Linear Algebra Optimization in C using OpenMPI

Introduction & Motivation

- Demand for faster and more efficient computing methods
- Traditional sequential computing: Tasks are executed one after another by a single processor. This method often falls short when tackling complex and large scale problems
- Parallel computing: Tasks are executed simultaneously across multiple processors, enabling execution of sub tasks at the same time.
- Open Message Passing Interface (OpenMPI) library is used in C to parallelize the four following functions: Matrix multiplication, REF, RREF, and LU Decomposition

Algorithms

Matrix multiplication is parallelized by dividing the work of the rows of the result matrix based on the number of processes. Assuming we have P processes, the number of rows per process is equal to the number of rows in A divided by the number of processes minus 1. Each worker node multiplies its chunk of A with the entire matrix B. Then results are gathered back together with the resulting matrix.

REF is parallelized by distributing the matrix rows across worker processes. At each pivot step, the pivot row is broadcast to all workers, who update their local rows in parallel to eliminate entries below the pivot. The master node gathers the updated rows to form the upper triangular matrix.

Algorithms

RREF is parallelized similarly to REF, with rows distributed among worker processes. After completing the forward elimination to form an upper triangular matrix, backward elimination zeroes out the entries above each pivot. At each step, the relevant pivot row is broadcast so that each worker can eliminate the corresponding entries in their local rows. Each pivot row is scaled to make the pivot element 1. The master node collects the fully reduced rows from each process.

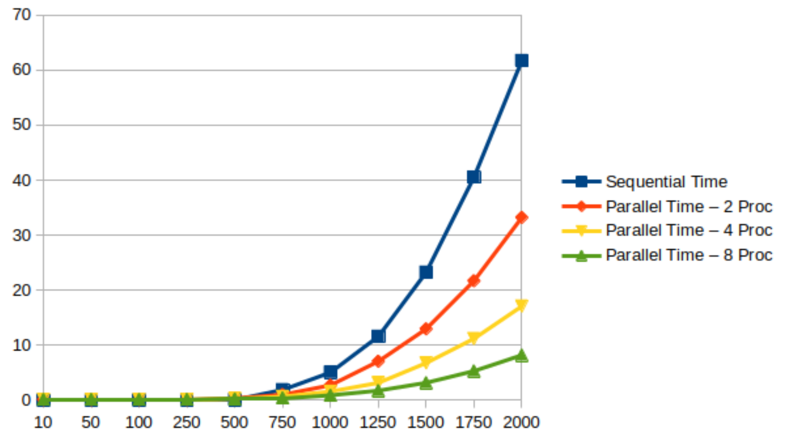
LU Decomposition is parallelized by distributing the rows of the input matrix among worker processes, as with REF. At each pivot step, the pivot row is broadcast to all workers to update the rows below, forming the upper triangular matrix U. At the same time, the multipliers used during elimination are stored in place to build the lower triangular matrix L. This update is done locally on each worker's portion of the matrix. After all the steps, the master node gathers the final rows of L and U to reconstruct the full decomposition.

Results

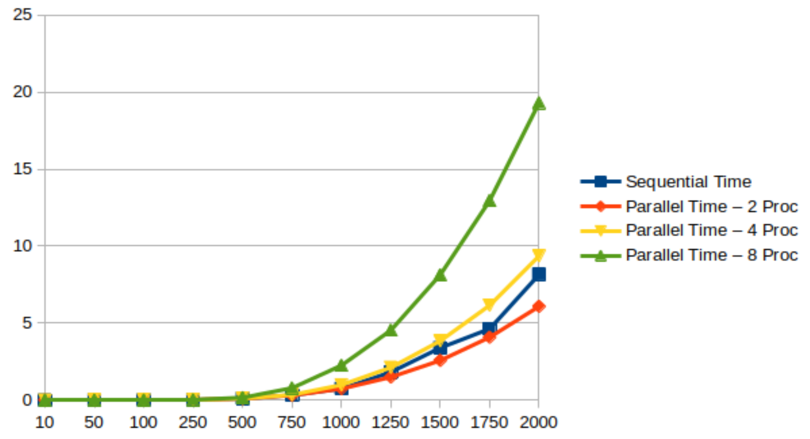
Timing benchmarks on square matrices from 10x10 to 2,000x2,000 showed that parallel matrix multiplication, REF, and RREF significantly outperformed their sequential versions as size increased. For instance, REF on a 2,000x2,000 matrix dropped from 9.5 seconds to 2.0 seconds with 8 processes. Matrix multiplication and RREF saw similar gains, though some slowdown occurred from pivot synchronization. In contrast, LU decomposition was much slower in parallel due to heavy communication overhead from frequent MPI_Bcast and MPI_Barrier calls.

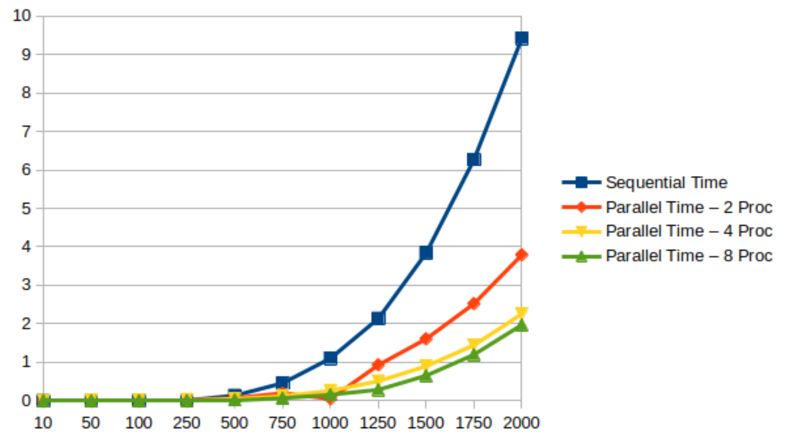
Results

Matrix Multiplication

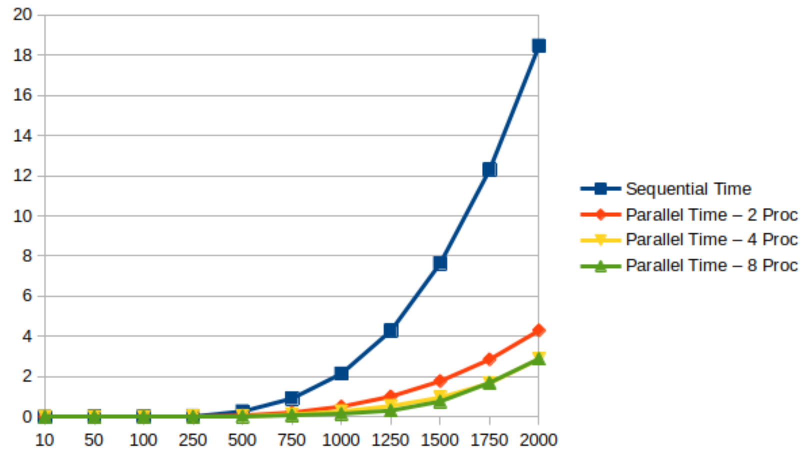


LU Decomposition





REF



RREF

Conclusion

Overall, the parallel implementations of matrix multiplication, REF, and RREF demonstrated clear performance advantages on large matrices, validating the effectiveness of OpenMPI for computational speedups. However, LU decomposition proved less efficient in parallel due to communication overhead, highlighting that not all algorithms have the ability to benefit equally from multiprocessing.