EECS 358 – Introduction to Parallel Computing
Homework 1 (100 points)

EECS 358 Spring 2016
G. Memik April 21, 2016

DUE: 11:59pm, Thursday, May 5, 2016

You can work either alone or form a group of 2.

Problem 1 [20 pts.]

Consider the shared memory parallel matrix vector multiplication algorithm described in the class notes, slides 6.15 – 6.19. Using the speedup and scalability analysis techniques similar to those employed in the class notes (slides 2.29 – 2.31), answer the following.

a) [5 pts.] Analyze the running time of the serial algorithm as a function of the matrix mensions n and m. You may assume all operations take unit time. (More appropriate runtime other techniques will be presented later in the course – "big O" notation.)

b) [5 pts.] Analyze the running time of the parallel algorithm as a function of n, m, and p. You may assume that n and m are evenly divisible by p.

c) [5 pts.] Obtain expressions for speedup, S_p, and the Amdahl's fraction alpha.

d) [5 pts.] Determine if the algorithm is effective (Slide 2.15).

Problem 2 [30 pts.]

This problem is meant to familiarize you with using the shared memory machine. You do not need to write any code.

a) [0 pts.] Login to the 358smp machine using either rlogin (if you are in Wilkinson) or ssh (if you are not)

```
% rlogin 358smp.eecs.northwestern.edu
```

or

```
% ssh 358smp.eecs.northwestern.edu
```

Optionally, you can enable xwindows. If connecting from a Wilkinson machine:

```
% setenv DISPLAY  <local_machine>:0.0
```

Note that in your local machine you need to give the display permission (if you are using ssh, this may not work)

```
> xhost 358smp.eecs.northwestern.edu
```

b) [20 pts.] Copy the tar file hw1.tar to your own directory

```
% cp ~memik/hw1.tar ~yourloginname/
```

Untar the files:

```
% cd ~yourloginname
% tar –xvf hw1.tar
% cd hw1
```

There are several example applications in the hw1 directory. The pi.c contains the serial program for computing the value of pi. You can compile this for the origin using

```
% cc –o pi pi.c
```

Then, execute and time it in a serial mode as:

```
% time ./pi
```

This will process the output of pi and the timings for running the program on one processor. See the file "pi.c" for details of which times correspond to what measure.

REPORT THE RUNTIME

Next, look at the file "pi2.c", which is a parallel version of the pi program using explicit parallel programming (parallel thread) using functions for forking, locking, etc. (the pthread implementations are run on this application and each such instruction is followed by the IRIX syntax).

You can compile the program using

```
% g++ -o pi2 pi2.c -lpthread
```

Then, execute and time it in parallel mode as:

```
% time ./pi2
```

REPORT THE RUNTIMES FOR 1, 4, and 8 processors.

Next, look at the file "pi1.c", which is a parallel version of the pi program using implicit parallel programming using compiler directives such as "#pragma pfor", etc.

You can compile the program using

```
% g++ -o pi1 pi1.c -fopenmp
```

Then, execute and time it in parallel mode as:

```
% time ./pi1
```

REPORT THE RUNTIMES FOR 1, 4, and 8 processors and for dynamic and static scheduling techniques (the scheduling technique is enforced by the directive preceding the main loop).

c) [10 pts.] Next, look at the file "multdot.c". This file contains a parallel version of matrix vector multiplication using the dot product method using parallelization directives. Compile the program using

```
% g++ –o multdot multdot.c -fopenmp
```

Then, execute and time it in parallel mode as:

```
% time ./multdot
```

REPORT THE RUNTIMES FOR 1, 4, and 8 processors.

Problem 3 [50 pts.]

In this problem, you are asked to write a parallel algorithm for solving a set of dense linear equations of the form A*x = b, where A is an N x N matrix and x and b are column vectors. You will use Gaussian elimination without pivoting. The algorithm has two parts:

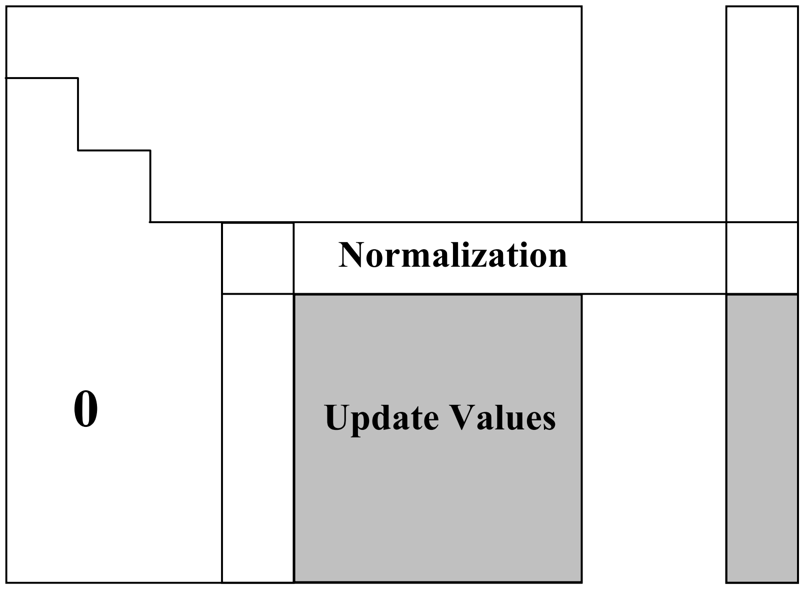
(a) Gaussian Elimination: The original system of equations is reduced to an upper triangular form

$$U x = y$$

where U is a matrix of size N x N in which all elements below the diagonal are zero, which are modified values of the A matrix. In addition, the diagonal elements have the value 1. The column vector y is the modified version of the b vector when you do the updates on the matrix and the vector in the Gaussian elimination stage.

(b) Back substitution: The new system of equations is solved to obtain the values of x.

The Gaussian elimination stage of the algorithm comprises N-1 steps. In the algorithm, the ith step eliminates nonzero subdiagonal elements in column i by subtracting the ith row from row j in the range [i+1,n], in each case scaling the ith row by the factor Aji / Aii sa as to make the element Aji zero. See the figure below for illustration:



Hence, the algorithm sweeps down the matrix from the top corner to the bottom right corner, leaving subdiagonal elements behind it.

The serial code for the algorithm is provided in the same hw1 directory for the previous questions. The file name is gauss.c

In the parallel algorithm, you **must** use dynamic scheduling. The whole point of parallel programming is performance, so you will be graded partially on the efficiency of your algorithm.

Suggestions:

- Forking processes is very expensive. Your program does not need to fork more than once.
- Each process should grab tasks until all tasks have been completed. Tasks of O(n) operations would be considered sufficiently large to hide the cost of obtaining a task, so a logical breakdown of tasks would be to let each task be responsible for the creation of one zero element in the lower diagonal of A.
- You may observe a slowdown from 1 processor to 2 because the locking overhead is minimal with 1 processor.
- Consider carefully the data dependencies in Gaussian elimination and the order in which tasks may be distributed.
- Gaussian elimination involves $O(n^3)$ operations. The back substitution requires $O(n^2)$ operations, so you are not expected to parallelize back substitution.

- The algorithm should scale, assuming n is much larger than the number of processors.
- The machine will likely be overloaded the night(s) before the assignment is due. ***Do the assignment early***.
- ***Send specific questions to the TA and the instructor.***

a) [20 pts.] Write a shared memory parallel algorithm using explicit parallel programming constructs (pthread). Begin with the provided serial code “gauss.c”. Compile with

```
% g++ -o gauss gauss.c -lpthread
```

To test your program, use a command like (1024x1024 matrix, 4 processors, seed=5555):

```
./gauss 1024 4 5555
```

b) [20 pts.] Beginning with the same serial code, write a shared memory parallel algorithm using the implicit parallel programming directives ($DOACROSS or #pragma parallel, etc.). Compile with

```
% g++ -o gauss gauss.c -fopenmp
```

c) [10 pts.] Evaluate the performance of both algorithms on a given matrix size (5000x5000). Hand in the timing results of your code for 1, 4, and 8 processors.

In addition to a report describing the timing results, please submit your codes for Gaussian elimination (modified gauss.c for section a and b) using Canvas. There should be clear comments at the beginning of the code explaining your algorithm.