Assignment1: Due on Jan 24

Grading Criteria:

- 10% Program correctness.
- 40% Program clarity, elegance and parallelization. The programs should be well-documented internally with comments.
- 10% Program scalability and performance.
- 40% Writeup. Your grade on the writeup includes the quality of the writing (grammar, sentence structure), whether all of the required elements are included, and the clarity of your explanations.

Moodle submission link will be up next week. We will strictly check for plagiarism against code available online or within your own class. Offenders will get a fail grade in the course. OpemMP hello world at ubuntu-openmp. More resources will be made online after Jan 7 lecture on OpenMP.

In this assignment, you will develop two parallel implementations of LU decomposition that use Gaussian elimination to factor a dense *N* x *N* matrix into an upper-triangular one and a lower-triangular one. In matrix computations, *pivoting* involves finding the largest magnitude value in a row, column, or both and then interchanging rows and/or columns in the matrix for the next step in the algorithm. The purpose of pivoting is to reduce round-off error, which enhances numerical stability. In your assignment, you will use row pivoting, a form of pivoting involves interchanging rows of a trailing submatrix based on the largest value in the current column. To perform LU decomposition with row pivoting, you will compute a permutation matrix **P** such that **PA** = **LU**. The permutation matrix keeps track of row exchanges performed.

Below is pseudocode for a sequential implementation of LU decomposition with row pivoting.

```
    inputs: a(n,n)
    outputs: π(n), l(n,n), and u(n,n)

    initialize π as a vector of length n
    initialize u as an n x n matrix with 0s below the diagonal
    initialize l as an n x n matrix with 1s on the diagonal and 0s above the
diagonal
    for i = 1 to n
      π[i] = i
    for k = 1 to n
      max = 0
      for i = k to n
        if max < |a(i,k)|
          max = |a(i,k)|
          k' = i
      if max == 0
        error (singular matrix)
      swap π[k] and π[k']
      swap a(k,:) and a(k',:)
      swap l(k,1:k-1) and l(k',1:k-1)
      u(k,k) = a(k,k)
      for i = k+1 to n
        l(i,k) = a(i,k)/u(k,k)
        u(k,i) = a(k,i)
      for i = k+1 to n
        for j = k+1 to n
```

```
        a(i,j) = a(i,j) - l(i,k)*u(k,j)

    Here, the vector π is a compact representation of a permutation matrix
p(n,n),
    which is very sparse. For the ith row of p, π(i) stores the column index of
    the sole position that contains a 1.
```

You will write two shared-memory parallel programs that perform LU decomposition using row pivoting. You will develop one solution using the Pthreads programming model and one using OpenMP.

Each LU decomposition implementation should accept two arguments: *n* - the size of a matrix, followed by `t` - the number of threads. Your programs will allocate an *n* x *n* matrix **a** of double precision (64-bit) floating point variables. You should initialize the matrix with uniform random numbers computed using a suitable random number generator, such as `drand48`, `drand48_r`, or the C++11 facilities for pseudo-random number generation. (Note: if you are generating random numbers in parallel, you will need to use a reentrant random number generator and seed the random number generator for each thread differently.) Apply LU decomposition with partial pivoting to factor the matrix into an upper-triangular one and a lower-triangular one.

To check your answer, compute the sum of Euclidean norms of the columns of the residual matrix (this sum is known as the L2,1 norm) computed as **PA-LU**. Print the value of the L2,1 norm of the residual. (It should be *very* small.)

The verification step need not be parallelized. Have your program time the LU decomposition phase by reading the real-time clock before and after and printing the difference.

The formal components of the assignment are listed below:

- Write a shared-memory parallel program that uses OpenMP to perform LU decomposition with partial pivoting.
- Write a shared-memory parallel program that uses Pthreads to perform LU decomposition with partial pivoting.
- Write a document that describes how your programs work. This document should *not* include your programs, though it may include figures containing pseudo-code that sketch the key elements of your parallelization strategy for each implementation. Explain how your program partitions the data, work and exploits parallelism. Justify your implementation choices. Explain how the parallel work is synchronized.

  Use problem size *n = 8000* to evaluate the performance of your implementations. If your sequential running time is too long for the interactive queue, you may base your timing measurements on n=7000. Prepare a table that includes your timing measurements for the LU decomposition phase of your implementations on 1, 2, 4, 8, and 16 threads. Graph of the parallel efficiency of your program executions. Plot a point for each of the executions. The x axis should show the number of processors. The Y axis should show your measured parallel efficiency for the execution. Construct your plot so that the X axis of the graph intersects the Y axis at Y=0.

In this assignment, reading and writing shared data will account for much of the execution cost. Accordingly, you should pay attention to how you lay out the data and how your parallelizations

interact with your data layout. You should consider whether you want to use a contiguous layout for the array, or whether you want to represent the array as a vector, of **n** pointers to n-element data vectors. You should explicitly consider how false sharing might arise and take appropriate steps to minimize its impact on performance.