Project 1: Parallel LU Decomposition with Partial Pivoting

Scientific Supercomputing

Assigned: Oct 3, 2018
Due: Nov 6, 2018

1 Introduction

The objective of this project is to formulate and implement an algorithm to factor a (possibly permuted) matrix A into its lower and upper factors L and U to facilitate the solution of a system of linear equations. Large systems of equations (on the order of millions of equations) that typically arise from the discretization of a partial differential equation require us to provide LU factorization in a parallel context.

Recall that our LU factoring of $Ax = b$ results in $PA = LU$; which, if we substitute into $Ax = b$, results in $LUx = Pb$. We can then solve for x using a two-step process; let $y = Ux$, then we can solve for y using $Ly = Pb$ by forward substitution and then solve for x by backsubstitution in $Ux = y$.

Write and debug a parallel LU decomposition algorithm with partial pivoting using OpenMP with Fortran or C/C++. I must see some evidence of parallel efficiency in your results. In this project, for brevity, you will not be required to write a parallel forward/backsubstitution algorithm. However, 30 additional points will be awarded to those who do.

2 Block Cyclic Partitioning

To achieve reasonable load balancing, your algorithm must use block-cyclic partitioning. The partitioning itself can be described with two parameters: p, which is the number of threads utilized, and r, which is the recursion level. For example, if $p = 4$ and $r = 0$, the blocks of the matrix are assigned to threads as such:

$$\begin{bmatrix} 0 & 1 \\ 2 & 3 \end{bmatrix}$$

As another example, if $p = 4$ and $r = 2$, the blocks of the matrix may be assigned to the threads as:

$$\begin{bmatrix} 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 2 & 3 & 2 & 3 & 2 & 3 & 2 & 3 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 2 & 3 & 2 & 3 & 2 & 3 & 2 & 3 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 2 & 3 & 2 & 3 & 2 & 3 & 2 & 3 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 2 & 3 & 2 & 3 & 2 & 3 & 2 & 3 \end{bmatrix}$$

In this project, you may always assume that you are dealing with $n \times n$ square matrices, p is a perfect square, and that n is always evenly divisible by $\left(\sqrt{p}\right)^r \sqrt{p}$. You may distribute the blocks to the threads in any manner you see fit (as long as it is block cyclical), but you must clearly document the process in you discussion (a deliverable).

3 Implementation

Your program should be structured such that it reads the matrix dimensions and matrix data from a data file on one thread (the main execution thread or a HOST thread that you specify) only. The host thread is then responsible for distributing the workload to the rest of the available threads (as appropriate for block cyclic partitioning).

There are several "communication" steps in the parallel LU decomposition:

- communication of pivot values in order to find the pivot
- broadcast of the pivot row to the appropriate submatrix A_s
- broadcast of the multipliers to the submatrix A_s that requires them
- communication of the upper row entries to complete the elimination phase

Although this information is available to all threads, each step must completed accurately in a thread-safe manner.

There are multiple forms of the parallel LU decomposition algorithm, and you are free to explore and to implement the parallel LU decomposition of your choice. Recall that you may not use any pre-packaged code; all parts of this assignment that you submit must be your work. You must use either C/C++ or FORTRAN to complete this assignment.

4 Test Matrices

Perform the LU decomposition of the following matrices; report your P, L, and U for each:

$$A_1 = \begin{bmatrix} 1 & 2 & 2 \\ 4 & 4 & 2 \\ 4 & 6 & 4 \end{bmatrix} \tag{1}$$

$$A_2 = \begin{bmatrix} 4 & 6 & 7 & 8 \\ 1 & 3 & 2 & 4 \\ 6 & 5 & 7 & 8 \\ 9 & 8 & 7 & 6 \end{bmatrix} \tag{2}$$

$$A_3 = \begin{bmatrix} 4 & 6 & 7 & 8 \\ 1 & 0 & 2 & 4 \\ 6 & 5 & 7 & 8 \\ 9 & 8 & 7 & 6 \end{bmatrix} \tag{3}$$

5 Verification

1. Ensure that your parallel algorithm performs properly utilizing one thread $p = 1$ and $r = 0$ on matrices A_1, A_2, and A_3. You may utilize the results of Homework 2 in this respect.

2. Ensure that your parallel algorithm obtains the correct LU factors for A_2 and A_3, with $p = 1$,$r = 1$ and $p = 1$, $r = 2$.

3. Ensure that your parallel algorithm obtains the correct LU factors for A_2 and A_3, with $p = 4$, $r = 0$ and $p = 4$,$r = 1$.

4. A Hankel matrix is given by the following formula:

$$a_{ij} = 1/(2 * (n - i - j + 3/2)) \tag{4}$$

   where i and j are one-based indices (*C/C++ programmers pay attention*). For a Hankel matrix of size $n = 10$, report your LU factors for $p = 4$ and $r = 0$.

The results of the items above must be documented in your project; report the L and U factors. You may perform these tests on any machine of your choosing.

6 Performance

In this section, the Hankel matrix above will be used exclusively due its ease of construction. These computations must be performed on the Alabama Supercomputer.

- On a single thread with $r = 0$, compute and time the LU factorization of Hankel matrices with size $n = 256$, $n = 512$, $n = 1024$, $n = 2048$, and $n = 4096$. Noting that the LU decomposition is an $O(n^3)$ algorithm, determine the coefficients c_i that relates the cost of the algorithm (in seconds) to n: $T(n) = c_3n^3 + c_2n^2 + c_1n + c_0$ (Hint: linear least squares is useful here). Once the c_i are known, you have an expression for the predicted runtime for any matrix size n. Plot this estimated run time (as a function of n; I want to see a curve here) as well as the actual run times on a single graph. In addition, plot the same two quantities on a graph with a log-log scale. Discuss your results.

- On four thread ($p = 4$) run every possible combination of $n = 1024$, $n = 2048$, and $n = 4096$ with $r = 0$, $r = 1$, $r = 2$, $r = 3$ (this is twelve total cases). Plot all of the run times on the same graph. Discuss your results.

7 Administrative

There are *many* different implementation approaches in the parallel LU algorithm, and many choices for you to make for an *efficient* implementation. You are allowed discretion to make choices that make the implementation easier at the cost of some efficiency in this project; however, you must document these choices as you make them. So, one section of your project should discuss any built-in inefficiencies and the expected effect that they might have. Examining operation counts and communication costs specifically will greatly aid in this analysis.

Suggestion: use MATLAB or GNU octave (a MATLAB clone freely available on Linux platforms, among others) to help debug your code. However, MATLAB/OCTAVE results will not be accepted.

Submit at least the following for your project:

- presentation of the algorithm (section 2)
- source code
- discussions and results for the Verification and Performance sections.
- discussion of implementation choices and the impact on efficiency