Introduction to Parallel Computer Architecture
Midterm Exam

Instructor: Prof. Naga Kandasamy
ECE Department, Drexel University

November 1, 2015

The exam is due November 8, 2015, by 11:59 pm. It comprises of two questions. You may work on this exam in a team of up to two people.

1. **(25 points)** This problem asks you to develop multi-core implementations of the *Cholesky decomposition* method. A brief description 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 $x^{\mathsf{T}}Ax > 0$ 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 $U^{\mathsf{T}}U = A$. So, to solve $Ax = b$, one solves first $U^{\mathsf{T}}y = b$ 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 $A = U^T U$. So,

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

$$= \begin{pmatrix} u_{11}^2 & u_{11}u_{12} & u_{11}u_{13} & u_{11}u_{14} \\ u_{11}u_{12} & u_{12}^2 + u_{22}^2 & u_{12}u_{13} + u_{22}u_{23} & u_{12}u_{14} + u_{22}u_{24} \\ u_{11}u_{13} & u_{12}u_{13} + u_{22}u_{23} & u_{13}^2 + u_{23}^2 + u_{33}^2 & u_{13}u_{14} + u_{23}u_{24} + u_{33}u_{34} \\ u_{11}u_{14} & u_{12}u_{14} + u_{22}u_{24} & u_{13}u_{14} + u_{23}u_{24} + u_{33}u_{34} & u_{14}^2 + u_{24}^2 + u_{34}^2 + u_{44}^2 \end{pmatrix}$$

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

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

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

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: procedure CHOLESKY(A)
 2: int i, j, k;
 3: for k := 0 to n − 1 do
 4:     A[k, k] := √A[k, k];      /* Obtain the square root of the diagonal element. */
 5:     for j := k + 1 to n − 1 do
 6:         A[k, j] := A[k, j]/A[k, k];      /* The division step. */
 7:     end for
 8:     for i := k + 1 to n − 1 do
 9:         for j := i to n − 1 do
10:             A[i, j] := A[i, j] - A[k, i] × A[k, j];      /* The elimination step. */
11:         end for
12:     end for
13: 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 $U^T U$ 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 `cholesky.zip`. Upload all of the files needed to run your code as a single zip file on the BBLearn site. Note that the code must compile and run on the `xunil-04` machine.

This question will be graded as follows:

- **(15 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.

- **(10 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.

Include a brief report in your zip file describing how you designed your program, using code or pseudocode to help the discussion, and the amount of speedup obtained for 2, 4, 8, and 16 threads over the serial version, for matrices of the following sizes: 512×512, 1024×1024, and 2048×2048 elements.

2. **(15 points)** Given a function $f(x)$ and end points a and b, where $a < b$, we wish to estimate the area under this curve; that is, we wish to determine $\int_a^b f(x)\,\mathrm{d}x$.

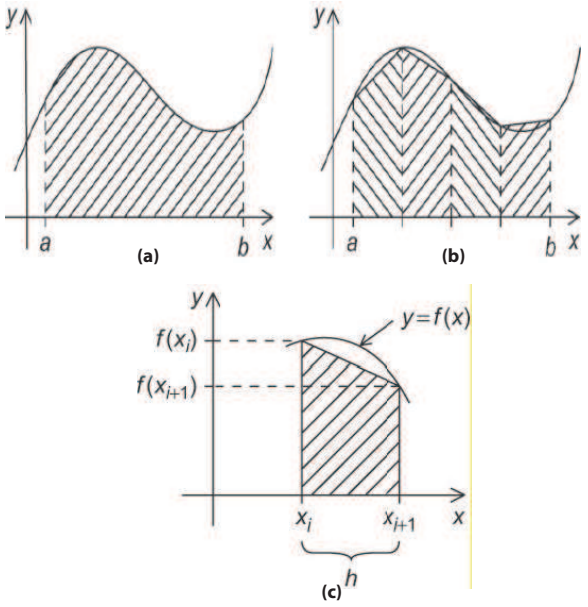

Figure 1: Illustration of the trapezoidal rule: (a) area to be estimated; (b) approximate area using trapezoids; and (c) area under one trapezoid.

The area between the graph of $f(x)$, the vertical lines $x = a$ and $x = b$, and the x-axis can be estimated as shown in Fig. 1 (b) by dividing the interval $[a, b]$ into n subintervals and approximating the area over each subinterval by the area of a trapezoid. Fig. 1(c) shows one such trapezoid where the base of the trapezoid is the subinterval, its vertical sides are the vertical lines through the endpoints of the subinterval, and the fourth side is the secant line joining the points where the vertical lines cross the graph. If the endpoints of the subinterval are x_i and x_{i+1}, then the length of the subinterval is $h = x_{i+1} - x_i$, and if the lengths of the two vertical segments are $f(x_i)$ and $f(x_{i+1})$, then the area of a single trapezoid is $\frac{h}{2}[f(x_i) + f(x_{i+1})]$. If each subinterval has the same length then $h = (b - a)/n$. Also, if we call the leftmost endpoint x_0 and the rightmost endpoint x_n, we have

$$x_0 = a, x_1 = a + h, x_2 = a + 2h, \ldots, x_{n-1} = a + (n - 1)h, x_n = b,$$

and our approximation of the total area under the curve will be

$$\int_a^b f(x)\,\mathrm{d}x = h[f(x_0)/2 + f(x_1) + f(x_2) + \cdots + f(x_{n-1}) + f(x_n)/2].$$

Thus, the pseudo-code for a serial algorithm is provided in the function TRAP.

The program provided to you accepts no arguments. The values for a, b, and n are defined within the program and so is the function $f(x)$ (within the file `trap.c`). The reference solution which will be compared with your pthread-based output. Edit the `compute_using_pthreads()` function within the file `trap.c` to complete the functionality of the trapezoidal rule on a multi-core CPU. The source files for this question are available in a zip file called `trap.zip`.

```
1: procedure TRAP(a, b, n)
2: h := (b − a)/n;
3: sum := (f(a) + f(b))/2.0;
4: for i := 1 to n − 1 step 1 do
5:     x_i := a + i × h;
6:     sum := sum + f(x_i);
7: end for
8: sum := h × sum;
```

Upload all of the files needed to run your code as a single zip file on BBLearn. Also, provide a brief write-up describing: (1) the design of your multi-threaded implementation (provide code or pseudocode to clarify the discussion); and (2) the speedup achieved over the serial version for 2, 4, 8, and 16 threads.