Robert Crimi[1]

Thomas Nelson[1], Elizabeth Jessup[1], Jeremy Siek[2]

University of Colorado Boulder[1], Indiana University[2]

SC13 Poster

Optimizing Built-To-Order BLAS while Considering Matrix Order

Many applications of linear algebra use sequences of Basic Linear Algebra Subprograms (BLAS). Computer scientists apply tuning techniques to improve data locality to create efficient implementations of those routines enabling scientists to build high-performance software at reduced cost. However, because the BLAS are individually optimized, users may not see the performance they desire when using a sequence of BLAS. We are developing the Build To Order (BTO) compiler for tuning such sequences. One challenge of optimizing linear algebra is that each matrix order may require a different tuning technique. In this poster, we present an algorithm to efficiently choose code variants for a range of matrix orders.

BTO allows users to write in familiar MATLAB code and get back highly optimized C code. BTO can form any combination of BLAS along with performing loop fusions, cache tiling, and shared-memory parallelism. Because these techniques take advantage of the memory hierarchy, they can be very difficult to write by hand. While BTO has been shown to provide huge performance improvements, it lacks the capability to optimize code when the matrix order varies. For example, in matrix bidiagonalization, the order of the matrix being worked on constantly decreases. If a user tries to run a bidiagonalization kernel on BTO, their code is only optimized for one matrix order, and therefore they do not see the performance that they sought. For this reason, we were motivated to find the link between matrix order and tuning techniques. The result is the Matrix Order Partitioning Algorithm that is the subject of this poster.

By collecting and analyzing data on what BTO chose to do in order to optimize different matrix orders and MATLAB kernels, we arrived at an algorithm. We concluded that there are matrix order ranges in which the same code variant is the fastest. We needed to make decisions on how to split up ranges to be most efficient while keeping size of the output code small. If the size of the code is too large, it is difficult for users to understand what it is doing and they may even see performance loss. For example, if BTO chooses to switch optimizations on every matrix order, then there are a lot of branches in the code, most likely decreasing performance.

After much consideration, we took the top three performing code variants for each matrix order and found ranges in which a certain code variant is retained in the top three. Using this algorithm proved to be very efficient. Comparing the performance of the best BTO could do over the matrix order range with how our algorithm performed; we saw that our algorithm was well within 1% of the best.