for this question are available in a zip file called `problem_2.zip`.

Answer the following questions.

- **E**dit the `compute_using_pthread_red_black()` and `compute_using_pthread_jacobi()` functions in the file `solver.c` to complete the functionality of the equation solver using pthreads. Do not change the source code elsewhere (except for adding timing-related code). The size of the grid is guaranteed to be 8192×8192 elements.
- E-mail me all of the files needed to run your code as a single zip file called `solution_2.zip`. The code must compile and run on the xunil cluster. Also, provide a two/three page report describing: (1) the design of your multi-threaded implementations (use code or pseudocode to clarify the discussion); (2) the speedup obtained over the serial version for 4, 8, and 16 threads.