Matrix factorization routines on heterogeneous architectures

Nikita Shustrov, Nadya Mozartova, Tamara Kashevarova, Konstantin Arturov
Intel Corporation
Nikita.A.Shustrov@Intel.com, Nadezhda.Mozartova@Intel.com, Tamara.P.Kashevarova@Intel.com,
Konstantin.I.Arturov@Intel.com

Abstract

In this work we consider a method for parallelizing matrix factorization algorithms on systems with Intel® Xeon Phi™ coprocessors. We provide performance results of matrix factorization routines implementing this approach and available in Intel® Math Kernel Library (Intel MKL) on the Intel® Xeon® processor line with Intel Xeon Phi coprocessors.

Summary

New heterogeneous systems consisting of a multicore CPU with coprocessors introduce new challenges to designing efficient parallel algorithms. Simultaneous usage of all computational resources of such systems for solving one large problem requires uneven distribution of data and computations that leads to more complex parallelization methods.

In this work we present a new parallelization method for matrix factorization that efficiently utilizes all computational resources of a heterogeneous system consisting of a multicore CPU with coprocessors. We will show how this method can be applied for parallelization of the key linear algebra factorization algorithms (QR, LU, and Cholesky) on systems with Intel Xeon Phi coprocessors.

Our matrix factorization method is based on the panel factorization approach [5]. The panel factorization approach has advantages over communication avoiding, tile methods [5], and their combination [8]:

- no additional computational cost;
- no additional memory consumption.

The panel factorization approach has the same computational cost and memory usage as classic LAPACK algorithms [4], which makes this approach preferable for systems with coprocessors. The implementation preserves the LAPACK standard interfaces and data layout. The algorithm can be applied to any matrices. The implementation of our method is DAG-based [6] and uses panel factorization kernels that were redesigned and rewritten for new Intel Xeon Phi products [7] [9].

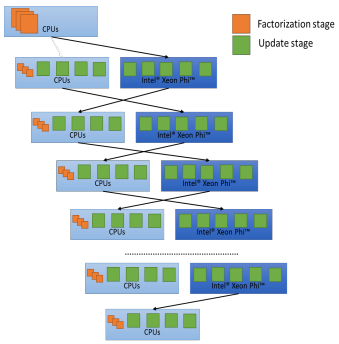

Figure 1: Algorithm represented as DAG

Figure 1 shows the algorithm represented as DAG. The algorithm implementation has the following features:

- At the beginning, CPUs produce a number of factorized panels and send to coprocessors as many panels as needed to maximize coprocessor utilization.
- Coprocessors perform ”update” stages in parallel.
- CPUs perform both ”factorization” stages and ”update” stages in parallel.
- To achieve the best load balance, a coprocessor may send a panel back to the CPU side on any process stage.

The proposed method provides a high degree of parallelism while minimizing synchronizations and communications. The algorithm enables adaptable workload distribution between CPUs and coprocessors to improve load balancing, namely:

- Adaptable data/task distribution on the fly between CPUs and coprocessors.
- No limit on number of coprocessors on heterogenous systems.
- Scalability. A system with CPUs and one coprocessor shows 3x performance improvement and a system with CPUs and two coprocessors shows 5x performance improvement.
- No algorithmic limitations on matrix sizes.

Our algorithm is implemented within the framework of Intel MKL [3] LU, QR and Cholesky factorization routines. The implemented routines detect the presence of Intel Xeon Phi coprocessors and automatically offload the computations that benefit from additional computational resources. This usage model hides the complexity of heterogeneous systems from the user, providing ease of use and the same API as usual Intel MKL routines.

This parallelization method can be effectively applied to other LAPACK [4] algorithms.

1. REFERENCES

[1] Netlib - A collection of mathematical software, papers, and databases. http://www.netlib.org.

[2] Jakub Kurzak, and Jack Dongarra. Implementing Linear Algebra Routines on Multi-Core Processors with Pipelining and a Look Ahead. UT-CS-06-581, September 2006. LAPACK Working note #178.

[3] Intel® Math Kernel Library, http://www.intel.com/software/products/mkl.

[4] LAPACK - Linear Algebra PACKage, http://www.netlib.org/lapack.

[5] Sergey V. Kuznetsov. An approach of the QR factorization for tall-and-skinny matrices on multicore platforms, 11th International conference, Proceedings of PARA 2012, Helsinki, Finland, LNCS, vol. 7782, Springer Verlag, pp. 235-249, 2013

[6] A. Kobotov, S. V. Kuznetsov. Efficient dynamic parallelization for the QR factorization. In Proceedings of PARA 2008: 9th International Workshop on State-of-the-Art in Scientific and Parallel Computing, 2008.

[7] Alexander Heinecke, Vaidyanathan Karthikeyan, Smelyanskiy Mikhail, Kobotov Alexander V, Dubtsov Roman S, Henry Greg, Shet Aniruddha G, Chrysos George, Dubey Pradeep. Design and Implementation of the Linpack Benchmark for Single and Multi-Node Systems Based on Intel® Xeon Phi™ Coprocessor, IEEE International Parallel & Distributed Processing Symposium (IPDPS), 2013.

[8] E. Agullo, C. Augonnet, J. Dongarra, M. Faverge, H. Ltaief, S. Thibault, and S. Tomov. 'QR Factorization on a Multicore Node Enhanced with Multiple GPU Accelerators', IPDPS 2011.

[9] Michael Deisher, Mikhail Smelyanskiy, Brian Nickerson, Victor W. Lee, Michael Chuvelev, Pradeep Dubey. Designing and Dynamically Load Balancing Hybrid LU for Multi/Many-core, International Supercomputer Conference, 2011.