Comparative Study of One-Sided Factorizations with Multiple Software Packages on Multi-Core Hardware

LAPACK Working Note ♯217

Emmanuel Agullo, Bilel Hadri, Hatem Ltaief and Jack Dongarra*

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

Abstract

The emergence and continuing use of multi-core architectures require changes in the existing software and sometimes even a redesign of the established algorithms in order to take advantage of now prevailing parallelism. The Parallel Linear Algebra for Scalable Multi-core Architectures (PLASMA) is a project that aims to achieve both high performance and portability across a wide range of multi-core architectures. We present in this paper a comparative study of PLASMA's performance against established linear algebra packages (LAPACK and ScaLAPACK), against new approaches at parallel execution (Task Based Linear Algebra Subroutines – TBLAS), and against equivalent commercial software offerings (MKL, ESSL and PESSL). Our experiments were conducted on one-sided linear algebra factorizations (LU, QR and Cholesky) and used multi-core architectures (based on Intel Xeon EMT64 and IBM Power6). The performance results show improvements brought by new algorithms on up to 32 cores – the largest multi-core system we could access.

1 Introduction and Motivations

The emergence of multi-core microprocessor designs marked the beginning of a forced march toward an era of computing in which research applications must be able to exploit parallelism at a continuing pace and unprecedented scale [1]. This confronts the scientific software community with both a daunting challenge and a unique opportunity. The challenge arises from the disturbing mismatch between the design of systems based on this new chip architecture – hundreds of thousands of nodes, a million or more cores, reduced bandwidth and memory available to cores – and the components of the traditional software stack, such as numerical libraries, on which scientific applications have relied for their accuracy and performance. So long as library developers could depend on ever increasing clock speeds and instruction level parallelism, they could also settle for incremental improvements in the scalability of their algorithms.

*Research reported here was partially supported by the National Science Foundation and Microsoft Research.

But to deliver on the promise of tomorrow's petascale systems, library designers must find methods and algorithms that can effectively exploit levels of parallelism that are orders of magnitude greater than most of today's systems offer. This is an unsolved problem.

To answer this challenge, we have developed a project called Parallel Linear Algebra Software for Multi-core Architectures (PLASMA) [2, 3]. PLASMA is a redesign of LAPACK [4] and ScaLAPACK [5] for shared memory computers based on multi-core processor architectures. It addresses 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. To achieve high performance on this type of architecture, PLASMA relies on tile algorithms, which provide fine granularity parallelism. The standard linear algebra algorithms can then be represented as a Directed Acyclic Graph (DAG) [6] where nodes represent tasks, either panel factorization or update of a block-column, and edges represent dependencies among them. Moreover, the development of programming models that enforce asynchronous, out of order scheduling of operations is the concept used as the basis for the definition of a scalable yet highly efficient software framework for computational linear algebra applications. In PLASMA, parallelism is no longer hidden inside Basic Linear Algebra Subprograms (BLAS) [7] but is brought to the fore to yield much better performance. Each of the one-sided tile factorizations presents unique challenges to parallel programming. Cholesky factorization 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 and numerical (only for LU) constraints than the Cholesky factorization. PLASMA is currently scheduled statically with a trade off between load balancing and data reuse.

The development of these new algorithms for multi-core systems on one-sided factorizations motivated us to show improvements made by the tile algorithms in performing a comparative study of PLASMA against established linear algebra packages (LAPACK and ScaLAPACK) as well as equivalent commercial software offerings (MKL, ESSL and PESSL). Moreover, we also compared a new approach at parallel execution called TBLAS [8] (Task Based Linear Algebra Subroutines) in the Cholesky and QR cases – the LU factorization not being implemented yet. The experiments were conducted on two different multi-core architectures based on Intel Xeon EMT64 and IBM Power6, respectively.

The paper is organized as follows. Section 2 provides an overview of the libraries and the hardware used in the experiments, as well as a guideline to interpret the results. Section 3 describes the tuning process to achieve the best performance of each software. In section 4 we present a comparative performance evaluation of the different libraries before concluding and presenting future work directions in Section 5.

2 Experimental environment

We provide below an overview of the software used for the comparison against PLASMA as well as a description of the hardware platforms, *i.e.*, Intel Xeon EMT64 and IBM Power6 machines. We also introduce some performance metrics that will be useful to the interpretation of the experimental results.

2.1 Libraries

2.1.1 LAPACK, MKL and ESSL

LAPACK 3.2 [4], an acronym for Linear Algebra PACKage, is a library of Fortran subroutines for solving the most commonly occurring problems in numerical linear algebra for high-performance computers. It has been designed to achieve high efficiency on a wide range of modern high-performance computers such as vector processors and shared memory multiprocessors. LAPACK supersedes LINPACK [9] and EISPACK [10, 11]. The major improvement was to rely on the use of a standard set of Basic Linear Algebra Subprograms (BLAS) [7], which can be optimized for each computing environment. LAPACK is designed on top of the level 1, 2, and 3 BLAS, and nearly all of the parallelism in the LAPACK routines is contained in the BLAS.

MKL 10.1 (Math Kernel Library) [12], a commercial offering from Intel, is a library of highly optimized, extensively threaded math routines that require maximum performance such as BLAS. We consider in this paper the 10.1 version – the latest available version when the paper was submitted.

ESSL 4.3 (Engineering Scientific Subroutine Library) [13] is a collection of subroutines providing a wide range of performance-tuned mathematical functions specifically designed to improve the performance of scientific applications on the IBM Power processor-based servers (and other IBM machines).

2.1.2 ScaLAPACK and PESSL

ScaLAPACK 1.8.0 (Scalable Linear Algebra PACKage) [5] is a parallel implementation of LAPACK for parallel architectures such as massively parallel SIMD machines, or distributed memory machines. It is based on the Message Passing Interface (MPI) [14]. As in LAPACK, ScaLAPACK routines rely on block-partitioned algorithms in order to minimize the frequency of data movement between different levels of the memory hierarchy. The fundamental building blocks of the ScaLAPACK library are distributed memory versions of the Level 1, Level 2, Level 3 BLAS, called the Parallel BLAS or PBLAS [15], and a set of Basic Linear Algebra Communication Subprograms (BLACS) [16] for communication tasks that arise frequently in parallel linear algebra computations. In the ScaLAPACK routines, the majority of interprocessor communication occurs within the PBLAS and the interface is as similar to the BLAS as possible. The source code of the top software layer of ScaLAPACK looks very similar to that of LAPACK.

PESSL 3.3 (Parallel Engineering Scientific Subroutine Library) [13] is a scalable mathematical library that supports parallel processing applications. It is an IBM commercial offering for Power processor-based servers (and other IBM machines).

Although ScaLAPACK and PESSL are primarily designed for distributed memory architectures, a driver optimized for shared-memory machines allows memory copies instead of actual communications.

2.1.3 TBLAS

The TBLAS [8] project is an on-going effort to create a scalable, high-performance task-based linear algebra library that works in both shared memory and distributed memory environments. TBLAS takes inspiration from SMP Superscalar (SMPSs) [17], PLASMA and ScaLAPACK. It adopts a dynamic task scheduling approach to execute dense linear algebra algorithms on multi-core systems. A task-based library replaces the existing linear algebra subroutines such as PBLAS. TBLAS transparently provides the same interface and functionality as ScaLAPACK. Its runtime system extracts a DAG of tasks from the sequence of function calls of a task-based linear algebra program. Data dependencies between tasks are identified dynamically. Their execution is driven by data availability. To handle large DAGs, a fixed-size task window is enforced to unroll a portion of the active DAG on demand. Hints regarding critical paths (*e.g.*, panel tasks for factorizations) may also be provided to increase the priority of the corresponding tasks. Although the TBLAS runtime system transparently handles any data movement required for distributed memory executions, we will not use this feature since we focus on shared-memory multi-core architectures. So far the TBLAS project is in an experimental stage and the LU factorization (DGETRF routine) is not completed yet.

PLASMA, TBLAS, LAPACK and ScaLAPACK are all linked with the optimized vendor BLAS available on the system, which are MKL 10.1 and ESSL 4.3 on the Intel64 and Power6 architectures, respectively.

2.2 Hardware

2.2.1 Intel64-based 16 cores machine

Our first architecture is a quad-socket quad-core machine based on an Intel Xeon EMT64 E7340 processor operating at 2.39 GHz. The theoretical peak is equal to 9.6 Gflop/s/ per core or 153.2 Gflop/s for the whole node, composed of 16 cores. There are two levels of cache. The level-1 cache, local to the core, is divided into 32 kB of instruction cache and 32 kB of data cache. Each quad-core processor being actually composed of two dual-core Core2 architectures, the level-2 cache has 2×4 MB per socket (each dual-core shares 4 MB). The effective bus speed is 1066 MHz per socket leading to a bandwidth of 8.5 GB/s (per socket). The machine is running Linux 2.6.25 and provides Intel Compilers 11.0 together with the MKL 10.1 vendor library.

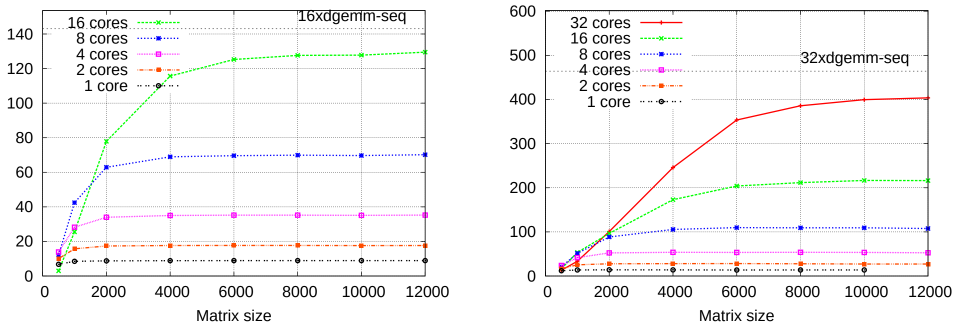
2.2.2 Power6-based 32 cores machine

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. There are three levels of cache. The level-1 cache, local to the core, can contain 64 kB of data and 64 kB of instructions; the level-2 cache is composed of 4 MB per core, accessible by the other core; and the level-3 cache is composed of 32 MB common to both cores of a processor with one controller per core (80 GB/s). An amount of 256 GB of memory is available and the memory bus (75 GB/s) is shared by the 32 cores of the node. The machine runs AIX 5.3 and provides the xlf 12.1 and xlc 10.1 compilers together with the ESSL 4.3 vendor library. Actually, the whole machine is the IBM Vargas[1] – ranked 50th on the TOP500 of November 2008 [18] – from IDRIS[2] and is composed of 112 SMP nodes for a total 67.38 Tflops/s theoretical peak. However, we use only a single node at a time since we focus on shared-memory, multi-core architectures.

2.3 Performance metrics (How to interpret the graphs)

In this section, we present some performance references and definitions that we will consistently report in the paper to evaluate the efficiency of the libraries presented above.

We report three performance references. The first one is the *theoretical peak* of the hardware used. All the graphs in the paper are scaled to that value, which depends on the processor used (Intel64 or Power6) and the number of cores involved (or the maximum number of cores if results on different number of cores are presented in the same graph). For instance, the graphs of Figures 1(a) and 1(b) are scaled to 153.2 Gflop/s and 601.6 Gflop/s since they respectively involve a maximum of 16 Intel64 cores and 32 Power6 cores. The second reference is the parallel vendor DGEMM performance (*i.e.*, the MKL and ESSL ones on Intel64 and Power6, respectively). DGEMM is a commonly used performance reference for parallel linear algebra algorithms; achieving a comparable performance is challenging since it is a parallel-friendly level-3 BLAS operation. Figures 1(a) and 1(b) illustrate very good scalability on both platforms. The last reference we consider is related to the peak performance of the serial computational intensive



(a) Intel64 (b) Power6

Figure 1: DGEMM performance (Gflop/s).

level-3 BLAS kernel effectively used by PLASMA. The kernel used depends on the factorization; they are respectively dgemm-seq, dssrfb-seq and dssssm-seq for the Cholesky (DPOTRF), QR (DGEQRF) and LU (DGETRF) algorithms. The dssrfb-seq and dssssm-seq routines are new kernels (not part of BLAS nor LAPACK standard) that have been specifically designed for tile algorithms. They involve extra-flops (that can lead to a total 50% overhead if no inner blocking occurs) [2].

Their complete description is out of the scope of this paper and more information can be found in [2]. The per-

[1] http://www.idris.fr/eng/Resources/index-vargas.html

[2] Institut du Développement et des Ressources en Informatique Scientifique: http://www.idris.fr.

formance of a PLASMA thread (running on a single core) is bounded by the performance – that we note *kernelseq* – of the corresponding serial level-3 kernel. Therefore, the performance of an execution of PLASMA on p cores is bounded by $p \times kernelseq$. For instance, Figures 2(a), 2(b) and 2(c) respectively show the performance upper bound for the DPOTRF, DGEQRF and DGETRF (PLASMA) factorizations; they correspond to the horizontal 16×dgemm-seq, 16×dssrfb-seq and 16×dssssm-seq dashed lines.

The *speedup* of a parallel execution on p cores is defined by the following formula: $s_p = \frac{t_1}{t_p}$ where p is the number of cores, t_1 the execution time of the sequential algorithm and t_p the execution time of the parallel algorithm when p cores are used. A *linear* speedup is obtained when $s_p = p$. This corresponds to the ideal case where doubling the number of cores doubles the speed.

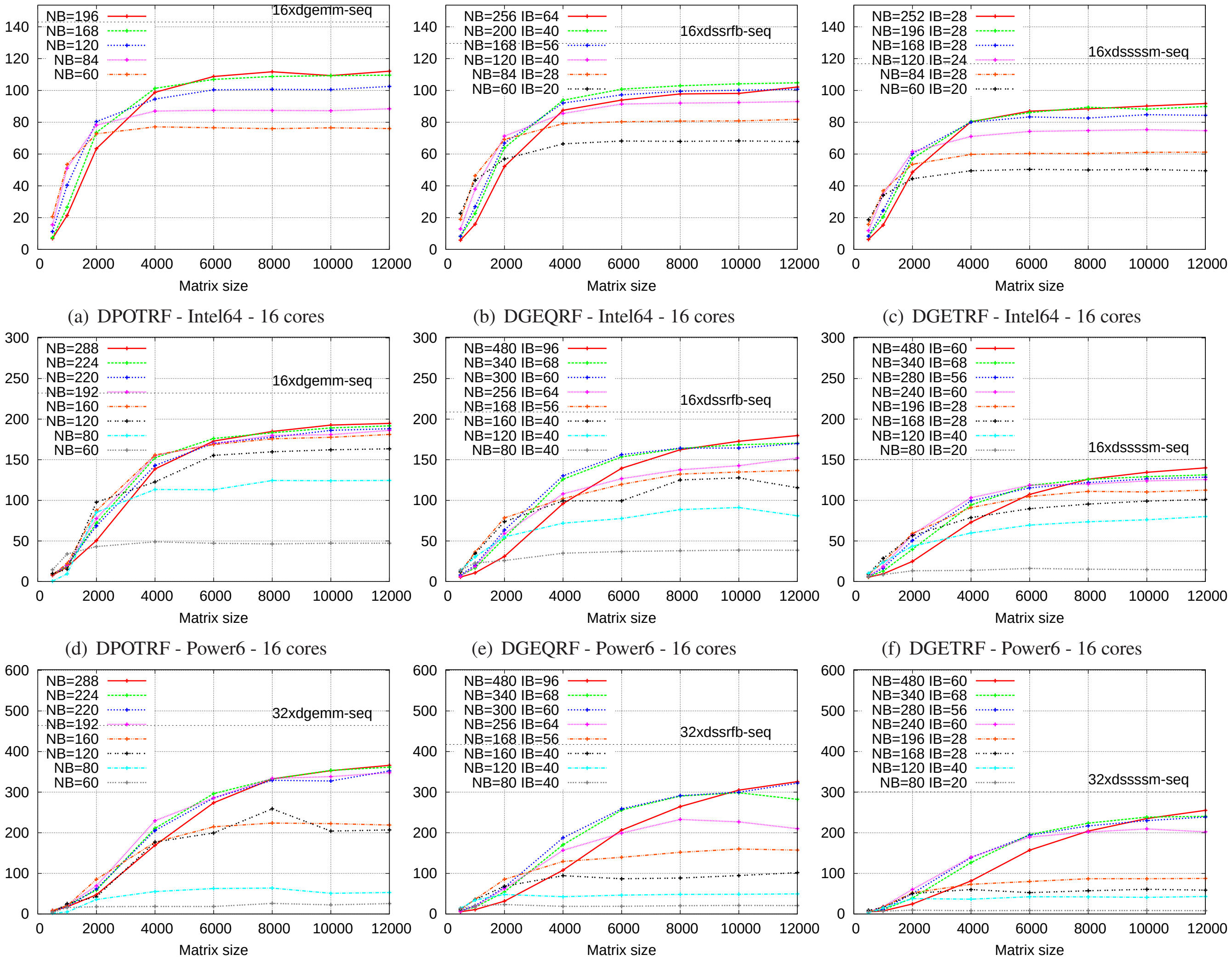
Another performance metric commonly used for parallel performance evaluation is the *efficiency*. It is usually defined as follows: $E_p = \frac{t_1}{p \times t_p}$. However, this metric does not provide much information about the relative performance of a library against another one, which is the focus of this paper. Therefore we introduce the *normalized efficiency* as $e_p = \frac{t_{kernelseq}}{p \times t_p}$ where $t_{kernelseq}$ is the time that the serial level-3 kernel (dgemm-seq, dssrfb-seq or dssssm-seq) would spend to perform an equivalent number of effective floating point operations. We say it is normalized since the sequential reference time $t_{kernelseq}$ consistently refers to a common operation independently of the considered library (but that still depends on the considered factorization). Therefore, that metric directly allows us to compare two libraries to each other. Furthermore, PLASMA normalized efficiency is bounded by a value of 1 ($e_p(PLASMA) \leq 1$) since the level-3 serial kernels are the tasks that achieve the highest performance during the PLASMA factorization process. That normalized efficiency can also be interpreted geometrically. It corresponds to the performance ratio of the considered operation to the $p \times kernelseq$ reference. For instance, in Figure 2(b), the normalized efficiency of PLASMA is the ratio between the PLASMA performance graphs to the 16×dgerfb-seq dashed line. The peak performance (obtained when $NB = 200$, $IB = 40$ and $N = 12000$) being equal to 102.1 Gflop/s and the 16×dgerfb-seq performance being equal to 129.6 Gflops, the peak normalized efficiency is thus equal to $e_p = 0.79$.

3 Tuning

Maximizing the performance and minimizing the execution time of scientific applications is a daily challenge for the HPC community. The libraries presented in the previous section have tunable execution parameters and a wrong setting for a single parameter can dramatically slow down the whole application. This section presents an overview of the tuning method applied to each library.

3.1 PLASMA

PLASMA performance strongly depends on tunable execution parameters trading off utilization of different system resources, the outer and the inner blocking sizes as illustrated in Figure 2. The outer block size (NB) trades off



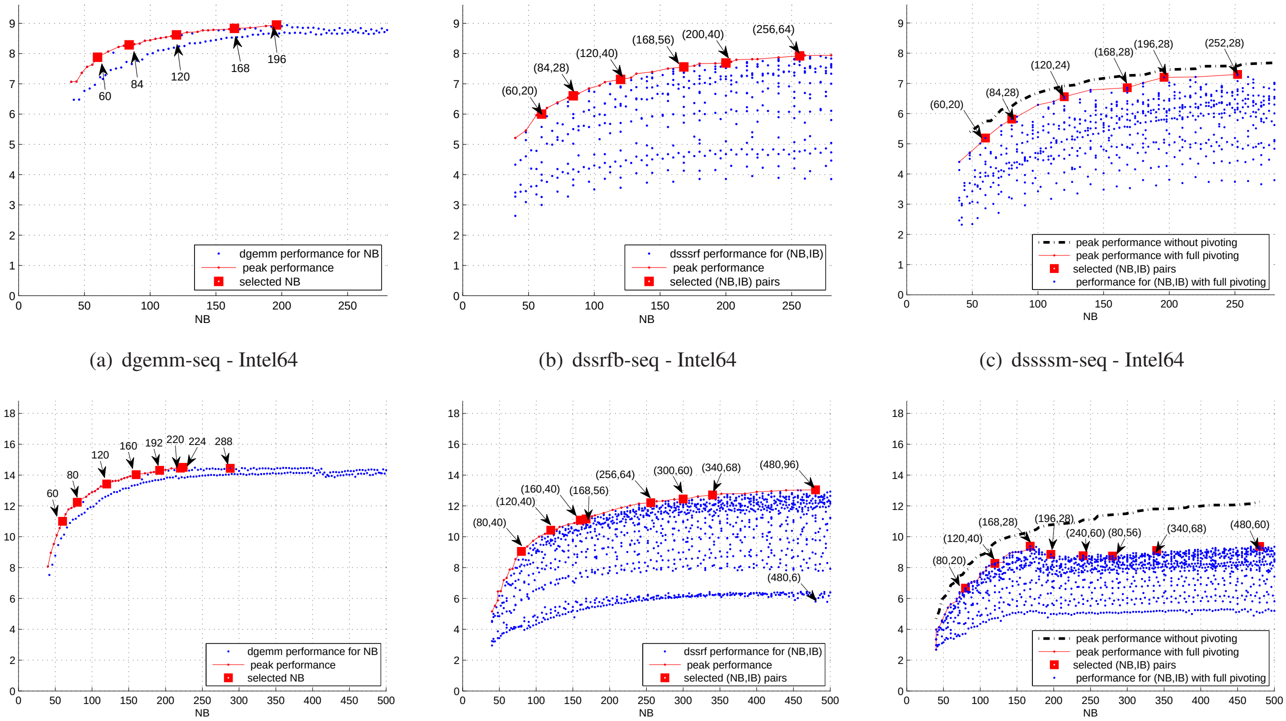
(a) DPOTRF - Intel64 - 16 cores (b) DGEQRF - Intel64 - 16 cores (c) DGETRF - Intel64 - 16 cores

(d) DPOTRF - Power6 - 16 cores (e) DGEQRF - Power6 - 16 cores (f) DGETRF - Power6 - 16 cores

(g) DPOTRF - Power6 - 32 cores (h) DGEQRF - Power6 - 32 cores (i) DGETRF - Power6 - 32 cores

Figure 2: Effect of (NB,IB) on PLASMA performance (Gflop/s).

parallelization granularity and scheduling flexibility with single core utilization, while the inner block size (IB) trades off memory load with extra-flops due to redundant calculations. Only the QR and LU factorizations use inner blocking. If no inner blocking occurs, the resulting extra-flops overhead may represent 25% and 50% for the QR and LU factorization, respectively [2]. Tuning PLASMA consists of finding the (NB,IB) pairs that maximize the performance depending on the matrix size and on the number of cores. An *exhaustive search* is cumbersome since the search space is huge. For instance, in the QR and LU cases, there are 1352 possible combinations for (NB,IB) even if we constrain NB to be an even integer between 40 and 500 and if we constrain IB to divide NB. All these combinations cannot be explored for large matrices ($N >> 1000$) on effective factorizations in a reasonable time. Knowing that this process should be repeated for each number of cores and each matrix size motivates us to consider a *pruned search*. The idea is that tuning the serial level-3 kernel (dgemm-seq, dssrfb-seq and dssssm-seq) is not time-consuming since peak performance is reached on relatively small input matrices ($NB < 500$) that can be processed fast. Therefore, we first tune those serial kernels. As illustrated in Figure 3, not all the (NB,IB) pairs result in a high performance. For instance, the (480,6) pair leads to a performance of 6.0 Gflop/s whereas the (480,96) pair achieves 12.2 Gflop/s,



(a) dgemm-seq - Intel64

(b) dssrfb-seq - Intel64

(c) dssssm-seq - Intel64

(d) dgemm-seq - Power6

(e) dssrfb-seq - Power6

(f) dssssm-seq - Power6

Figure 3: Effect of (NB,IB) on the performance of the serial PLASMA computational intensive level-3 BLAS kernels (Gflop/s).

for the dssrfb-seq kernel on Power6 (Figure 3(e)). We select a limited number of (NB,IB) pairs (*pruning* step) that achieve a local maximum performance on the range of NB. We have selected five or six pairs on the Intel64 machine for each factorization and eight on the Power6 machine (Figure 3). We then benchmark the performance of PLASMA factorizations only with this limited number of combinations (as seen in Figure 2). Finally, the best performance obtained is selected.

The dssssm-seq efficiency depends on the amount of pivoting performed. The average amount of pivoting effectively performed during a factorization is matrix-dependent. Because the test matrices used for our LU benchmark are randomly generated with a uniform distribution, the amount of pivoting is likely to be important. Therefore, we have selected the (NB,IB) pairs from dssssm-seq executions with full pivoting (figures 3(c) and 3(f)). The dssssm-seq performance drop due to pivoting can reach more than 2 Gflop/s on Power6 (Figure 3(f)).

We have validated our *pruned search* methodology for the three one-sided factorizations on Intel64 16 cores. To do so, we have measured the relative performance overhead (percentage) of the pruned search (PS) over the exhaustive search (ES), that is: $100 \times (\frac{ES}{PS} - 1)$. Table 1 shows that the pruned search performance overhead is bounded by 2%.

Table 1: Overhead (in %) of Pruned search (Gflop/s) over Exhaustive search (Gflop/s) on Intel64 16 cores

	DPOTRF			DGEQRF			DGETRF		
Matrix Size	Pruned Search	Exhaustive Search	Over-head	Pruned Search	Exhaustive Search	Over-head	Pruned Search	Exhaustive Search	Over-head
1000	53.44	52.93	-0.95	46.35	46.91	1.20	36.85	36.54	-0.84
2000	79.71	81.08	1.72	74.45	74.95	0.67	61.57	62.17	0.97
4000	101.34	101.09	-0.25	93.72	93.82	0.11	81.17	80.91	-0.32
6000	108.78	109.21	0.39	100.42	100.79	0.37	86.95	88.23	1.47
8000	112.62	112.58	-0.03	102.81	102.95	0.14	89.43	89.47	0.04

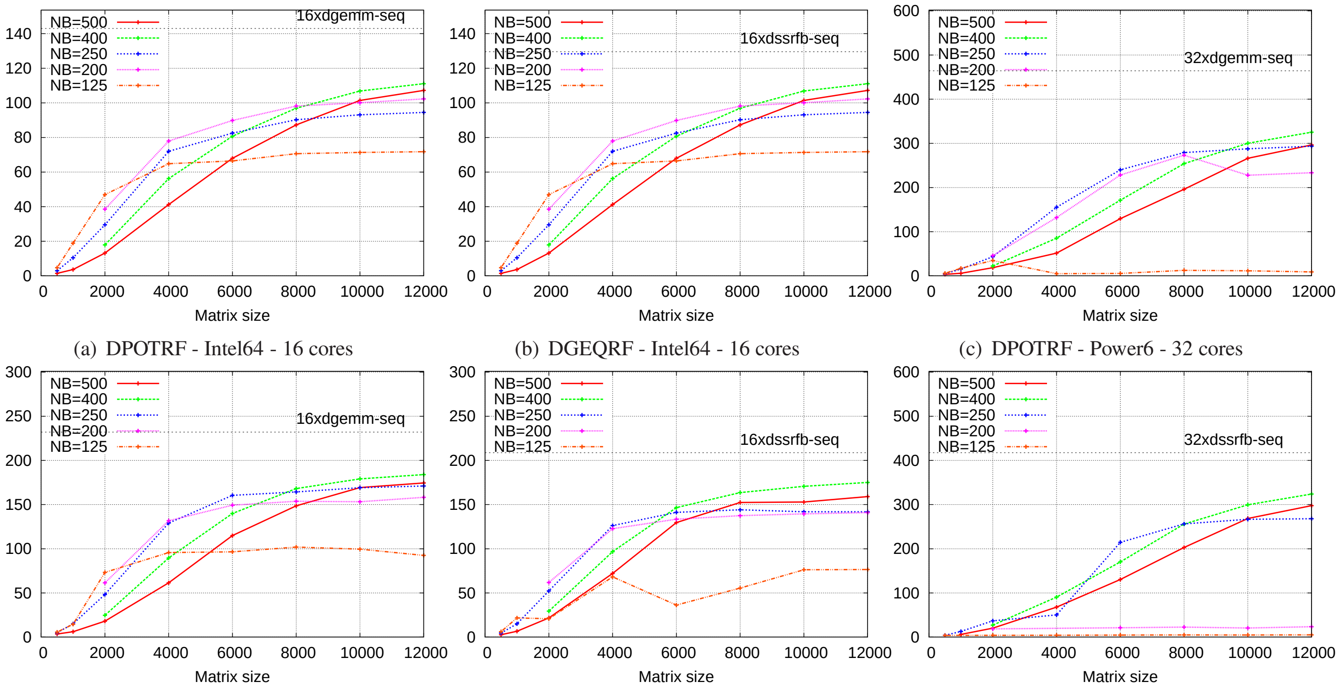
Because the performance may slightly vary from one run to another on cache-based architectures [19], we could furthermore observe in some cases higher performance (up to 0.95%) with pruned search (negative overheads in Table 1). However, the (NB,IB) pair that leads to the highest performance obtained with one method consistently matches the pair leading to the highest performance obtained with the other method. These results validate our approach. Therefore, in the following, all the experimental results presented for PLASMA will correspond to experiments conducted with a *pruned search* methodology.

3.2 TBLAS

Like PLASMA, TBLAS performance relies on two tunable parameters, NB and IB. However, preliminary experimental results led the authors to systematically set IB equal to 20% of NB (for the QR factorization). Figure 4 shows the effect of NB on TBLAS performance. In its current development stage, TBLAS requires that NB divides the matrix size N; this constraint reduces the search space and makes it possible to perform an exhaustive search. In the graphs, the TBLAS results presented correspond to experiments conducted with an *exhaustive search* (and the best performance is reported for each matrix size and each number of cores considered).

3.3 ScaLAPACK and PESSL

ScaLAPACK and PESSL are based on block-partitioned algorithms and the matrix is distributed among a processor grid. The performance thus depends on the the topology of the processor grid and on the block size (NB). For instance, when using 32 cores, the grid topology can be 8x4 (8 rows, 4 columns), 4x8, 2x16, 16x2, 1x32 or 32x1. Four possible block sizes are commonly used: 32, 64, 128 and 256. Figures 5 show the effect of the block size NB on the performance of ScaLAPACK and PESSL (we report the maximum performance obtained for the different possible processor grid configurations). Because there is a limited number of possible combinations, in the following we will report performance results obtained with an exhaustive search: for each matrix size and number of cores, all the possible processor grid distributions are tried in combination with the four block sizes ($NB \in \{32, 64, 128, 256\}$). And the highest performance is finally reported.



(a) DPOTRF - Intel64 - 16 cores
(b) DGEQRF - Intel64 - 16 cores
(c) DPOTRF - Power6 - 32 cores
(d) DPOTRF - Power6 - 16 cores
(e) DGEQRF - Power6 - 16 cores
(f) DGEQRF - Power6 - 32 cores

Figure 4: Effect of NB on TBLAS performance (Gflop/s).

3.4 LAPACK, MKL and ESSL

In LAPACK, the blocking factor (NB) is hard coded in the auxiliary routine ILAENV [20]. This parameter has been set to 64 for both LU and Cholesky factorizations and to 32 for QR factorization[3].

MKL and ESSL have been highly optimized by their respective vendors.

4 Comparison to other libraries

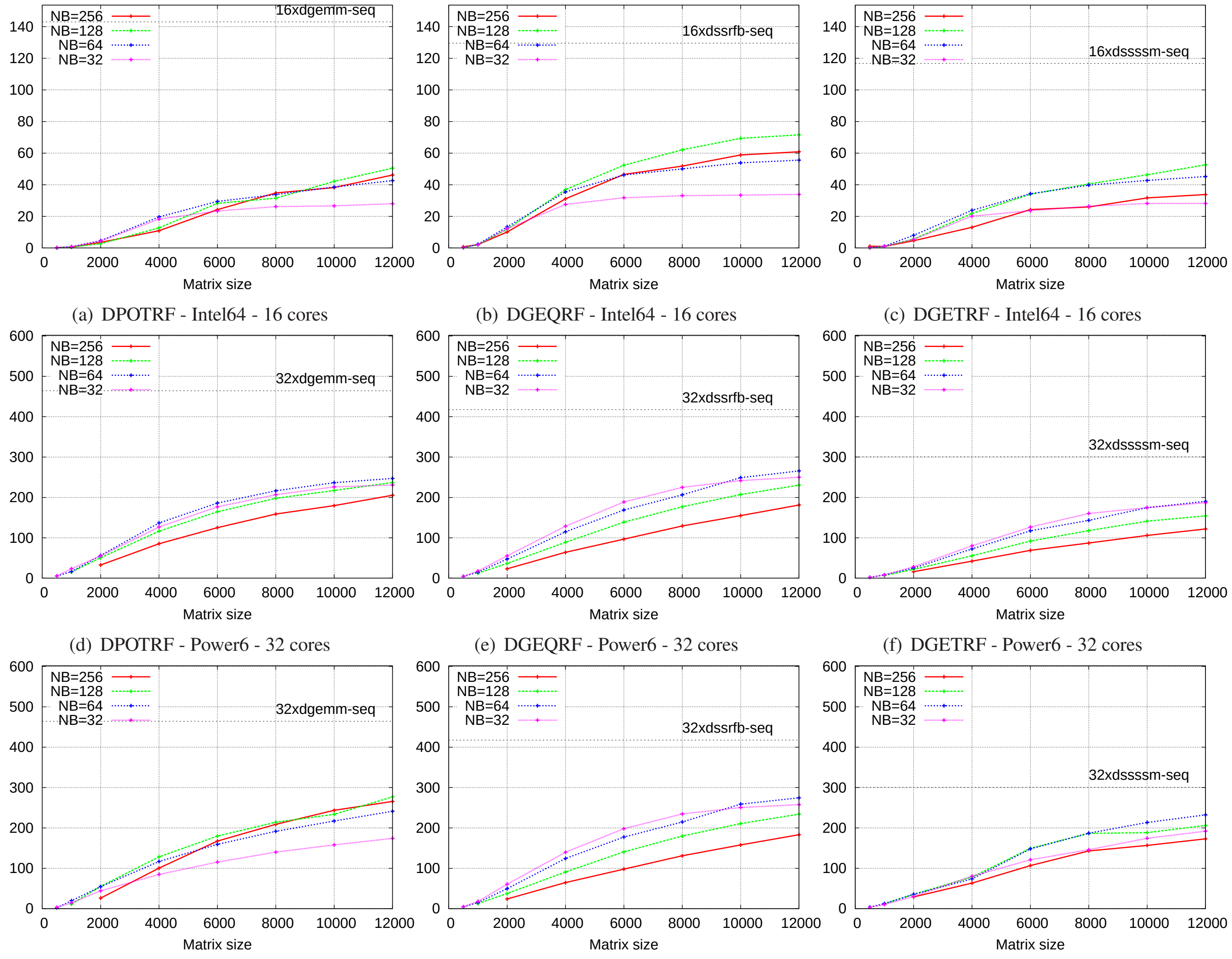
4.1 Methodology

Factorizations are performed in double precision. We briefly recall the tuning techniques used in the experiments discussed below according to Section 3. We employ a pruned search for PLASMA; an exhaustive search for TBLAS, ScaLAPACK and PESSL; and we consider that 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 [21] before timing a factorization[4]. However, the TLB (Translation Lookaside Buffer) is flushed between two executions: the loop over the different

[3] These values may not be optimum for all the test cases. Another approach might have consisted in tuning NB for each number of cores and each matrix size. However, since we did not expect this library to be competitive, we did not investigate that possibility further.

[4] It is kernel usage, not problem size, that dictates whether one wish to flush the cache [21]. Warm (or partially warm) cache executions are plausible for dense linear factorizations. For instance, sparse linear solvers, which rely on dense kernels, intend to maximize data reuse between successive calls to dense operations.



(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

(g) DPOTRF - Power6 - 32 cores (h) DGEQRF - Power6 - 32 cores (i) DGETRF - Power6 - 32 cores

Figure 5: Effect of NB on ScaLAPACK (a,b,c,d,e,f) and PESSL (g,h,i) performance (Gflop/s).

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

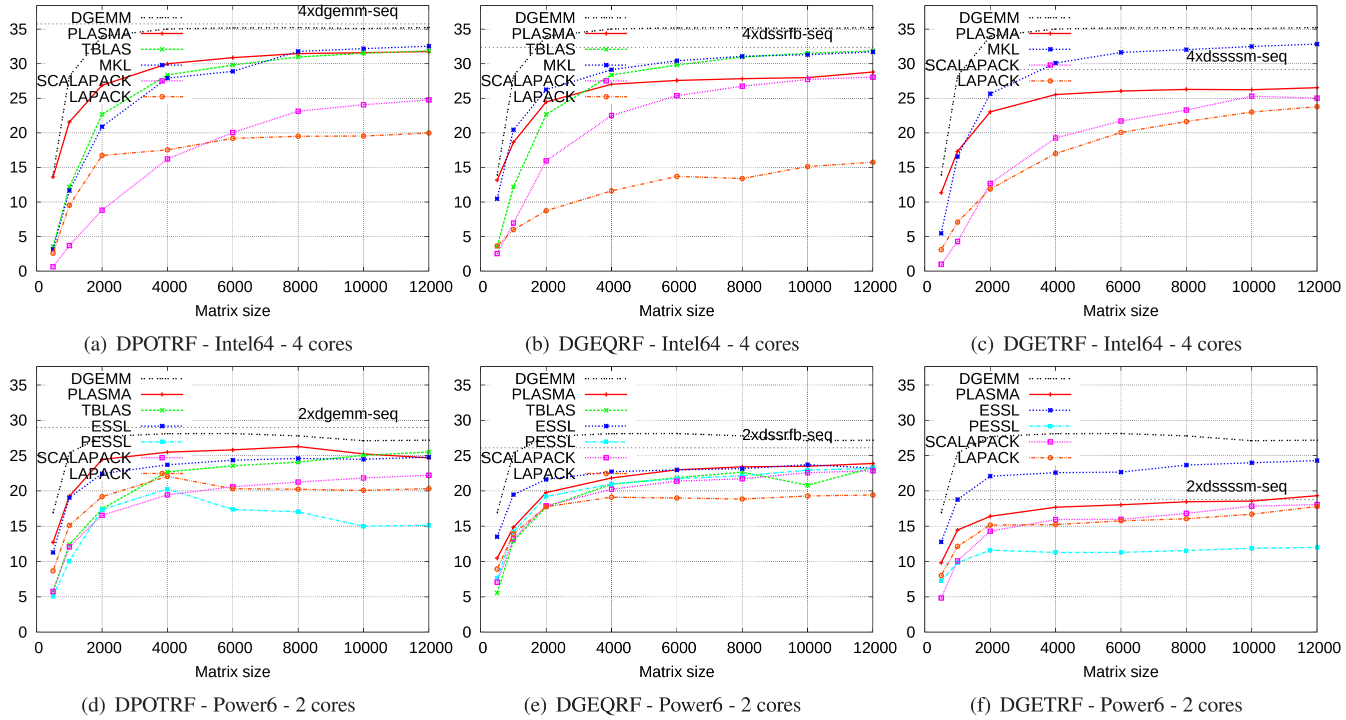
ScaLAPACK, PESSL and PLASMA interfaces allow the user to provide data distributed on the cores. In our shared-memory multi-core environment, because we do not flush the caches, these libraries have thus the advantage to start the factorization with part of the data distributed on the caches. This is 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.

4.2 Experiments on few cores

We present in this section results of experiments conducted on a single socket (4 cores on Intel64 and 2 cores on Power6). In these conditions, data reuse between cores is likely to be efficient since the cores share different levels of cache (see Section 2.2). Therefore, we may expect a high speedup (see definition in Section 2.3) for the different algorithms. Figures 6(a) and 6(d) indeed show that parallel DGEMM – given as a reference – almost reaches a linear speedup on both platforms ($s_4 \approx 4$ on Intel64 and $s_2 \approx 2$ on Power6).

Even LAPACK has a decent speedup. In the DPOTRF case, most of the computation is spent by the dgemm-seq kernel. Since LAPACK exploits parallelism at the BLAS level, the parallel DGEMM version is actually executed. Figures 6(a) and 6(d) show that LAPACK DPOTRF efficiency follows the one of the parallel DGEMM at a lower but decent rate. On Intel64, its performance stabilizes at 20.0 Gflop/s, which corresponds to a normalized efficiency $e_4 = 0.56$ (according to the definition provided in Section 2.3). Parallelism is obtained by processing each BLAS task with all the cores. However, since those tasks are processed in sequence, this approach is very synchronous and does not scale with the number of cores. Indeed, Figure 12(a) shows that LAPACK DPOTRF does not benefit from any significant performance improvement anymore when more than 4 cores (*i.e.* more than one socket) are (is) involved. On Power6, the LAPACK DPOTRF performance also stabilizes at 20.0 Gflop/s (on 2 cores this time, see Figure 6(d)) which corresponds to an even higher normalized efficiency $e_2 = 0.69$. The very high bandwidth (80 MB/s per core) of the L3 shared cache strongly limits the overhead due to this synchronous approach. Furthermore, the high memory bandwidth (75 GB/s shared bus) allows a non negligible improvement of the LAPACK DPOTRF when going to more than 2 cores – and up to 16 – as illustrated by Figure 12(d). However, the performance hardly exceeds 100 Gflop/s, which is only one sixth of the theoretical peak of the 32 cores node. Similar observations can be reported from Figures 6 and 12 for the LAPACK DGEQRF and DGETRF routines (except that a non negligible performance improvement can be obtained up to 8 cores in the LAPACK DGETRF case on Intel – see Figure 12(a)). In a nutshell, parallelism at the BLAS level can achieve a decent performance when the cores involved are limited to a single socket.

Figures 6(c) and 6(f) show that the vendor libraries (MKL and ESSL) outperform the other libraries – including PLASMA– when the LU factorization (DGETRF routine) is executed on a single socket. They even outperform the upper bound of the tile algorithms. Indeed, geometrically, their graphs are above the 4×dssssm-seq (*resp.* 2×dssssm-seq) dashed lines and, numerically, their peak normalized efficiency $e_4 = 1.13$ (*resp.* $e_2 = 1.29$) is higher than 1. This result shows that when the hardware allows a very efficient data reuse across the cores – thanks to the high bandwidth of shared level of caches – the lower performance of the kernels used in the tile algorithms *cannot* be balanced by the better scheduling opportunities that these algorithms provide. The lower performance of tile algorithms kernels is both due to a current non optimized implementation (that we plan to improve) and to the extra-flops performed (that may be decreased at the price of a lower data reuse). The dssssm-seq kernel is the one involving the largest amount of extra-flops for going to tile algorithms (that can lead to a total 50% overhead if no inner blocking occurs). When the extra-flops count is lower, tile algorithms remain competitive. Figures 6(b) and 6(e) illustrate this phenomenon for the QR factorization (DGEQRF routine) whose extra-flops count is bounded by 25% (and less if inner blocking occurs). In this case, tile algorithms (PLASMA and TBLAS) can almost compete against vendor libraries; their better scheduling opportunities enable balancing the extra-flops count (as well as the current non optimized implementation of their



(a) DPOTRF - Intel64 - 4 cores (b) DGEQRF - Intel64 - 4 cores (c) DGETRF - Intel64 - 4 cores

(d) DPOTRF - Power6 - 2 cores (e) DGEQRF - Power6 - 2 cores (f) DGETRF - Power6 - 2 cores

Figure 6: Performance comparison when a single socket is used (Gflop/s).

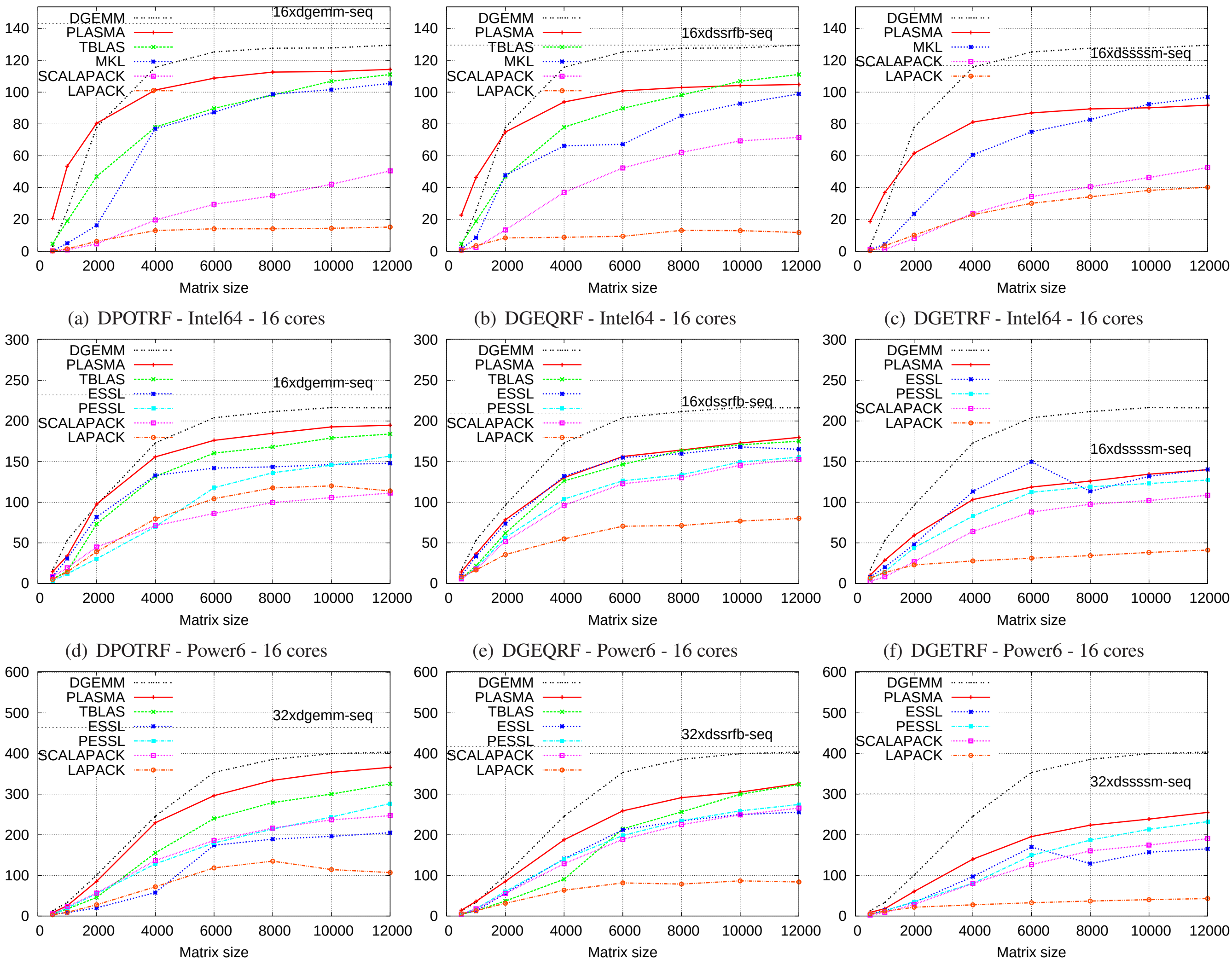
serial kernels) although the hardware efficiency for parallel data reuse limits the impact of a non optimum scheduling. The Cholesky factorization (DPOTRF routine) does not involve extra-flops for going to tile algorithms and directly relies on a kernel optimized by the vendor (dgemm-seq). As illustrated in figures 6(a) and 6(d), in this case, tile algorithms (and in particular PLASMA) outperform the algorithms implemented in the other libraries, including the vendor libraries.

In a nutshell, the overhead of imperfect scheduling strategies is limited when the algorithms are executed on few cores that share levels of cache they can access with a very high bandwidth. In this case, tile algorithms have a higher performance than established linear algebra packages depending on whether the extra-flops count for going to tile and the overhead of a non optimized serial kernel is not too large compared to the limited practical improvement gained by their scheduling opportunities. The advantage of tile algorithms relies on their very good scalability achieved by their scheduling opportunities; figures 8 and 9 show their excellent practical scalability on both machines in the PLASMA and TBLAS cases, respectively. In the next section, we discuss their relative performance against other libraries when a larger number of cores are used.

4.3 Large number of cores

We present in this section results of experiments conducted on a larger number of cores (16 cores on Intel64; 16 and 32 cores on Power6). In these conditions, the cost of data reuse between cores is higher when the cores do not share a common level of cache. For instance, on the Power6 architecture, if a core accesses data that have been processed by another processor, it will access that data through the memory bus, which is shared by all the 32 cores. Even if that bus has a high bandwidth (75 GB/s), the actual throughput is limited when many cores simultaneously access the memory. Therefore, we expect a higher sensitivity of the algorithms to the scheduling strategy than we indicated in Section 4.2.

Figures 7(a), 7(d) and 7(g) 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



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

(d) DPOTRF - Power6 - 16 cores (e) DGEQRF - Power6 - 16 cores (f) DGETRF - Power6 - 16 cores

(g) DPOTRF - Power6 - 32 cores (h) DGEQRF - Power6 - 32 cores (i) DGETRF - Power6 - 32 cores

Figure 7: Performance comparison on a large number of cores (Gflop/s).

reference up to a matrix size $N = 2000$ on the Intel64 machine when 16 cores are used. This is not contradictory since

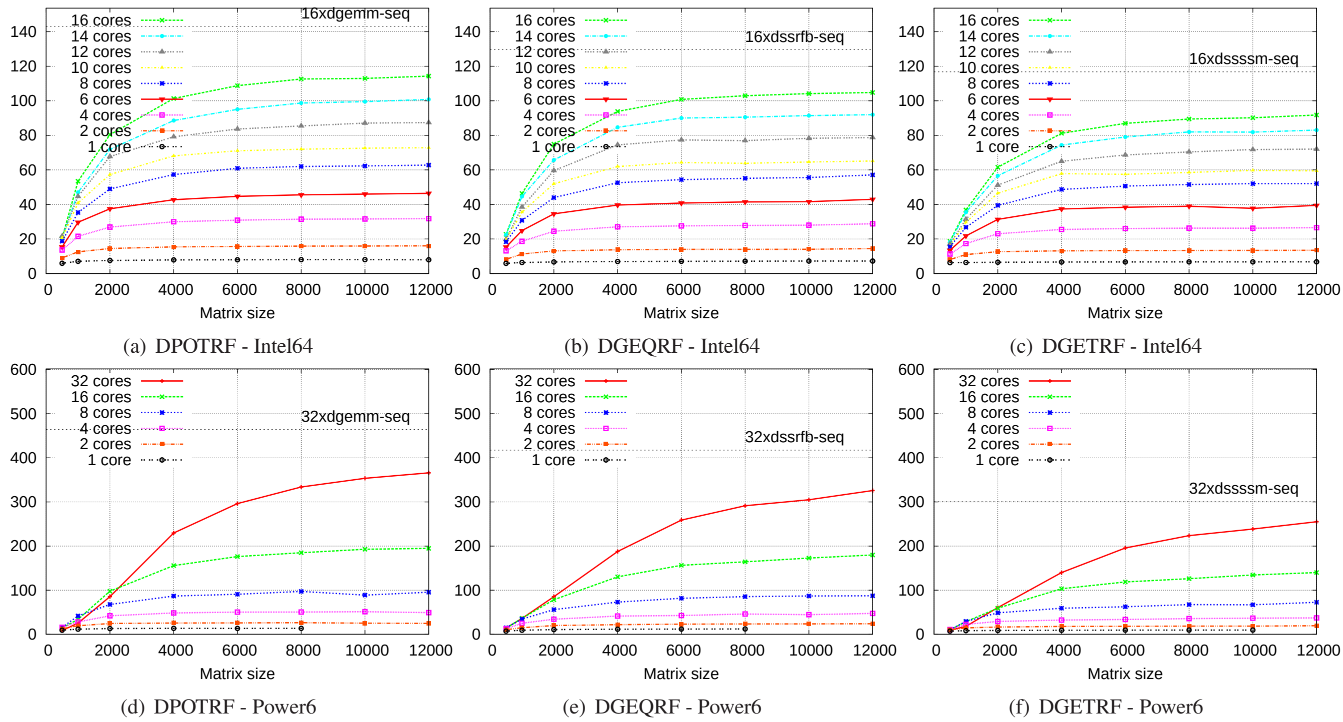
PLASMA does not rely on the parallel version of DGEMM. Each PLASMA thread indeed uses the serial dgemm-seq. Better performance can then be achieved thanks to better scheduling. However, the DPOTRF factorization cannot only be performed with efficient level-3 BLAS operations (it also involves level-2 BLAS operations). Therefore, as expected, the parallel DGEMM dominates all the other operations when the matrix size becomes larger. This illustrates the fact that using a fine enough granularity (as in tile algorithms) is more critical when processing small or moderate size matrices. Indeed, the better load balancing allowed by a fine granularity does not impact the steady state of the DAG processing. This also explains the major improvement brought by PLASMA compared to the other libraries on matrices of small and moderate size.

Still considering the DPOTRF routine, the vendor libraries (MKL and ESSL) remain the most competitive solution versus tile algorithms (but they have a lower efficiency than tile algorithms) compared to ScaLAPACK and LAPACK approaches, up to 16 cores. However, ESSL does not scale well to 32 cores (Figure 11). ScaLAPACK and its vendor equivalent, PESSL, both outperform ESSL on 32 cores (Figure 7(g)). This result generalizes, to some extent, to all the factorization routines. ScaLAPACK and PESSL still scale up to 32 cores (Figure 10) – as opposed to ESSL– and they both (with a slight advantage for PESSL) achieve a performance comparable to ESSL for the DGEQRF (QR) and DGETRF (LU) routines on Power6 when 32 cores are used (Figures 7). Similarly, figures 11(a), 11(b) and 11(c) illustrate the not so good scalability of the vendor library when all the 16 cores of the Intel64 platform are used.

Figures 7(b) and 7(h) 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 TBLAS is also very competitive. These results demonstrate the excellent scalability of tile algorithms and show that it is worth performing a little amount of extra-flops to obtain tasks more convenient to schedule. 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 ($N \geq 10,000$). Indeed, when the matrices processed are large, the critical issue of scheduling 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 to move towards an hybrid scheduling strategy that would assign priorities according to a trade off between data reuse and critical path progress and would process available tasks dynamically. Figure 7(e) illustrates the performance of the QR factorization (DGEQRF routine) when only half of the available cores are used on Power6. In this case, PLASMA still achieves the highest performance but TBLAS and ESSL almost reach a similar performance.

Finally, figures 7(c), 7(f) and 7(i) 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 and strongly improve the performance on large matrices. Indeed, on Intel64 when 16 cores are used, the PLASMA LU peak normalized efficiency (see definition in Section 2.3) is equal to 78.6%; so there is still room for improving the performance by enabling dynamic scheduling. Furthermore, as already mentioned, an optimized implementation of the dssssm-seq kernel will also improve the performance of tile algorithms.



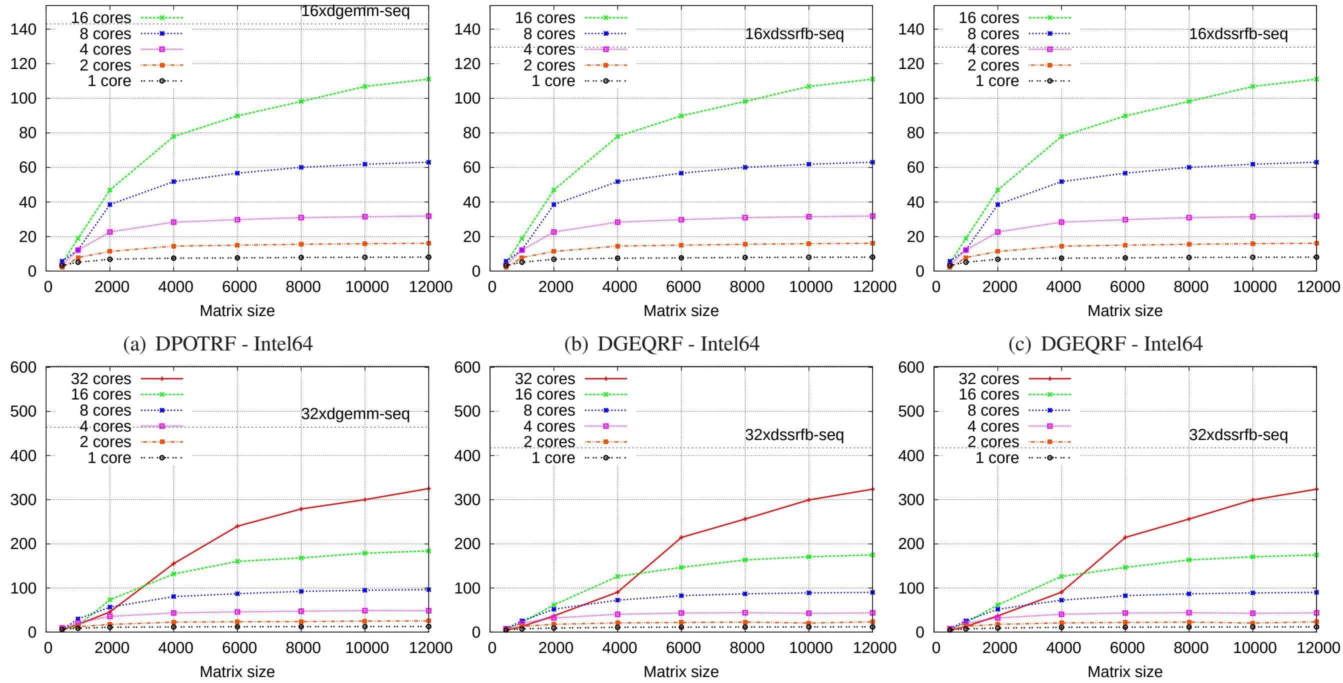
(a) DPOTRF - Intel64
(b) DGEQRF - Intel64
(c) DGETRF - Intel64
(d) DPOTRF - Power6
(e) DGEQRF - Power6
(f) DGETRF - Power6

Figure 8: PLASMA performance (Gflop/s).

5 Conclusion and perspectives

This paper has analyzed and discussed the behavior of several software approaches for achieving high performance and portability of one-sided factorizations on multi-core architectures. In particular, 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. Preliminary experiments using tile algorithms for two-sided transformations, *i.e.*, the Hessenberg reduction [22] (first step for the standard eigenvalue problem) and the bidiagonal reduction [23] (first step for the singular value decomposition), show promising results.

We have shown the efficiency of our pruned-search methodology for tuning PLASMA. However, we currently need



(a) DPOTRF - Intel64 (b) DGEQRF - Intel64 (c) DGEQRF - Intel64
(d) DPOTRF - Power6 (e) DGEQRF - Power6 (f) DGEQRF - Power6

Figure 9: TBLAS performance (Gflop/s).

to manually pick up the (NB,IB) samples (from the serial level-3 kernel benchmarking) that are going to be tested in the parallel factorizations. We are currently working to automate this pruning process. Furthermore, not all the matrix sizes and number of cores can be tested. 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 distribution.

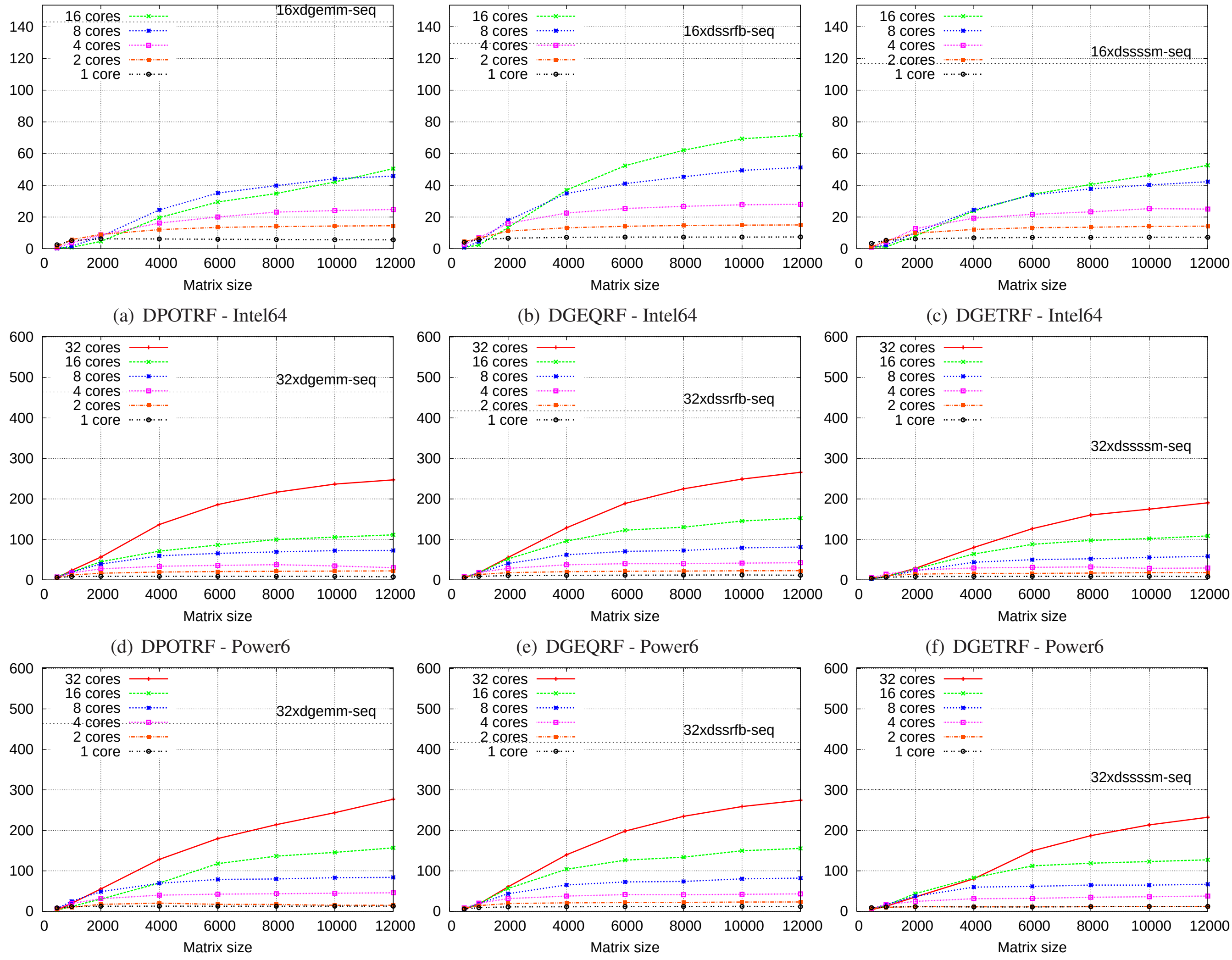
Furthermore, because the factorization performance strongly depends on the computational intensive serial level-3 kernels, their optimization is paramount. Unlike DGEMM, the dssrfb-seq and dssssm-seq kernels are not a single call to level-3 BLAS operations, but are composed of successive calls, since the inefficiency. dssrfb-seq and dssssm-seq could achieve similar performance if implemented as a monolithic code and heavily optimized.

The experiments have also shown the limits of static scheduling for the factorization of large matrices. We are currently working on the implementation of a hybrid scheduling for PLASMA. Even if they are not on the critical path, tasks will be dynamically scheduled on idle cores so as to maximize data reuse.

References

[1] Herb Sutter. A fundamental turn toward concurrency in software. *Dr. Dobb's Journal*, 30(3), 2005.

[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.



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

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

(g) DPOTRF - Power6 (h) DGEQRF - Power6 (i) DGETRF - Power6

Figure 10: ScaLAPACK (a,b,c,d,e,f) and PESSL (g,h,i) performance (Gflop/s).

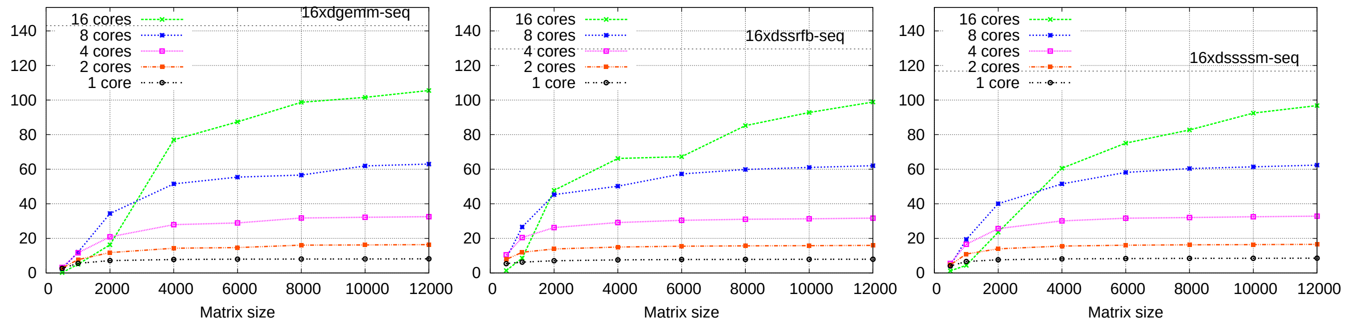
[3] Buttari A., Dongarra J., Kurzak J., Langou J., Luszczek P., and Tomov S. The impact of multicore on math software. *PARA 2006, Umea, Sweden*, June 2006.

[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.

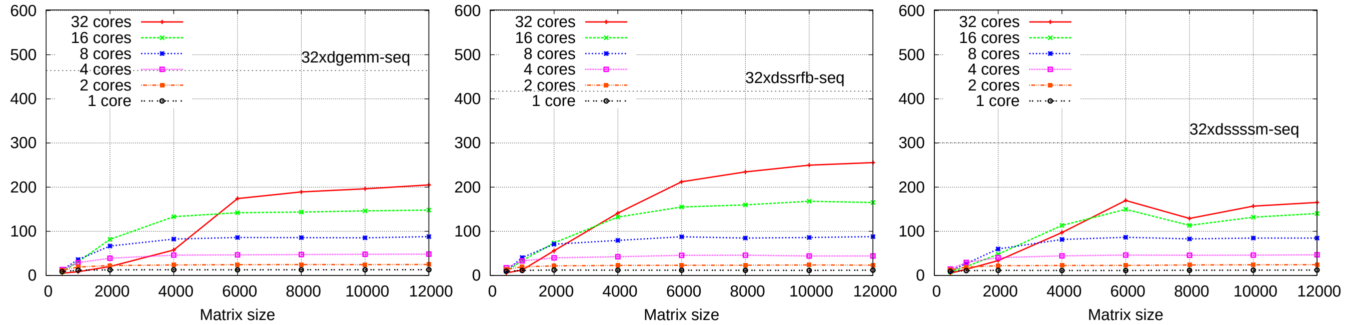
[5] Blackford L. S., Choi J., Cleary A., D'Azevedo E., Demmel J., Dhillon I., Dongarra J., Hammarling S., Henry G., Petitet A., Stanley K., Walker D., and Whaley R. C. *ScaLAPACK Users' Guide*. SIAM, 1997.

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

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

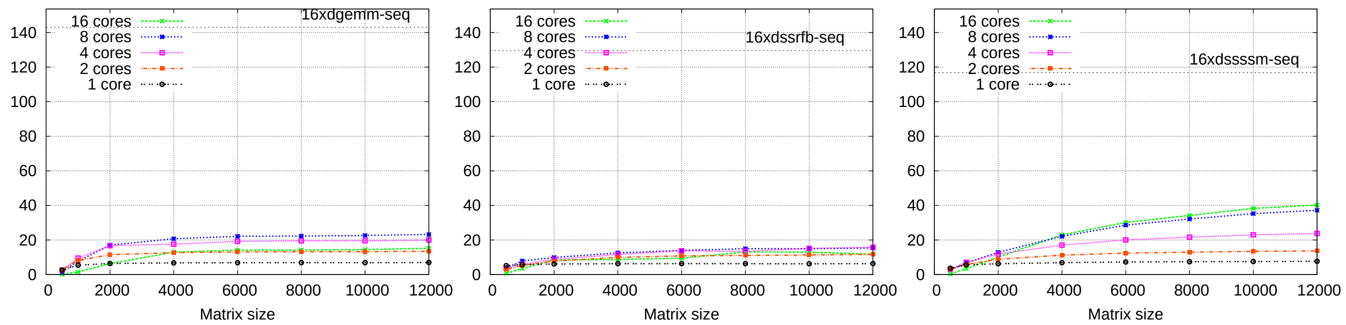


(a) DPOTRF - Intel64 (b) DGEQRF - Intel64 (c) DGETRF - Intel64

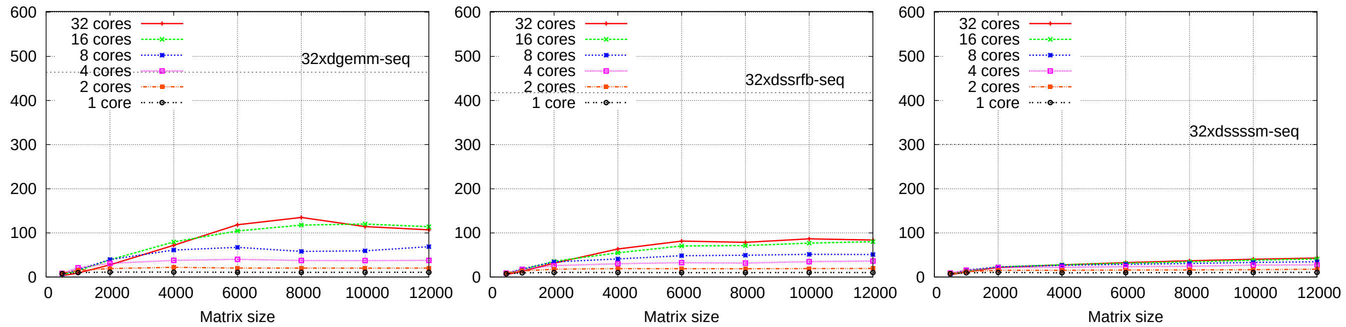


(d) DPOTRF - Power6 (e) DGEQRF - Power6 (f) DGETRF - Power6

Figure 11: MKL (a,b,c) and ESSL (d,e,f) performance (Gflop/s).



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



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

Figure 12: LAPACK performance (Gflop/s).

[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] Dongarra J., Bunch J., Moler C., and Stewart G. W. *LINPACK User's Guide*. SIAM, Philadelphia, PA, 1979.

[10] Garbow B. S., Boyle J. M., Dongarra J. J., and Moler C. B. *Matrix Eigensystem Routines – EISPACK Guide Extension*, volume 51 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 1977.

[11] Smith B. T., Boyle J. M., Dongarra J. J., Garbow B. S., Ikebe Y., Klema V. C., and Moler C. B. *Matrix Eigensystem Routines – EISPACK Guide*. vol 6, Lecture Notes in Computer Science. Springer-Verlag, Berlin, 1976.

[12] Intel, Math Kernel Library (MKL). http://www.intel.com/software/products/mkl/.

[13] IBM, Engineering and Scientific Subroutine Library (ESSL) and Parallel ESSL. http://www-03.ibm.com/systems/p/software/essl/.

[14] Gropp W., Lusk E., and Skjellum A. *Using MPI: portable parallel programming with the message-passing interface*. MIT Press, Cambridge, MA, USA, 1994.

[15] PBLAS: Parallel basic linear algebra subprograms. http://www.netlib.org/scalapack/pblas_qref.html.

[16] BLACS: Basic linear algebra communication subprograms. http://www.netlib.org/blacs/.

[17] SMP Superscalar. http://www.bsc.es/ → Computer Sciences → Programming Models → SMP Superscalar.

[18] Meuer H., Strohmaier E., Dongarra J., and Simon H. *TOP500 Supercomputer Sites*, 32$^{\text{nd}}$ edition, November 2008. (The report can be downloaded from http://www.netlib.org/benchmark/top500.html).

[19] Sloss A., Symes D., and Wright C. *ARM System Developer's Guide: Designing and Optimizing System Software*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2004.

[20] Whaley R. C. Empirically tuning lapacks blocking factor for increased performance. *Computer Aspects of Numerical Algorithms, Wisla, Poland*, October 20-22, 2008.

[21] 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.

[22] Ltaief H., Kurzak J., and Dongarra J. Parallel block hessenberg reduction using algorithms-by-tiles for multicore architectures revisited. *University of Tennessee CS Tech. Report, UT-CS-08-624 (also LAWN #208)*, 2008.

[23] Ltaief H., Kurzak J., and Dongarra J. Parallel band two-sided matrix bidiagonalization for multicore architectures. *University of Tennessee CS Tech. Report, UT-CS-08-631 (also LAWN #209)*, 2008.