CISC 360 Project

Ryan McKenna, Matthew Paul, Niko Gerassimakis,
Neil Duffy, James Kerrigan

May 12, 2015

1 Introduction

LU Factorization is the most common technique used to solve systems of linear equations. It is most useful when solving a dense linear system and is only appropriate when the system is square. It works by decomposing a square matrix A into an lower triangle matrix, L and an upper triangular matrix U such that

$$A = LU$$

1.1 Impact

Triangular matrices have a number of nice properties that make them easier to work with. For example, if T is a triangular matrix, then you can solve the equation

$$Tx = b$$

in $O(n^2)$ time, as opposed to $O(n^3)$ for full matrices. Computationally, this means we can solve k equations of the form

$$Ax = b_i$$

for $1 \leq i \leq k$ in $O(n^3 + kn^2)$ as opposed to $O(kn^3)$.

1.2 Use Cases

Dense linear algebra is very important for mathematicians, scientists, and engineers alike. Linear algebra comes up in so many situations:

- Physics
- Partial differential equations
- Graph theory
- Statistics / Curve Fitting
- Sports Ranking

Solving linear systems is a key component of linear algebra, and LU factorization is the best known way to solve these systems. Having a highly optimized LU Factorization algorithm gives you the ability to (1) solve systems faster, and (2) solve bigger systems.

2 Background

2.1 Triangular Matrices

2.2 Algorithm Description

3 Existing Work

4 Approach

We will start off with a simple implementation of LU factorization that is not optimized for any particular architecture which will serve as a starting point to measure performance improvements.

4.1 Architecture

Our initial optimizations target a multicore CPU architecture, such as the current Intel Haswell series i7 4970K processor. By utilizing OpenMP, our code will be accessible on a wide variety of platforms. This project may also utilize CUDA in order to implement parallel optimizations on the GPU. GPU parallelization is advantageous as it allows us to take advantage of a large number of cores + threads. Code optimizations will target the most recent nvidia GTX 970 GPU.

4.2 Optimizations

TO-DO: cache, loop unrolling, openMP, vector operations, cuda

5 Results

5.1 Sequential Optimizations

5.2 Parallel Optimizations

6 Conclusion