Introduction to Parallel Computer Architecture
Gaussian Elimination using OpenMP

Instructor: Prof. Naga Kandasamy, ECE Department, Drexel University

October 24, 2015

The assignment is due on November 1, 2015. You may work on this assignment in a team of up to two people.

Consider the problem of solving a system of linear equations of the form

$$\begin{array}{ccccc}
a_{0,0}x_0 & + a_{0,1}x_1 & + \cdots & + a_{0,n-1}x_{n-1} & = b_0, \\
a_{1,0}x_0 & + a_{1,1}x_1 & + \cdots & + a_{1,n-1}x_{n-1} & = b_1, \\
. & . & & . & . \\
. & . & & . & . \\
a_{n-1,0}x_0 & + a_{n-1,1}x_1 & + \cdots & + a_{n-1,n-1}x_{n-1} & = b_{n-1}.
\end{array}$$

In matrix notation, the above system is written as $Ax = b$ where A is a dense $n \times n$ matrix of coefficients such that $A[i,j] = a_{i,j}$, b is an $n \times 1$ vector $[b_0, b_1, \ldots, b_{n-1}]^T$, and x is the desired solution vector $[x_0, x_1, \ldots, x_{n-1}]^T$. From here on, we will denote the matrix elements $a_{i,j}$ and x_i by $A[i,j]$ and $x[i]$, respectively. A system of equations $Ax = b$ is usually solved in two stages. First, through a set of algebraic manipulations, the original system of equations is reduced to an upper triangular system of the form

$$\begin{array}{cccccc}
x_0 & + u_{0,1}x_1 & + u_{0,2}x_2 & + \cdots & + u_{0,n-1}x_{n-1} & = y_0, \\
 & x_1 & + u_{1,2}x_2 & + \cdots & + u_{1,n-1}x_{n-1} & = y_1, \\
 & & & & . & . \\
 & & & & . & . \\
 & & & & x_{n-1} & = y_{n-1}.
\end{array}$$

We write the above system as $Ux = y$, where U is a unit upper-triangular matrix, that is, one where the subdiagonal entries are zero and all principal diagonal entries are equal to one. More formally, $U[i,j] = 0$ if $i > j$, otherwise $U[i,j] = u_{i,j}$, and furthermore, $U[i,i] = 1$ for $0 \le i < n$. In the second stage of solving a system of linear equations, the upper-triangular system is solved for the variables in reverse order, from $x[n-1]$ to $x[0]$ using a procedure called back-substitution.

```
1: procedure GAUSS_ELIMINATE(A, b, y)
2: int i, j, k;
3: for k := 0 to n − 1 do
4:     for j := k + 1 to n − 1 do
5:         A[k, j] := A[k, j]/A[k, k];     /* Division step. */
6:     end for
7:     y[k] := b[k]/A[k, k];
8:     A[k, k] := 1;
9:     for i := k + 1 to n − 1 do
10:        for j := k + 1 to n − 1 do
11:            A[i, j] := A[i, j] - A[i, k] × A[k, j];     /* Elimination step. */
12:        end for
13:        b[i] := b[i] − A[i, k] × y[k];
14:        A[i, k] := 0;
15:    end for
16: end for
```

A serial implementation of a simple Gaussian elimination algorithm is shown above. The algorithm converts the system of linear equations $Ax = b$ into a unit upper-triangular system $Ux = y$. We assume that the matrix U shares storage with A and overwrites the upper-triangular portion of A. So, the element $A[k, j]$ computed in line 5 of the code is actually $U[k, j]$. Similarly, the element $A[k, k]$ that is equated to 1 in line 8 is $U[k, k]$. Also, our program assumes that $A[k, k] \neq 0$ when it is used as a divisor in lines 5 and 7. So, our implementation is numerically unstable, though it should not be a concern for this assignment. For k ranging from 0 to $n - 1$, the Gaussian elimination procedure systematically eliminates the variable $x[k]$ from equations $k + 1$ to $n - 1$ so that the matrix of coefficients becomes upper-triangular. In the $k^{\texttt{th}}$ iteration of the outer loop (line 3), an appropriate multiple of the $k^{\texttt{th}}$ equation is subtracted from each of the equations $k + 1$ to $n - 1$.

This problem asks you to develop a parallel formulation of `GAUSS_ELIMINATE` using OpenMP. The program given to you accepts no arguments. The CPU computes the reference (or single threaded) solution which is compared with the result provided by the multi-threaded implementation. If the solutions match within the specified tolerance, the application will print out "Test PASSED" to the screen before exiting. Edit the `gauss_eliminate_using_openmp()` function in `gauss_eliminate.c` to complete the functionality of Gaussian elimination using OpenMP. Build the code as follows:

Submit all of the files needed to run your code as a single zip file via BBLearn. Also include a brief report describing how you designed your multi-threaded code, using code or pseudocode to help the discussion, and the amount of speedup obtained over the serial version for 2, 4, 8, and 16 threads, for the following matrix sizes: 1024×1024, 2048×2048, 4096×4096, and 8192×8192.