## Authors:

Andrew Kizzier

Rachel Krohn

James Tillma

Jonathan Tomes

## Included:

Makefile

prog4\_shared.c

prog4\_dist.c

Prog4.pdf

## Compiling:

Simply use the included make file.

Or for the individual programs:

Shared Memory Version:

gcc –g –wall –fopenmp –o prog4\_shared prog4\_shared.c

Distributed Memory Version:

mpicc –g –wall –std=c99 –lm –o prog4\_dist prog4\_dist.c

## Usages:

Shared:

prog4\_shared n

Where n is the number of processes.

Distributed:

mpiexec –np n prog4\_dist

Where n is the number of processes.

NOTE: The number of rows in the matrix must be divisible by the number of processes.

## Problem:

Perform LU decomposition of a matrix of a set number of rows. And we must perform this using both a shared memory version and a distributed memory version. We are requiring that the matrix be a square matrix as many of the applications for the L and U matrices require that the matrix be square.

## Algorithms:

We will use Gaussian row elimination in order to create the upper right matrix that is U and the lower left matrix that is L. This requires that we do row swapping as we discover the need to. Upon eliminating a row we insert the scalars used into L. After we have completed all row eliminations we print each matrix (A, P, L, and U) to output. We then print to output that A = (LU)\* P’ to show that LU decomposition was successful.

## Timings:

# Foster’s Algorithm

## Partitioning & Communication

There are three main tasks for this problem of LU decomposition, calculating the U matrix, the L matrix, and the P matrix. Calculating the U matrix is divided into each row and determining how to eliminate the rows in the column below the row’s starting value. Calculating the L matrix, is just a matter of finding the scalar that was used in eliminating the rows in calculating the U matrix. The P matrix can be calculated as we swap rows as necessary, looking to put the max absolute value in the diagonal. So the main task we can put into parallel is calculating the scalars to calculate the U. The L matrix is created as we are calculating the U matrix. To calculate P we just need to swap the corresponding rows that we swapped in A as we were calculating U, however we only need to swap rows after we have started eliminating rows. With each row swap in U and P… we’ll have to swap rows below the diagonal in L.

What we have to communicate U, as that is both an input and what we are calculating. We must also communicate the L matrix, as well as the P matrix. Each process will only need 1 row however.

## Agglomeration

Each, row in U needs to eliminate the values in the column below its starting value. Since L is dependent on these scalar values, we need to calculate matrix U first. However, since we have the scalars at the time of calculating U, we can actually combine the two tasks. As we step down through the rows in the U matrix, we can insert the resulting scalars into matrix L. This means we will have to only communicate each row of U at each phase of the program. This also is true of creating the P matrix. So we can create the L and P matrices at the time of

## Mapping

We can map each row in U to a process. This would result in solving the U matrix, L matrix, and P matrix in Phases, one for each row. Where the process whose phase it is, broadcasts its row to the other processes and they calculate the scalar necessary to negate the appropriate values in their row, and place this value into their row in the L matrix. As they are going through… and its discovered that swaps are necessary… we need to swap the rows.