Study of Exploiting Coarse-Grained Parallelism in Block-Oriented Numerical Linear Algebra Routines

Gerson C. Kroiz[1]*, **Alexandre Bardakoff**[2], **Timothy Blattner**[2], and **Walid Keyrouz**[2]

[1] Department of Mathematics and Statistics, University of Maryland, Baltimore County, Baltimore, MD 21250, USA
[2] Information Tech. Lab., National Institue of Standards and Technology, 100 Bureau Drive Gaithersburg, MD 20899, USA

We have developed streaming implementations of two numerical linear algebra operations that further exploit the block decomposition strategies commonly used in these operations to obtain performance. The implementations formulate algorithms as data flow graphs and use coarse-grained parallelism to (1) emit a block in the result matrix as soon as it becomes available and (2) compute on multiple blocks in parallel. This streaming design benefits data flow graphs consisting of multiple linear algebra operations as it removes synchronization points between successive operations: a result block from an operation can be used immediately in an algorithm's successor operations without waiting for the full result from the first operation. Early comparisons with OpenBLAS functions on CPUs show comparable performance for computing with large dense matrices and an earliest arrival time of a result block that is up to 50x smaller than the time needed for a full result. More thorough studies can show the impact of such implementations on the performance of systems by chaining multiple linear algebra operations.

Keywords: Coarse-Grain Parallelism, Dataflow Graphs, Matrix Multiplication, LU Decomposition, Streaming Data

1 Introduction

Numerical linear algebra libraries are a collection of algorithms that are often used to study and compare performance on computing systems [1]. Algorithms within these libraries are highly optimized; this makes them difficult to further improve. Libraries that implement numerical linear algebra functions divide input matrices and vectors into sub-matrices and sub-vectors, respectively. The library function then uses the sub-pieces to take advantage of hardware parallelism in modern systems, vector instructions and multi-core CPUs, thereby improving performance. Applications that use linear algebra libraries operate in lock-step mode: they invoke a function and wait for its results. As such, an application that uses a sequence of such functions invokes them sequentially. One possible optimization for these types of applications is to stream partial-result blocks in and out of the functions; this will allow the next function in a sequence to begin much sooner. This paper briefly explores this streaming design and how it can influence the wait time on outputted data.

2 Algorithm Development

In this study, we implement block streaming for Matrix Multiplication (MM) [2, 3] and LU Decomposition with Partial Pivoting (LU) [4]. The challenge with expressing an algorithm using streaming operations is the need to represent the *state* throughout the algorithm. This state identifies when a partial result block can be emitted so it can be used in following operations. Our implementations use the data flow library, Hedgehog [5], which requires a developer to explicitly represent an algorithm's state and makes it easy to reason about this state. The implementations of both algorithms use the block streaming design pattern described in Section 1 and invoke BLAS functions from the OpenBLAS [6] library to operate on blocks.

As an example of a block streaming implementation, LU decomposition with partial pivoting consists of four operations in a loop: (1) Gaussian elimination of the diagonal block, (2) row swap, (3) factorization of the outer row and column, and (4) update of the matrix's remainder. Blocks can be streamed out of the algorithm after Gaussian elimination and factorization of the rows. In Hedgehog, we fully express the state of the blocks in order to identify the earliest moment a block can stream out of the algorithm.

3 Performance Study

A preliminary study is conducted to compare the end-to-end performances of the Hedgehog (v1.0) implementations and their OpenBLAS [6] counterparts. The evaluation machine has two Intel Xeon E5-2680 v4 CPUs @2.40 GHz (28 physical cores, 56 logical cores) with 512 GB of DDR4 memory. We ran computations on double precision matrices of size $32\,768 \times 32\,768$. Figure 1 shows the average performance over 10 trials of the Hedgehog (HH) implementation of Matrix Multiplication and LU Decomposition relative to the performance of the `dgemm` and `dgetrf` functions from OpenBLAS v0.3.6.

The results show that Hedgehog's `dgemm` function improves performance over the OpenBLAS implementation by a factor of 1.5x when using 2048×2048 and 4096×4096 blocks. In contrast, Hedgehog's `dgetrf` and its OpenBLAS counterpart

[*] Corresponding author: e-mail gkroiz1@umbc.edu

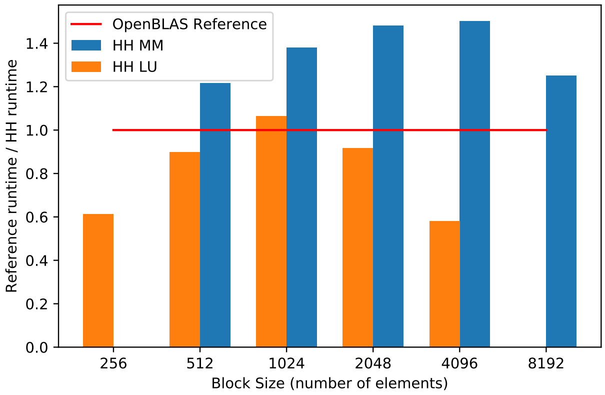

Fig. 1: Relative performance to OpenBLAS for LU and MM.

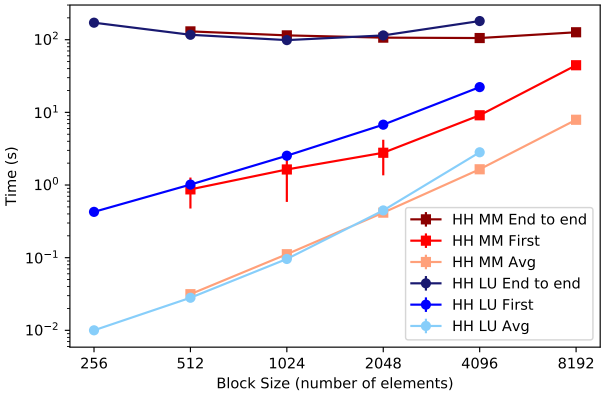

Fig. 2: Block production performance for MM and LU.

have similar performance; the Hedgehog function improves performance by 1.06x only when using 1024×1024 blocks. These results indicate that representing and maintaining state in the Hedgehog implementations do not have a negative impact on end-to-end performance. Further study needs to be conducted on smaller and larger matrix sizes.

Figure 2 illustrates the relationships between the block size and three time quantities that a developer uses to select a block size that optimizes performance. These quantities are: (1) release of the first result block (*first*), (2) average release of a result block (*avg*), and (3) full result. When using 2048×2048 blocks, the first-release times for the Hedgehog implementations of Matrix Multiplication (MM) and LU Decomposition (LU), were $2.78\,\mathrm{s}$ and $6.77\,\mathrm{s}$; these times compare favorably with full result times of $106.76\,\mathrm{s}$ and $114.57\,\mathrm{s}$ respectively; the corresponding average-release times were $0.42\,\mathrm{s}$ and $0.45\,\mathrm{s}$.

These results are promising for applications that chain multiple operations together. A *simplistic* analysis of an application, which composes Matrix Multiplication and LU Decomposition and then uses the latter's results, yield the following: (1) the application could potentially start emitting its results after $2.78\,\mathrm{s} + 6.77\,\mathrm{s} = 9.55\,\mathrm{s}$ instead of waiting for the full $221.33\,\mathrm{s}$; (2) the application can also start emitting additional result blocks at an average rate of $2 \times \max(0.42\,\mathrm{s}, 0.45\,\mathrm{s}) = 0.90\,\mathrm{s}$ assuming that hardware resources can support the parallelism in both operations. If verified by actual experiments and a detailed analysis, the streaming approach promises to yield times that are several orders of magnitude smaller than end-to-end times for either operation.

4 Conclusions

The results show that there are block configurations in which the Hedgehog implementations of Matrix Multiplication and LU Decomposition outperform their corresponding OpenBLAS functions, highlighting that the stream-based designs of these algorithms do not strain computational performance. Furthermore, these algorithms are capable of streaming out finalized blocks well before the matrix is fully computed. This feature will play a significant role in designing pipelines of algorithms in order to reduce the wait times between operations. For example, if the block size is small, then the rate in which blocks stream increases; however, the end-to-end performance of the operation could degrade. These dynamics of selecting the optimal block size of an operation and between operations is an interesting topic for future study.

Disclaimer

No approval or endorsement of any commercial product by the National Institute of Standards and Technology is intended or implied. Certain commercial software, products, and systems are identified in this report to facilitate better understanding. Such identification does not imply recommendations or endorsement by NIST, nor does it imply that the software and products identified are necessarily the best available for the purpose.

References

[1] J. J. Dongarra, Performance of various computers using standard linear equations software, Tech. Rep. CS - 89 - 85, Electrical Engineering and Computer Science Department, University of Tennessee Knoxville; Computer Science and Mathematics Division, Oak Ridge National Laboratory Oak Ridge; University of Manchester, June 15 2015, http://www.netlib.org/benchmark/performance.ps. Last access: 2020-02-13.

[2] T. Blattner, W. Keyrouz, S. S. Bhattacharyya, M. Halem, and M. Brady, Journal of Signal Processing Systems **89**(3), 457–467 (2017).

[3] G. H. Golub and C. F. van Loan, in: Matrix Computations, fourth edition (JHU Press, 2013), p. 26.

[4] J. Kurzak, P. Luszczek, M. Faverge, and J. Dongarra, IEEE Transactions on Parallel and Distributed Systems **24**(8), 1613–1621 (2013).

[5] A. Bardakoff, T. Blattner, and W. Keyrouz, Hedgehog (hh): A library to generate heterogeneous pipeline workflow systems, https://github.com/usnistgov/hedgehog, Last access: 2019-11-26.

[6] Z. Xianyi and M. Kroeker, OpenBLAS: An optimized BLAS library, http://www.openblas.net/, Last access: 2020-05-22.