Numerical Linear Algebra on Emerging Architectures: the PLASMA and MAGMA Projects

Emmanuel Agullo, Jim Demmel, Jack Dongarra, Bilel Hadri, Jakub Kurzak, Julien Langou, Hatem Ltaief, Piotr Luszczek, Stanimire Tomov[1]

Department of Electrical Engineering and Computer Science, University of Tennessee, USA

E-mail: eagullo,dongarra,hadri,kurzak,ltaief,luszczek,tomov@eecs.utk.edu

E-mail: demmel@cs.berkeley.edu

E-mail: julien.langou@ucdenver.edu

Abstract. The emergence and continuing use of multi-core architectures and graphics processing units require changes in the existing software and sometimes even a redesign of the established algorithms in order to take advantage of now prevailing parallelism. Parallel Linear Algebra for Scalable Multi-core Architectures (PLASMA) and Matrix Algebra on GPU and Multics Architectures (MAGMA) are two projects that aims to achieve high performance and portability across a wide range of multi-core architectures and hybrid systems respectively. We present in this document a comparative study of PLASMA's performance against established linear algebra packages and some preliminary results of MAGMA on hybrid multi-core and GPU systems.

1 Introduction

Recent activities of major chip manufacturers, such as Intel, AMD, IBM and NVIDIA, make it more evident than ever that future designs of microprocessors and large HPC systems will be heterogeneous in nature, relying on the integration of two major types of components. On the first hand, multi/many-cores CPU technology have been recently developed and the number of cores will continue to escalate because of the desire to pack more and more components on a chip while avoiding the power wall, instruction level parallelism wall, and the memory wall [1]. And on the other hand special purpose hardware and accelerators, especially Graphics Processing Units (GPUs) are in commodity production, and have outpaced standard CPUs in floating point performance in recent years, and have become as easy, if not easier to program than multi-core CPUs.

To address the critical and highly disruptive situation that is facing the linear algebra and high performance computing (HPC) community due to the introduction of multi-core architectures and GPUs, we have developed two projects called Parallel Linear Algebra Software for Multi-core Architectures (PLASMA) [2, 3] and Matrix Algebra on GPU and Multi-core Architectures (MAGMA). PLASMA is a redesign of LAPACK [4] and ScaLAPACK [5] for shared memory computers based on multi-core processor architectures. To achieve high performance on this type of architecture, PLASMA relies on tile algorithms, which provide fine granularity parallelism. The MAGMA project aims to develop a dense linear algebra library similar to LAPACK but for heterogeneous/hybrid architectures, starting with current "Multi-core+GPU" systems.

The document is organized as follows. Section 2 gives an overview on PLASMA and its performance

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

against established linear algebra packages such as LAPACK, ScaLAPACK as well as equivalent commercial software offerings (Intel MKL, IBM ESSL and IBM PESSL) on two different architectures. Section 3 presents MAGMA and some preliminary results of the project on hybrid multi-core and GPU systems.

2 PLASMA

To achieve high performance on multi-core architectures, PLASMA relies on tile algorithms, which provide fine granularity parallelism. The standard linear algebra algorithms can then be represented as Directed Acyclic Graphs (DAG) [6] where nodes represent tasks and edges represent dependencies among them. Our programming model enforces asynchronous, out of order scheduling of operations. This concept is used as the basis for a scalable yet highly efficient software framework for computational linear algebra applications.

In LAPACK, parallelism is obtained though the use of multithreaded Basic Linear Algebra Subprograms (BLAS) [7]. In PLASMA, parallelism is no longer hidden inside the BLAS but is brought to the fore to yield much better performance.

PLASMA performance strongly depends on tunable execution parameters trading off utilization of different system resources. The outer block size (NB) trades off parallelization granularity and scheduling flexibility with single core utilization, while the inner block size (IB) trades off memory load with extra-flops.

We present in this paper a study of the three one sided factorization present in LAPACK: Cholesky, QR, and LU. Each of the one-sided tile factorizations presents increasingly complex challenges to parallel programming. Cholesky is the easiest. The tile algorithm is represented by a DAG with relatively little work required on the critical path. LU and QR factorizations have exactly the same dependency pattern between the nodes of the DAG. These two factorizations exhibit much more severe scheduling constraints than the Cholesky factorization. Moreover the stability of the tile LU factorization is not yet well understood.

PLASMA is currently scheduled statically with a trade off between load balancing and data reuse.

2.1 Comparison to other libraries

We perform a comparative study of PLASMA against established linear algebra packages (LAPACK and ScaLAPACK) as well as equivalent commercial software offerings (Intel MKL, IBM ESSL and IBM PESSL) for Cholesky, LU and QR factorization. We also compare with a new approach at parallel execution called TBLAS [8] (Task Based Linear Algebra Subroutines) in the Cholesky and QR cases – the TBLAS LU factorization has not yet been implemented. The experiments were conducted on two different multi-core architectures based on Intel Xeon EMT64 and IBM Power6,

PLASMA, TBLAS, LAPACK and ScaLAPACK are all linked with the optimized vendor BLAS available on the system provided within Intel MKL 10.1 and IBM ESSL 4.3 on the Intel64 and Power6 architectures, respectively. The first architecture is a quad-socket quad-core machine based on an Intel Xeon EMT64 E7340 processor operating at 2.39 GHz. Its theoretical peak is equal to 9.6 Gflop/s/ per core or 153.2 Gflop/s for the whole node (16 cores). The second architecture is a SMP node composed of 16 dual-core Power6 processors. Each dual-core Power6 processor runs at 4.7 GHz, leading to a theoretical peak of 18.8 Gflop/s per core and 601.6 Gflop/s per node (32 cores).

2.2 Methodology

Factorizations are performed in double precision. PLASMA is tuned with the pruned search method as described in [9]. TBLAS, ScaLAPACK and PESSL have been tuned with an exhaustive search (because their search space is smaller). LAPACK, MKL and ESSL have been tuned by the vendor.

Furthermore, to capture the best possible behavior of each library, we repeat the number of executions (up to 10 times) and we report the highest performance obtained. We do not flush the caches [10] before timing a factorization[2]. However, the TLB (Translation Lookaside Buffer) is flushed between two executions: the

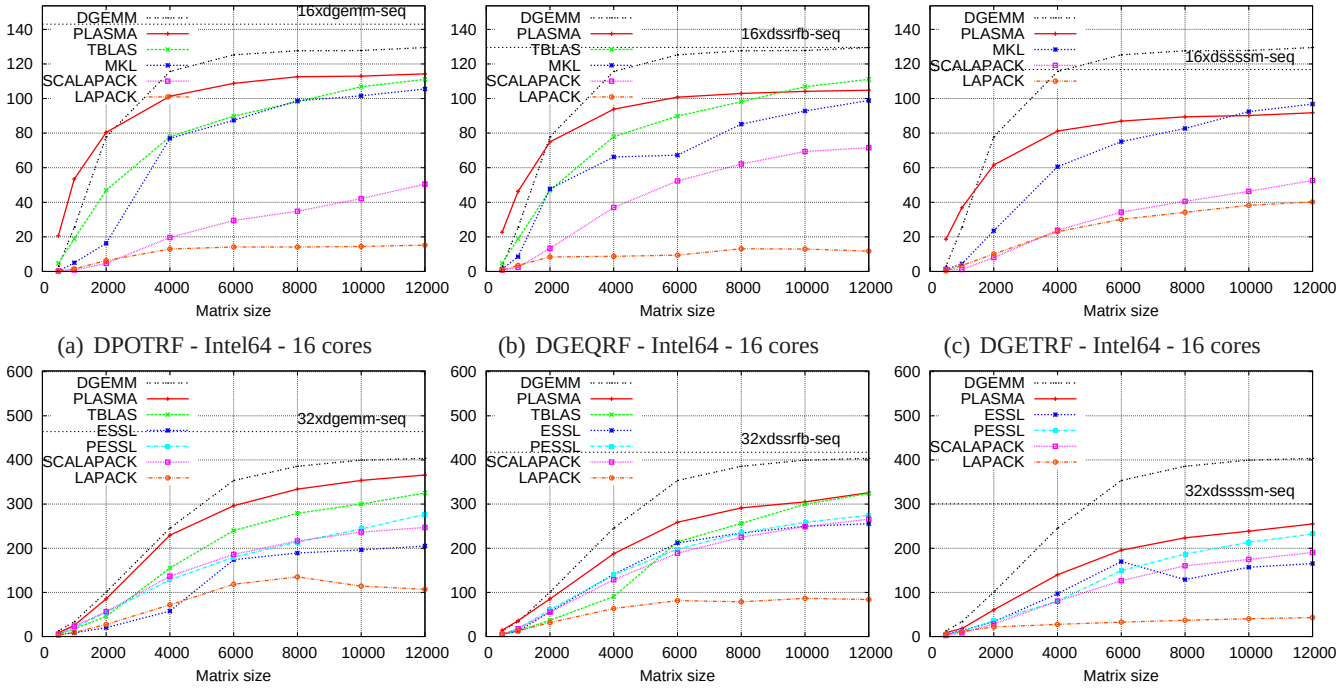
[2] It is kernel usage, not problem size, that dictates whether one wish to flush the cache [10]. Warm (or partially warm) cache executions are plausible for dense linear factorizations. For instance, sparse linear solvers, which rely on dense kernels, intend to

loop over the different executions is performed in a script (rather than within the executable) and calls several times the same executable.

ScaLAPACK, PESSL and PLASMA interfaces allow the user to provide data distributed on the cores. In our shared-memory multi-core environment, because we do not flush the caches, these libraries have thus the advantage to start the factorization with part of the data distributed on the caches. This is not negligible. For instance, a 8000×8000 double precision matrix can be held distributed on the L3 caches of the 32 cores of a Power6 node.

2.3 Experimental results

We present in this section results of experiments conducted on a large number of cores (16 cores on Intel64; 32 cores on Power6). Figures 1(a), and 1(d) present the Cholesky (DPOTRF routine) performance of the different libraries. PLASMA consistently outperforms the other libraries, followed by TBLAS. These results illustrate the performance improvement brought by tile algorithms. The higher efficiency of PLASMA compared to TBLAS is essentially due to a better data reuse. Indeed, PLASMA scheduling strategy maximizes data reuse and thus benefits from a better cache effect than TBLAS whose scheduler does not take into account date reuse. PLASMA is even faster than the parallel DGEMM reference up to a matrix size



(a) DPOTRF - Intel64 - 16 cores (b) DGEQRF - Intel64 - 16 cores (c) DGETRF - Intel64 - 16 cores

(d) DPOTRF - Power6 - 32 cores (e) DGEQRF - Power6 - 32 cores (f) DGETRF - Power6 - 32 cores

Figure 1: Performance comparison on a large number of cores (Gflop/s).

$N = 2000$ on the Intel64 machine when 16 cores are used. This is not contradictory since PLASMA does not rely on the parallel version of DGEMM. Each PLASMA thread indeed uses the serial dgemm-seq. Better performance can then be achieved thanks to better scheduling. For large matrix size, the parallel DGEMM dominates all the other operations. This illustrates the fact that using a fine enough granularity (as in tile algorithms) is more critical when processing small or moderate size matrices. This also explains the major improvement brought by PLASMA compared to the other libraries on matrices of small and moderate size.

Figures 1(b) and 1(e) illustrate the performance of the QR factorization (DGEQRF routine) when all the available cores are used (16 on Intel64 or 32 on Power6). PLASMA outperforms the other libraries and

maximize data reuse between successive calls to dense operations.

TBLAS is also very competitive. These results demonstrate the excellent scalability of tile algorithms. On the Intel64 machine, TBLAS actually has a better performance than PLASMA when 16 cores are used and when the matrix size is larger than or equal to 10,000. Indeed, when the matrices processed are large, the critical scheduling problem corresponds to maximizing a steady state throughput. The main disadvantage of a static schedule is that cores may be stalling in situations where work is available. This throughput is easier to maximize with a dynamic scheduling strategy. Approaches such as TBLAS, which do implement a dynamic scheduling, are thus likely to achieve a higher performance than approaches that implement a static scheduling (such as PLASMA currently does). All in all, these results are motivation for a dynamic scheduling mechanism that would assign priorities according to a trade off between data reuse and critical path progress.

Finally, figures 1(c), and 1(f) show that the LU factorization (DGETRF routine) has a performance behavior similar to the QR factorization and PLASMA again outperforms the other libraries. However, the lower efficiency of dssssm-seq compared to dssrfb-seq (dssssm-seq performs more extra-flops) induces a lower performance of the PLASMA LU factorization compared to the PLASMA QR one. On Intel64, this leads MKL to be slightly better than PLASMA when the matrix size is larger than or equal to 10,000 ($N \geq 10,000$). But, similarly to the QR case, moving towards a hybrid scheduling should remove the penalty due to the static scheduling strategy used in PLASMA which should improve the performance on large matrices. Furthermore, we note that an optimized implementation of the dssssm-seq kernel will improve the performance of tile algorithms.

3 MAGMA

The MAGMA research is based on the idea that, to address the complex challenges of the emerging hybrid environments, optimal software solutions will themselves have to hybridize, combining the strengths of different algorithms within a single framework. Building on this idea, we aim to design linear algebra algorithms and frameworks for hybrid manycore and GPUs systems that can enable applications to fully exploit the power that each of the hybrid components offers.

The problems and the challenges for developers in the new computational landscape of hybrid processors are daunting. Critical parts of the software infrastructure are already having a very difficult time keeping up with the pace of change: In some cases, performance is not scaling up as the number of cores grows because more and more time is spent on slow data movement rather than fast arithmetic.

Preliminary studies on a new class of "heterogeneity-aware" algorithms of "reduced communication" and "high-parallelism" confirm that this is the case. An example of the new class of communication-optimal algorithms is the RBT LP LU(NB) algorithm that we developed for Multi-core + GPU systems [11]. This algorithm aims at solving a nonsymmetric linear system of equations. Figure 2 shows its performance, comparing it with the pairwise pivoting (PwP) LU from PLASMA on current state-of-the-art multi-core systems, and the PP LU for 1 Core + 1 GPU. This algorithm underscores another change in the design space that is characteristic for many of the new techniques [12], namely that the new techniques often gain in speed for the price of relaxed accuracy. Understanding this trade-off of speed *vs* accuracy has to be further theoretically studied as it can lead to very efficient algorithms. For example here, experiments with random matrices show that LP LU(NB+64) is comparable in accuracy to PP LU, and LP LU(NB) loses only from 1 to 2 digits of accuracy to gain up to 30% in speed compared to PP LU. Another example, also related to reducing communication, is using mixed precision algorithms. Mixed precision solvers for example often achieve significant speedups on GPUs (e.g. up to 4$\times$ on the GTX 280 [12]) compared to double precision solvers but the speed depends on the condition number of the matrix.

4 Conclusion and perspectives

We have shown the performance improvements brought by tile algorithms on up to 32 cores – the largest shared memory multi-core system we could access. We may expect that these results generalize somewhat to other linear algebra algorithms and even any algorithm that can be expressed by a DAG of fine-grain tasks.

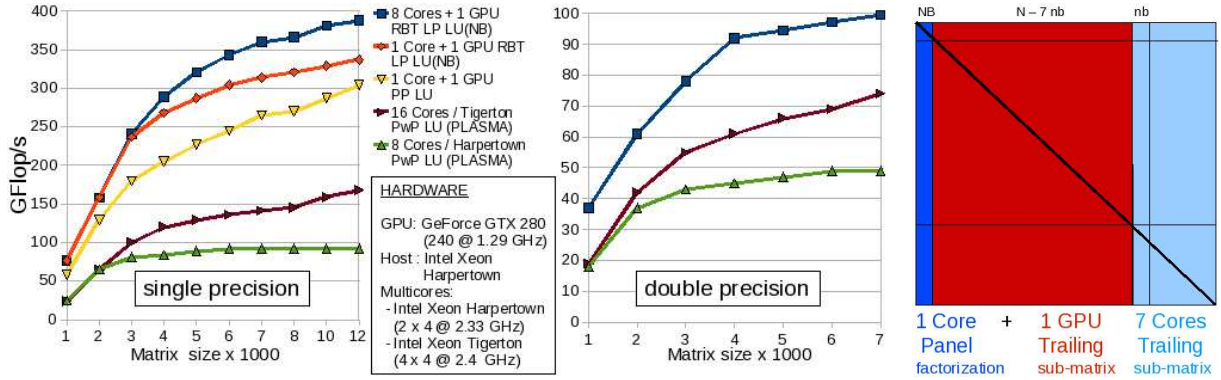


Figure 2: Performance of a communication-optimal LU for a Multi-core + GPU system in both single (left) and double (middle) precision arithmetic *vs* the PwP LU from PLASMA for multi-cores and the PP LU for 1 Core + 1 GPU system. The heterogeneity-aware work splitting used is shown on the right (for an 8 cores host).

We are also working on the interpolation of the optimum tuning parameters from a limited number of parallel executions among the range of cores and matrix sizes to the full set of possibilities. This on-going auto-tuning work should eventually be incorporated within the PLASMA software distribution. Our experiments have also shown the limits of static scheduling for the factorization of large matrices. We are currently working on the implementation of a dynamic scheduling for PLASMA.

Current work on MAGMA show that architecture trends have moved towards heterogeneous (GPU + CPU) designs of increased parallelism and communication costs, and software trends have to reflect on that. MAGMA addresses this with innovative heterogeneity-aware algorithms/techniques on extracting parallelism and reducing communication.

[1] K. Asanovic, R. Bodik, B. C. Catanzaro, J. J. Gebis, P. Husbands, K. Keutzer, D. A. Patterson, W. L. Plishker, J. Shalf, S. W. Williams, and K. A. Yelick. The landscape of parallel computing research: A view from Berkeley. Technical Report UCB/EECS-2006-183, Electrical Engineering and Computer Sciences Department, University of California at Berkeley, 2006.

[2] Buttari A., Langou J., Kurzak J., and Dongarra J. A class of parallel tiled linear algebra algorithms for multicore architectures. *Parallel Computing*, 35(1):38–53, 2009.

[3] Buttari A., Dongarra J., Kurzak J., Langou J., Luszczek P., and Tomov S. The impact of multicore on math software. *PARA 2006, Umea, Sweden*, June 2006.

[4] Anderson E., Bai Z., Bischof C., Blackford L. S., Demmel J. W., Dongarra J., Du Croz J., Greenbaum A., Hammarling S., McKenney A., and Sorensen D. *LAPACK Users' Guide*. SIAM, 1992.

[5] Blackford L. S., Choi J., Cleary A., D'Azevedo E., Demmel J., Dhillon I., Dongarra J., Hammarling S., Henry G., Petitet A., Stanley K., Walker D., and Whaley R. C. *ScaLAPACK Users' Guide*. SIAM, 1997.

[6] Christofides N. *Graph Theory: An algorithmic Approach*. 1975.

[7] BLAS: Basic linear algebra subprograms. http://www.netlib.org/blas/.

[8] Song F., YarKhan A., and Dongarra J. Dynamic task scheduling for linear algebra algorithms on distributed-memory multicore systems. Technical report, UTK CS Technical Report 638, 2009.

[9] E. Agullo, B. Hadri, H. Ltaief, and J. Dongarra. Comparative study of one-sided factorizations with multip le software packages on multi-core hardware. Technical report, 2009. Submitted to SC09.

[10] Whaley R. Clint and Castaldo Anthony M. Achieving accurate and context-sensitive timing for code optimization. *Software: Practice and Experience*, 38(15):1621–1642, 2008.

[11] Stanimire Tomov, Jack Dongarra, and Marc Baboulin. Towards dense linear algebra for hybrid GPU accelerated manycore systems. Technical Report UT-CS-08-632, University of Tennessee, 2008. Parallel Computing (submitted).

[12] M. Baboulin, J. Demmel, J. Dongarra, S. Tomov, and V. Volkov. Enhancing the performance of dense linear algebra solvers on GPUs [in the MAGMA project]. Poster at Supercomputing 08, November 18, 2008. http://www.cs.utk.edu/~tomov/SC08-poster.pdf.