Task-parallel algorithm for matrix factorizations

Tomohiro Suzuki
stomo@yamanashi.ac.jp
University of Yamanashi

Task-parallel algorithms have attracted attention as algorithms for highly parallel architectures in recent years. One of the reasons is that the block algorithm used for LAPACK, which is the de facto standard of numerical linear algebra library, cannot sufficiently utilize the recent highly parallel computing resources. The purpose of the task-parallel algorithm is to keep all computing resources running without stalling by executing a number of fine-grained tasks asynchronously while tracing data dependencies. We can easily describe task-parallel programs by using OpenMP `task` construct and `depend` clause. Besides, the `priority` clause introduced in OpenMP 4.5 allows more efficient task scheduling.

Among various scientific computations, matrix factorization is considered to be a well-matched algorithm with the task-parallel programming model. Also, since matrix factorization is a heavy process with $O(N^3)$ of computational complexity, there is a high demand for speeding up. In the block algorithm for matrix factorization, it is possible to generate the tasks that can be executed in parallel by dividing the trailing matrix into plural panels and performing subsequent matrix update for each panel (Fig. 1). Furthermore, a relatively deep *look-ahead* can be easily achieved by raising the priority of the panel decomposition task with the `priority` clause. However, the block algorithm can achieve little benefit from efficiency improvement by task parallelization. This algorithm lacks inherent parallelism.

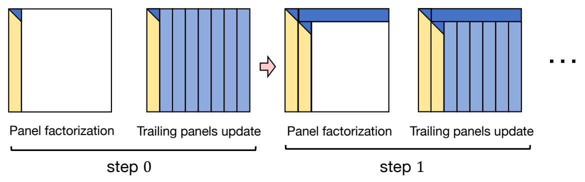

Figure 1: Block matrix factorization

The tile algorithm divides a matrix vertically and horizontally and decomposes and updates each submatrix (tile) (Fig. 2)[1]. By adjusting the tile size, it is possible to generate a sufficient amount of fine-grained tasks according to the number of parallel computing resources. The tasks are recognized by OpenMP `task` construct and scheduled by describing dependencies between tiles in `depend` clause. Smaller tiles can generate many fine-grained tasks, and this can resolve load imbalances, but it causes a decline in the performance of some tasks. Tile size tuning is a critical issue for the performance of the tile algorithm.

PLASMA[3] is a numerical linear algebra library for multi-core processors and provides routines for solving linear equations, eigenvalue problems, and singular value problems. The matrix factorization routine of this library is parallelized using the tile algorithm and follows the formula of OpenMP[4]. In PLASMA library, (one-sided) matrix factorizations such as LU, Cholesky, and QR factorizations are implemented. Two-sided matrix factorizations, such as block diagonalization, are also implemented by the tile algorithm[2]. The author knows an example of task parallelization using OpenMP for the block Jacobi method, which is an eigenvalue solver for symmetric matrices.

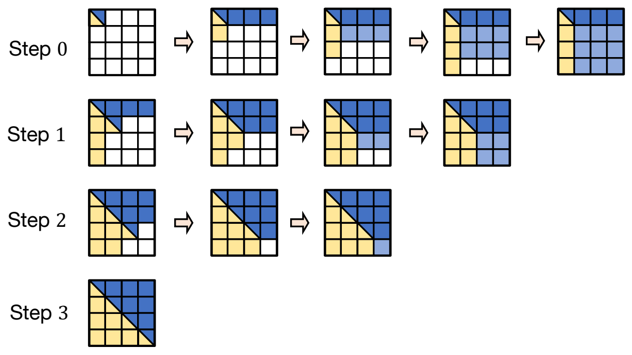

Figure 2: Tile matrix factorization (4×4 tiles)

In recent years, heterogeneous computing environments using GPUs and FPGAs as accelerators had become popular. The classical ways to get large speed benefits by throwing a large task into the accelerator is not compatible with the task-parallel programming model. It is necessary to verify the effectiveness of the task-parallel algorithm in such a system.

In this presentation, the author presents some brief experimental results of task-parallel programs for matrix factorization with both block and tile algorithms.

REFERENCES

[1] A. Buttari, J. Langou, J. Kurzak, and J. J. Dongarra, "A class of parallel tiled linear algebra algorithms for multicore architectures," LAPACK Working Note, Tech. Rep. 191, Sep. 2007. [Online]. Available: http://www.netlib.org/lapack/lawnspdf/lawn191.pdf
[2] H. Ltaief, J. Kurzak, and J. Dongarra, "Parallel band two-sided matrix bidiagonalization for multicore architectures." LAPACK Working Note, Tech. Rep. 209, Oct. 2008. [Online]. Available: http://www.netlib.org/lapack/lawnspdf/lawn209.pdf
[3] PLASMA. (2008) Accessed 2019-03-22. [Online]. Available: https://bitbucket.org/icl/plasma
[4] A. YarKhan, J. Kurzak, P. Luszczek, and J. Dongarra, "Porting the plasma numerical library to the OpenMP standard," *International Journal of Parallel Programming*, vol. 45, pp. 612 – 633. [Online]. Available: https://link.springer.com/article/10.1007/s10766-016-0441-6