Exploiting Fine-Grain Parallelism in Recursive LU Factorization[1]

Jack Dongarra [a], Mathieu Faverge [a], Hatem Ltaief [b], and Piotr Luszczek [a]
[a] *Department of Electrical Engineering and Computer Science, University of Tennessee, Knoxville*
Email: {dongarra,faverge,luszczek}@eecs.utk.edu
[b] *KAUST Supercomputing Laboratory, Thuwal, Saudi Arabia*
Email: Hatem.Ltaief@kaust.edu.sa

Keywords. recursion, LU factorization, parallel linear algebra, shared-memory synchronization, threaded parallelism

1. Introduction

The LU factorization is an important numerical algorithm for solving system of linear equations in science and engineering and is characteristic of many dense linear algebra computations. It has even become the de facto numerical algorithm implemented within the LINPACK benchmark to rank the most powerful supercomputers in the world, collected in the TOP500 website. In this context, the challenge in developing new algorithms for the scientific community resides in the combination of two goals: achieving high performance and maintaining the accuracy of the numerical algorithm. This paper proposes a novel approach for computing the LU factorization in parallel on multicore architectures, which not only improves the overall performance, but also sustains the numerical quality of the standard LU factorization algorithm with partial pivoting. While the update of the trailing submatrix is computationally intensive and highly parallel, the inherently problematic portion of the LU factorization is the panel factorization due to its memory-bound characteristic as well as the atomicity of selecting the appropriate pivots.

We present a new approach to LU factorization of (narrow and tall) panel submatrices. It uses a parallel fine-grained *recursive* formulation of the factorization. It is based on conflict-free partitioning of the data and lockless synchronization mechanisms. As a result, our implementation lets the overall computation naturally flow without contention. Our recursive panel factorization provides the necessary performance increase for the inherently problematic portion of the LU factorization of square matrices. The reason is that even though the panel

[1]Research reported here was partially supported by the National Science Foundation, the Department of Energy, and Microsoft Research.

factorization is a lower order term from the computational complexity perspective [1], it still poses a problem in the parallel setting from the theoretical [2] and practical standpoints [3]. For good performance of BLAS calls, panel width is commonly increased. This creates tension if the panel is a sequential operation because a larger panel width results in larger Amdahl's fraction [4]. Our own experiments revealed this to be a major obstacle to proper scalability of our implementation of square matrix LU factorization with partial pivoting – a result consistent with related efforts [3].

The performance results of our implementation reveal superlinear speedup and far exceeds what can be achieved with equivalent MKL and/or LAPACK routines.

2. Parallel Recursive LU Factorization of a Panel

This Section describes one of our most unique contributions, which is the parallelization of the LU factorization of a matrix panel using the recursive algorithm [5].

Even though the panel factorization is a lower order term – $O(\mathsf{N}^2)$ compared with the global $O(\mathsf{N}^3)$ – from the computational complexity perspective [1], it still poses a problem in the parallel setting from the theoretical [2] and practical standpoints [3]. To be more precise, the combined panel factorizations' complexity for the entire matrix is $O(\mathsf{N}^2\mathsf{NB})$, where N is panel height (and matrix dimension) and NB is panel width. For good performance of BLAS calls, panel width is commonly increased. This creates tension if the panel is a sequential operation because a larger panel width results in larger Amdahl's fraction [4]. Our own experiments revealed this to be a major obstacle to proper scalability of our implementation of tile LU factorization with partial pivoting – a result consistent with related efforts [3].

Aside from gaining high level formulation free of low level tuning parameters, recursive formulation affords us to dispense of a higher level tuning parameter commonly called algorithmic blocking. There is already panel width – a tunable value used for merging multiple panel columns together. Non-recursive panel factorizations could potentially establish another level of tuning called *inner-blocking* [6,7]. This is avoided in our implementation.

3. Scalability Results of the Parallel Recursive Panel Kernel

Figure 1 shows a scalability study on the NUMA machine *Opteron-48* (an eight-socket machine with a six-core AMD Opteron™ 8439 SE processor in each socket) of our parallel recursive panel LU factorization with four different panel widths: 32, 64, 128, and 256 against equivalent routines from LAPACK. We limit our parallelism level to 16 cores because our main factorization needs the remaining cores for trailing matrix updates. When compared with the panel factorization routine xGETF2() (mostly Level 2 BLAS), we achieve super-linear speedup for a wide range of panel heights with the maximum achieved effi-

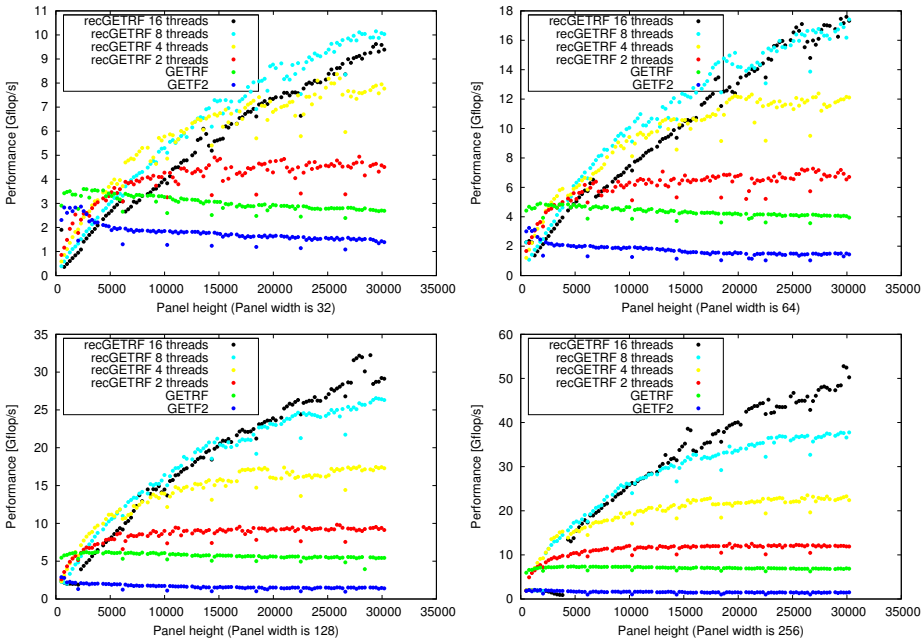

Figure 1. Scalability study of the recursive parallel panel factorization with various panel widths: 32 (top left), 64 (top right), 128 (bottom left), and 256 (bottom right).

ciency exceeding 550%. In an arguably more relevant comparison against the xGETRF() routine, which could be implemented with mostly Level 3 BLAS, we achieve perfect scaling for 2 and 4 threads and easily exceed 50% efficiency for 8 and 16 threads. This is consistent with the results presented in the related work section [3].

References

[1] E. Anderson and J. Dongarra. Implementation guide for LAPACK. Technical Report UT-CS-90-101, University of Tennessee, Computer Science Department, April 1990. LAPACK Working Note 18.

[2] G. M. Amdahl. Validity of the single-processor approach to achieving large scale computing capabilities. In *AFIPS Conference Proceedings*, volume 30, pages 483–485, Atlantic City, N.J., APR 18-20 1967. AFIPS Press, Reston, VA.

[3] Anthony M. Castaldo and R. Clint Whaley. Scaling LAPACK panel operations using parallel cache assignment. *Proceedings of the 15th ACM SIGPLAN symposium on Principles and practice of parallel programming*, pages 223–232, 2010.

[4] John L. Gustafson. Reevaluating Amdahl's Law. *Communications of ACM*, 31(5):532–533, 1988.

[5] Fred G. Gustavson. Recursion leads to automatic variable blocking for dense linear-algebra algorithms. *IBM Journal of Research and Development*, 41(6):737–755, November 1997.

[6] E. Agullo, J. Demmel, J. Dongarra, B. Hadri, J. Kurzak, J. Langou, H. Ltaief, P. Luszczek, and S. Tomov. Numerical linear algebra on emerging architectures: The PLASMA and MAGMA projects. *Journal of Physics: Conference Series*, 180, 2009.

[7] Emmanuel Agullo, Bilel Hadri, Hatem Ltaief, and Jack Dongarrra. Comparative study of one-sided factorizations with multiple software packages on multi-core hardware. In *SC '09: Pro-*

ceedings of the Conference on High Performance Computing Networking, Storage and Analysis, pages 1–12, New York, NY, USA, 2009. ACM.