Creating High-Performance Linear Algebra Using Lighthouse and Build-to-Order BLAS

Jeffrey Cook*, Elizabeth Jessup*, Sa-Lin Cheng Bernstein†, and Boyana Norris‡
*Department of Computer Science, University of Colorado, Boulder, CO, USA
†Computation Institute, University of Chicago and Argonne National Laboratory, Chicago, IL, USA
‡Department of Computer and Information Science, University of Oregon, Eugene, OR, USA

I. Introduction

Efficient linear algebra software is essential in high-performance computing, but writing it is a time-consuming task requiring expertise in multiple domains. Typically Basic Linear Algebra Subprograms (BLAS), containing the most common vector and matrix operations, is used either directly as the basis of linear algebra code, or as the substrate of more extensive packages such as LAPACK. Although solver collections like LAPACK give the programmer a more powerful API than BLAS, they still have the problems of a steep learning curve and of performance limited by that of the BLAS implementation. To take on the dual challenge of usability and optimization, we present Lighthouse [2], a web-based taxonomy of dense and sparse linear system and eigenvalue solvers from LAPACK, PETSc, and SLEPc. Lighthouse's guided search system has a user-centered design that assists scientists in finding the most appropriate routines available for their applications as well as the pertinent documentation and code templates. Where custom linear algebra kernels are required, its new interface to the Build-to-Order BLAS compiler (BTO) makes it significantly easier to create and use high-performance code.

II. The BTO Compiler

The BTO research compiler [1] converts MATLAB-style equations to C code. It uses a genetic algorithm to test numerous combinations of optimizations like loop fusion, threading, and cache tiling to reduce memory traffic and produce a machine-specific best-performing kernel. Operating on the user's whole routine together rather than building it out of BLAS calls is what distinguishes the BTO compiler. Since BLAS come optimized at the routine level, there is no higher-level optimization among sequentially composed calls; for example, the library may not take advantage of data reuse or perform possible loop fusions. Whereas this limits the optimizations that compilers can perform in BLAS-based programs, BTO searches for ways to improve overall code balance by producing a single-function routine that includes all steps of the kernel.

III. The BTO Web Client on Lighthouse

The Lighthouse site aims to make BTO more accessible than the original command-line program by adding adding a web client interface to remote instances of BTO test servers. On the site, a user enters the MATLAB-style equations for the kernel they would like transformed to C. Below this they may modify parameters for BTO, all of which have buttons for contextual help. Next, the user clicks a button to check the syntactic correctness of their equations. Then, the page asks the user to verify or modify variable types for the kernel, which have been parsed automatically. Finally the user submits the form which sends all inputs to a server process that interfaces with the actual BTO executable. When it finishes running, users see the resulting C code which is made available for download. In the poster, we include screenshots to illustrate the workflow advantages of using BTO this way over hand-coding BLAS calls.

IV. Benchmarking

In order to demonstrate the viability of codes generated by BTO, we present single-threaded run time data for several routines and compare them to equivalent BLAS calls. The first series of tests uses icc with MKL as the BLAS on the Intel Xeon x5660 processor; the second series uses gcc with ATLAS as the BLAS on the Intel Core i5 4670K.

V. Benchmarking Results

Routine	ATAX	BiCG	BiCGSTAB	GEMVER	HESSRED
icc/MKL	3.16	2.14	1.60	1.72	0.80
icc/BTO	3.35	2.45	2.22	2.14	2.02
gcc/ATLAS	1.25	1.25	1.29	8.64	3.91
gcc/BTO	2.60	2.72	3.34	2.89	2.20

Kernel run times in seconds for repeated executions.

VI. Conclusions

Lighthouse's BTO interface makes high-performance linear algebra code easy to create and to use. Benchmarks show that codes generated in minutes can have run times comparable to hand-coded BLAS calls, and in some cases out-pace them. Since architecture, compilers, libraries, and problem size and type all have a significant impact on application performance, it is difficult to draw definitive conclusions from this particular set of tests. The BTO compiler's inherent adaptation of kernels to the host system makes it an asset in overcoming these programming considerations, but does not replace well-designed software when absolute performance is desired. We emphasize BTO's strength in making fast code accessible to many with relatively little effort.

Acknowledgment

This work supported by NSF grant CCF-1219089.

REFERENCES

[1] R. Nair, S. Bernstein, E. Jessup and B. Norris. *Generating Customized Sparse Eigenvalue Solutions with Lighthouse*. Proceedings of the Ninth International Multi-Conference on Computing in the Global Information Technology June 22 - 17, 2014, Seville, Spain.

[2] B. Norris, S. Bernstein, R. Nair, E. Jessup. *Lighthouse: A User-Centered Web Service for Linear Algebra Software*. Elsevier Journal of Systems and Software (JSS): Special Issue on Software Engineering for Parallel Systems. To appear; arXiv preprint arXiv:1408.1363. 2014.