A Comprehensive Study of Task Coalescing for Selecting Parallelism Granularity in a Two-Stage Bidiagonal Reduction

Azzam Haidar*, Hatem Ltaief†, Piotr Luszczek*, and Jack Dongarra*
*Innovative Computing Laboratory, University of Tennessee, Knoxville, TN 37996, USA
Email: haidar,luszczek,dongarra@eecs.utk.edu
†KAUST Supercomputing Laboratory, Thuwal, Saudi Arabia
Email: Hatem.Ltaief@kaust.edu.sa

***Abstract*—We present new high performance numerical kernels combined with advanced optimization techniques that significantly increase the performance of parallel bidiagonal reduction. Our approach is based on developing efficient fine-grained computational tasks as well as reducing overheads associated with their high-level scheduling during the so-called bulge chasing procedure that is an essential phase of a scalable bidiagonalization procedure. In essence, we coalesce multiple tasks in a way that reduces the time needed to switch execution context between the scheduler and useful computational tasks. At the same time, we maintain the crucial information about the tasks and their data dependencies between the coalescing groups. This is the necessary condition to preserve numerical correctness of the computation. We show our annihilation strategy based on multiple applications of single orthogonal reflectors. Despite non-trivial characteristics in computational complexity and memory access patterns, our optimization approach smoothly applies to the annihilation scenario. The coalescing positively influences another equally important aspect of the bulge chasing stage: the memory reuse. For the tasks within the coalescing groups, the data is retained in high levels of the cache hierarchy and, as a consequence, operations that are normally memory-bound increase their ratio of computation to off-chip communication and become compute-bound which renders them amenable to efficient execution on multicore architectures. The performance for the new two-stage bidiagonal reduction is staggering. Our implementation results in up to 50-fold and 12-fold improvement ($\sim$ 130 Gflop/s) compared to the equivalent routines from LAPACK V3.2 and Intel MKL V10.3, respectively, on an eight socket hexa-core AMD Opteron multicore shared-memory system with a matrix size of 24000×24000. Last but not least, we provide a comprehensive study on the impact of the coalescing group size in terms of cache utilization and power consumption in the context of this new two-stage bidiagonal reduction.**

***Keywords*-Bidiagonal Reduction; Two-Stage Approach; Tile Algorithms; Bulge Chasing; Granularity Analysis; Power Profiling; Dynamic Scheduling;**

I. Introduction

The bidiagonal reduction (BRD) is an important first step when calculating the singular value decomposition (SVD) of a general rectangular dense matrix [1]–[3]. Various methods of obtaining such reductions have been presented before [4], [5]. Invariably, they all follow the decompositional approach to dense matrix computation [6]. In particular, the new two-stage BRD introduced in this paper is follow-up to previous numerical algorithms for one-sided and two-sided transformations based on the tile data layout format associated with the dynamic runtime environment system QUARK [7], in the context of the PLASMA library [8]–[11].

The whole two-stage BRD computation proceeds as follows. The dense matrix is first reduced to a band bidiagonal form using compute-intensive kernels. The extra off-diagonal elements are then annihilated using a bulge chasing procedure during a second stage, which is characterized by memory-bound kernels. While the parallelization of the first stage is straightforward and highly efficient, most of the performance bottlenecks reside in the scheduling of the fine-grained computational tasks involved in the bulge chasing phase. Indeed, there is a clear trade-off in the second stage between the degree of parallelism and the resulting overall performance – the larger the number of tasks the more opportunities for parallelism. Also, the larger the number of tasks the larger is the associated scheduling overhead due to low computational intensity. This may completely overshadow any gains from parallel execution and, in addition, it may excessively strain the memory bus if appropriate care is not taken into consideration when scheduling the tasks with nearly optimal core affinity to allow for cache reuse.

Our possible answer to solve the aforementioned problem is twofold. We provide cache-aware fine-grained annihilation kernels based on Householder reflectors [12], which chases down single elements at a time. Highly optimized for cache reuse, these kernels retain the data in high levels of the cache memory hierarchy and, as a consequence, operations that are originally memory-bound increase their ratio of computation to off-chip communication and become compute-bound which renders them amenable to efficient execution on multicore architectures. And this cache behavior is further enhanced thanks to the integration of advance optimization techniques based on task coalescing, where multiple standalone tasks are grouped together to work on the same, most likely shared data. We present the effect of the task group size on the cache utilization, the overall execution performance as well as the power consumption. All in all, the new two-stage BRD implementation results in up to 50-

fold and 12-fold improvement ($\sim$ 130 Gflop/s) compared to the equivalent routines from LAPACK V3.2 and Intel MKL V10.3, respectively, on an eight socket hexa-core AMD Opteron multicore shared-memory system with a matrix size of 24000×24000.

The remainder of this paper is organized as follows: Section II gives a detailed overview of previous research works in this area. Section III clearly describes our research contributions. Section IV recalls the two-stage BRD approach using tile algorithms. Section V presents the fine-grained and cache-friendly numerical kernels used during both stages associated with the dynamic scheduler QUARK. Section VI explains the importance of the coalescing group size. The impact of the coalescing group size on cache behavior and performance is shown in Section VII, comparing our implementation with the state-of-the-art, high performance dense linear algebra software libraries, LAPACK V3.2 [13] and Intel MKL V10.3 [14], an open-source and a commercial package, respectively. Section VIII demonstrates the impact of the coalescing group size on power consumption. Finally, Section IX summarizes the results of this paper and presents some future work.

II. Related Work

The standard approach from LAPACK [13] is to use a single phase to reduce a general (and possibly symmetric) matrix to a special form. This includes reductions to Hessenberg, tridiagonal, and bidiagonal forms each of which is a similarity matrix for general or symmetric eigenvalue problems or the singular value computation. This has been challenged by the method of splitting the reduction phase to condensed forms into multiple stages.

One of the first uses of a two-step reduction occurred in the context of out-of-core solvers for generalized symmetric eigenvalue problems [15]. Then, a multi-stage method was used to reduce a matrix to tridiagonal, bidiagonal and Hessenberg forms [16]. The number of stages necessary to reduce the matrix to the corresponding form was no longer fixed but was instead a tunable parameter, which depended on the specifics of the underlying hardware parameters. With this approach, it was possible to recast the expensive memory-bound operations, that occur during the panel factorization into a compute-bound procedure.

Consequently, a framework called Successive Band Reductions (SBR) was created [17], that integrated some of the multi-stage work. SBR toolbox may be used to reduce a symmetric dense matrix to tridiagonal form, required to solve the symmetric eigenvalue problem (SEVP). The toolbox applies two-sided orthogonal transformations based on Householder reflectors and successively reduces the matrix bandwidth size until a suitable width is reached. The off-diagonal elements are then annihilated column-wise, which produces large fill-in blocks or bulges that need to be chased down toward the bottom right corner of the matrix. The bulge chasing procedure may result in substantial increase in the floating point operation count when compared with the standard single-phase approach from LAPACK. If eigenvectors are required in addition to eigenvalues, then the transformations may be efficiently accumulated using Level 3 BLAS operations to generate these vectors. It is also noteworthy to mention that SBR heavily relies on multithreaded BLAS that are optimized to achieve satisfactory parallel performance. However, such parallelization paradigm incurs substantial overheads [10] as it fits the Bulk Synchronous Parallelism (BSP) model [18]. Communication bounds for such two-sided reductions have been established under the Communication Avoiding framework [5]. A multi-stage approach has also been applied to the Hessenberg reduction [19] as well as the QZ algorithm [20] for the generalized non symmetric eigenvalue problem. These approaches, in contrast to our own, use neither tile algorithms nor tile storage.

Tile algorithms have recently seen a rekindled interest when applied to the two-stage tridiagonal [10] and bidiagonal reductions [4]. Using high performance kernels combined with a data translation layer to execute on top of the tile data layout format, both implementations achieve a substantial improvement compared to the equivalent routines from the state-of-the-art numerical libraries. The off-diagonal elements are annihilated column-wise instead during the bulge chasing procedure, which engenders significant extra-flops due to the size of the bulges introduced. An element-wise annihilation has then been implemented based on cache-aware kernels, in the context of the two-stage tridiagonal reduction [9]. Using the coalescing task technique presented in this paper, the performance achieved is by far greater than any other available implementations.

The next Section highlights our main research contributions in this paper.

III. Core Contributions

The aim of this Section is to clearly distinguish our four main contributions:

- New high performance computational kernels based on element-wise annihilation have been implemented to efficiently tackle the second stage (i.e., the bulge chasing procedure) of the two-stage BRD algorithm. Running on top of tile data layout format, the kernels are optimized for cache reuse thanks to the fine-grained computation. An advanced optimization strategy, which consists of coalescing the fine-grained tasks, is employed to drastically remove the scheduler overhead as well as to enhance the memory reuse.
- The impact of the coalescing task size on the cache utilization has been studied through the performance counter library PAPI [21].
- The impact of the coalescing task size on the overall execution performance has been demonstrated on a

cutting-edge shared-memory multicore system.
- The impact of the coalescing task size on the algorithm power consumption has been analyzed using the PowerPack framework [22].

The combination of new highly optimized kernels and coalescing task techniques represent the crux of the research work presented in this paper.

The next Section recalls the two-stage approach for BRD using tile algorithms.

IV. Two-Stage Bidiagonal Reduction using Tile Algorithms

This Section recalls the principles of tile algorithms as well as the two-stage approach and how both can be applied in the context of BRD.

A. Tile Algorithm Principles

Tile algorithms consist of splitting the dense matrix into tiles. Each of the tiles is contiguous in memory, as opposed to the standard column-major data layout format (see Figure 1) in which only a single column is contiguous in memory and may not be use effectively for cache-blocked operations. The numerical algorithm proceeds by taking into account this high performance data storage, which may require the redesign of the original algorithm. In fact, not only the tile data layout format allows to substantially reduce the cache and TLB misses but it also allows to generate a tremendous amount of concurrent tasks. The program data flow can then be represented as a directed acyclic graph (DAG), where tasks represent nodes and edges – data dependencies between tasks. An efficient runtime environment system is necessary to schedule such computational tasks among the available processing units (see Section V-B).

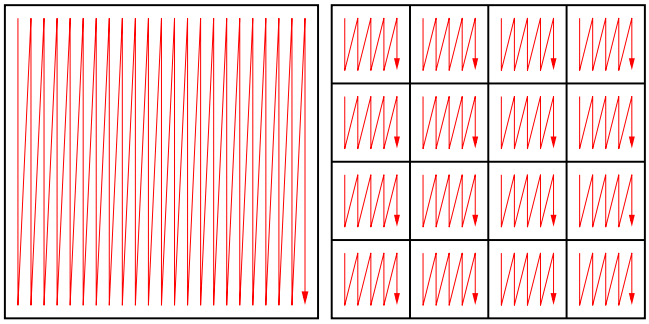


Figure 1. Column-major (left) and tile data layout (right) for a matrix.

B. Two-Stage Reduction Approach

As previously shown in Section II, the two-stage reduction approach has been extensively studied in the context of block algorithms (e.g., SBR [17]) and more recently with tile algorithms (e.g., tridiagonal [9], [10] and bidiagonal reductions [4]). The core idea consists of reducing the dense matrix to band form using compute-intensive kernels. This stage has also the particularity to be highly parallel by benefiting from the parallelism brought to the fore thanks to tile algorithms. The band form is further reduced to the final condensed form using the bulge chasing technique. This procedure annihilates the extra off-diagonal elements by chasing the created fill-in elements down to the bottom right side of the matrix using successive orthogonal transformations at each sweep. The condensed form is eventually computed and the corresponding eigenvalue or singular value solver can then be initiated.

C. Application to BRD Algorithm

The difficulty now is to map the two-stage approach with tile algorithms. While the first stage can entirely run on tile data layout format, there is however a mismatch for the bulge chasing procedure. The different orthogonal operations may span over some portions of multiple tiles and therefore, data dependencies need to be cautiously tracked to avoid any violations, which would jeopardize the numerical correctness. Multiple frameworks have already been implemented to handle such critical situations, namely the data translation layer in Section VI A and Section IV from [4], [10], respectively, and the function dependencies in Section 8.2 from [9]. Although the data translation layer expresses the data dependencies more naturally, the function dependencies framework is preferred in this paper because it is less restrictive and does not require an atomic access whenever a particular task spans across multiple tiles. Moreover, following the annihilation strategy introduced in [9], the extra off-diagonal entries are rather zeroed out element-by-element, as opposed to column-wise annihilation from [4]. Thus, the size of the fill-in structure during the bulge chasing procedure is severely reduced, which considerably engenders less floating-point operations. At the same time, the kernel computation intensity using element-wise annihilation decreases and in this case, it becomes paramount to make sure that the actual computation happens at the high level caches to balance the loss of computation intensity.

The next Section describes the high performance numerical kernels involved in the two-stage BRD tile algorithm.

V. Implementations Details

This Section recalls the numerical compute-bound kernels of the first stage (reduction to band bidiagonal form) and introduces the new high performance fine-grained and cache-friendly computational kernels involved in the second stage (reduction to condensed bidiagonal form).

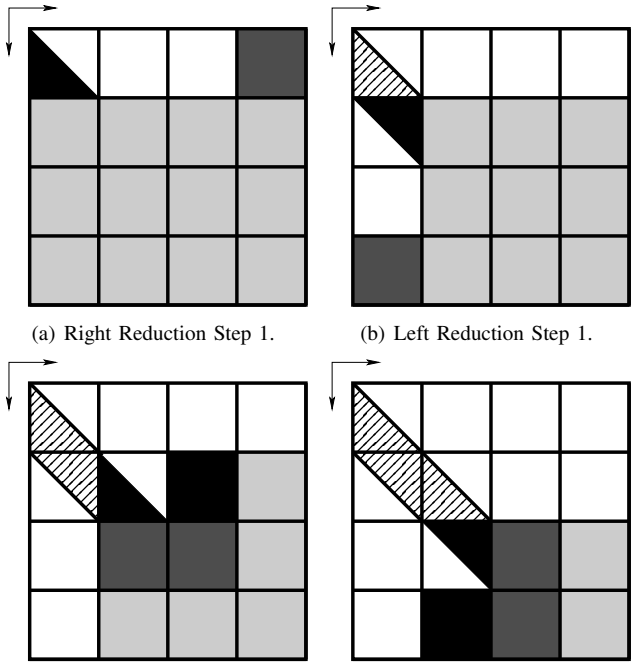
A. Kernel Descriptions

1) First Stage Kernels: These kernels are available from previous BRD implementations (see Section III A from [23]). Therefore, the purpose of this subsection is only to make the paper self-contained. This phase basically

interleaves the QR and LQ factorizations at each step and, all together, uses six compute-intensive kernels:

- CORE_DGEQRT/CORE_DGELQT compute a QR and an LQ factorizations of a single tile, respectively.
- CORE_DTSQRT/CORE_DTSLQT perform a QR and an LQ factorizations by combining a triangular tile (upper if QR, lower if LQ) with a corresponding full square tile. CORE_DTSQRT and CORE_DTSLQT are shown in Figure 2(a) and Figure 2(b), respectively.
- CORE_DORMQR/CORE_DORMLQ apply the orthogonal transformations computed from CORE_DGEQRT/CORE_DGELQT to the left/right side of the trailing submatrix.
- CORE_DTSMQR/CORE_DTSMLQ apply the orthogonal transformations computed from CORE_DTSQRT/CORE_DTSLQT to the left/right side of the trailing submatrix. The right and left applications from CORE_DTSMQR/CORE_DTSMLQ are laid out in Figure 2(c) and Figure 2(d) (the black and dark grey data tiles), respectively.

In terms of extra storage, only the triangular factor T, that was generated from the block reflectors [24], needs to be saved to eventually apply Level 3 BLAS operations, during the update of the trailing submatrix that has not been reduced yet.



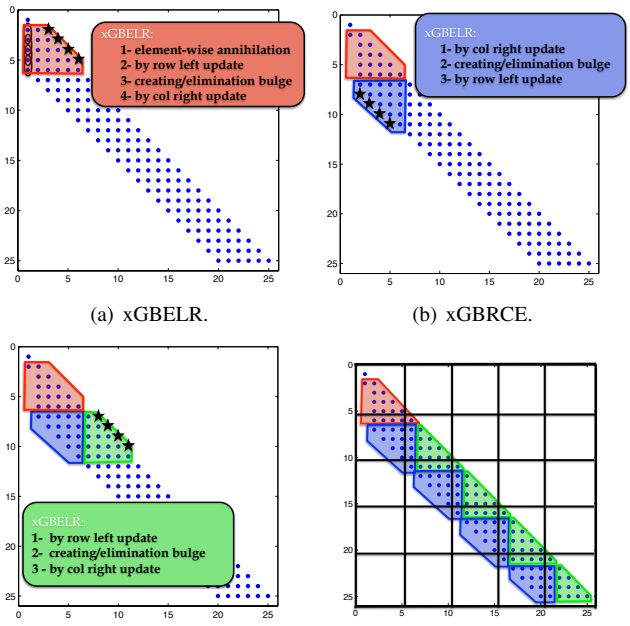
(a) Right Reduction Step 1. (b) Left Reduction Step 1.

(c) Right Reduction Step 2. (d) Left Reduction Step 2.

Figure 2. First stage: reduction to band bidiagonal form applied on a 4x4 tile matrix.

2) Second Stage Kernels: We have designed a novel bulge chasing algorithm (similar to our previous implementation [9]) based on three new kernels, which allow us to considerably enhance the data locality:

- The **xGBELR** kernel: this kernel triggers the beginning of each sweep by successive element-wise annihilations of the extra non-zero entries within a single column, as shown in Figure 3(a). It then applies *all* the left updates creating single bulges, which have to be immediately annihilated and then followed by the right updates on the corresponding data block loaded into the cache memory.
- The **xGBRCE** kernel: this kernel successively applies *all* the right updates coming from the previous kernel, either xGBELR or xGBLRX (described below). This subsequently generates single bulges, which have to be immediately annihilated by appropriate left transformations in order to eventually avoid an expansion of the fill-in structure (Figure 3(b)) by subsequent orthogonal transformations.
- The **xGBLRX** kernel: this kernel successively applies *all* the left updates (coming from the xGBRCE kernel) and create single bulge out of the diagonal, then similarly to xGBELR, it eliminate the bulge and apply the corresponding right updates, as depicted in (Figure 3(c)).



(a) xGBELR. (b) xGBRCE.

(c) xGBLRX. (d) Unrolling the kernel calls within one sweep.

Figure 3. First sweep on top of tile data layout of our bulge chasing procedure using the new memory-aware kernels.

Algorithm 1 represents the bulge chasing procedure, which reduces the band bidiagonal form to condensed form using element-wise annihilation.

Once the computational kernels are defined, the key is to efficiently schedule them on the available processing units.

Algorithm 1 Second stage: reduction from band bidiagonal to condensed form using the bulge chasing procedure based on element-wise annihilation.

```
1: for j = 1, 2 to N−1 do
2:   {Loop over the sweeps}
3:   last_sweep = j;
4:   for m = 1 to 3 do
5:     for k = 1, 2 to last_sweep do
6:       {I am at column k, generate my task identifier}
7:       me = (j-k) * 3 + m ;
8:       {Set the pointer (matrix row/col position) for
         kernels}
9:       p2 = floor((me+1)/2) * NB + k;
10:      p1 = p2 - NB +1;
11:      if (id == 1) then
12:        {the first red task at column k}
13:        DGBELR(A_{p1:p2,p1−1:p2});
14:      else if (mod(id,2) == 0) then
15:        {a blue task at column k}
16:        DGBRCE(A_{p2+1:p2+NB,p1:p2});
17:      else
18:        {a green task at column k}
19:        DGBLRX(A_{p1:p2,p1:p2});
20:      end if
21:    end for
22:  end for
23: end for
```

We rely on a highly productive runtime environment system, which is described in the next Section.

B. The Runtime Environment System QUARK

By now, multicore processors are ubiquitous in both low-end consumer electronics and high-end servers and supercomputer installations. This leads to the emergence of numerous multithreading frameworks, both academic and commercial, embracing the idea of task scheduling: Cilk [25], OpenMP (tasking features) [26], Intel Threading Building Blocks [27], just to name a few prominent examples. From our perspective, one especially important category of such frameworks are the multithreading systems based on dataflow principles. They represent the computation as a *Direct Acyclic Graph* (DAG) and schedule tasks at runtime through resolution of data hazards: *Read after Write* (RaW), *Write after Read* (WaR) and *Write after Write* (WaW). QUeueing And Runtime for Kernels (QUARK) is an example of such a system. Two other, very similar, academic projects are also available: StarSs [28] from Barcelona Supercomputer Center and StarPU [29] from INRIA Bordeaux. While all three systems have their strength and weaknesses, QUARK [7] has vital extensions for use in a numerical library.

VI. Coalescing Task Granularity Study

In this Section, we propose to study the consequences of coalescing of computational tasks.

A. Enhancing Cache Memory Reuse

As previously explained in Sections IV-C and V-A2, the element-wise annihilation kernels are so fine-grained that their computation performance is solely guided by the memory bus speed. Although these kernels are optimized for cache reuse, a single kernel function call cannot, on its own, take advantage of running in-cache. Once the data is fetched into the high level caches and the registers, it needs to stay there as much as possible and has to be reused between computational tasks. Indeed, the true sharing cache protocol will permit in practice to reduce the latency overhead and by the same token, to remove the pressure from the memory bus bandwidth.

The idea of coalescing computational tasks was born from this conclusion. By aggregating tasks together into groups, we ensure that the data will be reused among the various tasks belonging to a certain group. As a consequence, operations that are supposedly memory-bound increase their ratio of computation to off-chip communication and become compute-bound, which renders them amenable to efficient execution on multicore architectures.

B. Reducing the Scheduler Overhead

Another positive aspect of coalescing of computational tasks during the second stage is the direct attenuation of the scheduler overhead. As a dynamic scheduler, QUARK has shown very good performance on compute-intensive tile algorithms including QR, LU, and Cholesky factorizations [30], [31]. However, it could potentially get in the way when the tasks are mostly memory-bound and depend critically not only on good data locality but also on very short periods of time required for switching between tasks eligible for execution. Under such constraints tracking complex data dependencies may turn out to be prohibitively expensive. The very large amount of fine-grained tasks generated in the bulge chasing procedure will further exacerbate these problems. Naturally then, merging computational tasks is necessitated by the confluence of the above factors. By coalescing multiple annihilation steps, that could conceptually become separate tasks, we address directly both of the issues. When merged, tasks naturally follow good data locality guidelines by retaining data in caches for as long as the computation continues – QUARK does not migrate threads that are in the state of execution. And the look-up and update of the data dependence information takes place only once per each merged task. The savings in overhead will come from both lack of cache memory pollution from QUARK's internal data structures and elimination of tens if not hundreds of instructions that help QUARK make scheduling decisions and keep its data in a consistent state.

The next Section describes the impact of coalescing of tasks on the cache utilization and the execution performance. It also compares the new tile two-stage BRD algorithm with the state-of-the-art numerical libraries.

VII. Performance Results

A. Environment Settings

Our experiments have been performed on the largest shared-memory system we could access at the time of writing of this paper. It is representative of a vast class of servers and workstations commonly used for computationally intensive workloads. It clearly shows the industry's transition from chips with few cores to tens of cores; from compute nodes with order O(10) cores to O(100) designs, and from Front Side Bus memory interconnect (Intel's NetBurst and Core Architectures) to Non-Uniform Memory Access (NUMA) and cache coherent NUMA hardware (AMD's HyperTransport and Intel's QuickPath Interconnect). Our experimental server was composed of eight AMD Opteron(tm) Processor, 8439 SE of six cores (48 cores total), each running at 2.81 GHz with 128 GB of memory. The total number of cores is evenly spread among two physical boards. The cache size per core is 512 KB. All the computations are done in double precision arithmetic. The theoretical peak for this architecture in double precision is 539.5 Gflop/s (11.2 Gflop/s per core).

There are a number of software packages that implement the bidiagonal reduction. For comparison, we used as many as we were aware of, and here we briefly describe each one in turn. LAPACK [13] is a library of FORTRAN 77 subroutines for solving the most commonly occurring problems in dense matrix computations. LAPACK can solve systems of linear equations, linear least squares problems, eigenvalue problems and singular value problems. The equivalent routine name performing the bidiagonal reduction is DGEBRD. LAPACK has been linked with the optimized Intel MKL BLAS V10.3.2 to support parallelism at the BLAS level. ScaLAPACK [32], [33] is a library of high-performance linear algebra routines for distributed-memory message-passing MIMD computers and networks of workstations supporting PVM [34] and/or MPI [35]. It is a continuation of the LAPACK project, targeting distributed environment systems. The machine used in our experiments emulate to some extend a distributed environment and it is natural to include this library in our experiments. The equivalent routine name performing the bidiagonal reduction is PDGEBRD. Last but not least, we also compare our implementation against our previous version of the tile two-stage BRD algorithm [4].

We recall that the algorithmic complexity of the standard full bidiagonal reduction is $\frac{8}{3}N^3$ and this is the formula used to compute the performance in Gflop/s for all the experiments. We refer in the subsequent figures to our new tile two-stage BRD algorithm as PLASMA-DGEBRD and the associated singular value solver as PLASMA-DGESVD.

B. Coalescing Task Size Impact on Cache Behavior

One of the key studies is the effect of the group size on the cache utilization. Figure 4 shows the effect of the coalescing task size on the cache TLB misses. These results were generated using the performance counter library PAPI [21]. A coalescing task size of four seems to be the best value for minimizing the cache TLB misses and thus, one could expect the highest performance. A small group size engenders however a large penalty in terms of TLB misses. We also note that for large matrices, a group size of six, may give us similar performance to a group size equal to four.

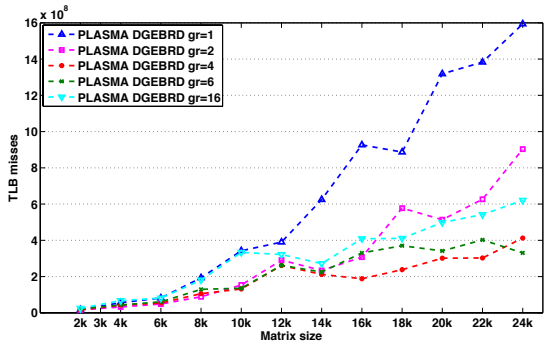


Figure 4. Effect of the group size on the cache TLB misses.

C. Coalescing Task Size Impact on Performance

Figure 5 shows the effect of the coalescing group size on the overall performance of the tile two-stage BRD algorithm. The trade-off is between the degree of parallelism and the amount of data reuse: the higher the degree of parallelism (due to small coalescing group size), the smaller the data reuse and the smaller the degree of parallelism (due to high coalescing group size), the higher the data reuse. In our experiments, a coalescing task size of four seems to get the highest performance when asymptotic matrix sizes are considered.

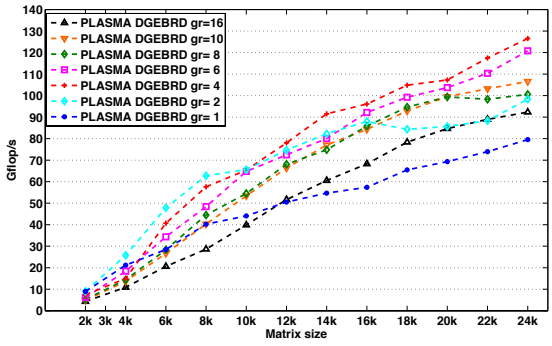


Figure 5. Effect of the group size on the tile two-stage BRD algorithm using eight socket hexa-core AMD Opteron (48 cores total).

D. Performance Comparisons

Figure 6 compares our tile two-stage BRD algorithm against the state-of-the-art numerical libraries described in Section VII-A. Our new implementation outperforms by far the other BRD algorithms. It achieves up to 50-fold speed up against LAPACK and 12-fold speed up against the equivalent routine from the commercial package Intel MKL. It also achieves an impressive 2-fold speed up against our previous implementation of the tile two-stage BRD algorithm [4], which further emphasizes the importance of the task coalescing technique.

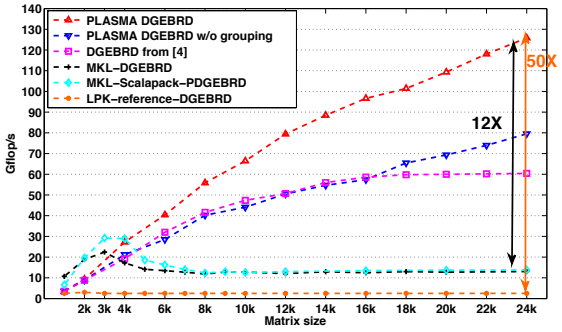


Figure 6. Performance comparisons of the tile two-stage BRD algorithm against the state-of-the-art numerical libraries using eight socket hexa-core AMD Opteron Processors (48 cores total).

E. Performance Scalability

Figure 7 presents the performance scalability of the tile two-stage BRD algorithm achieved on the entire system. Our new algorithm scales as the matrix sizes increase and asymptoticly achieves a perfect speedup, despite the side effects of running on a NUMA system.

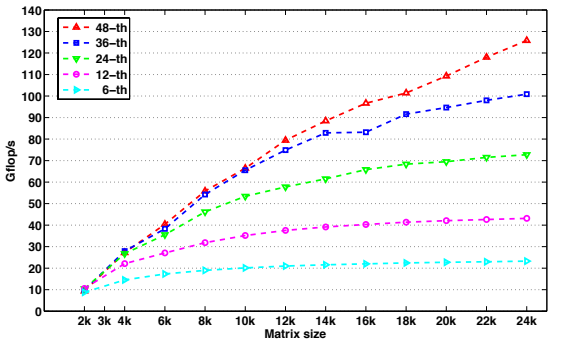


Figure 7. Performance scalability of the tile two-stage BRD algorithm.

F. Performance of the Singular Value Solver

Figure 8 shows the performance of the entire singular value solver. Calculating the singular values from the bidiagonal form is an $O(n^2)$ procedure and therefore, has not been optimized yet. We simply call the routine DBDSQR from LAPACK to perform the singular value computations. As expected, our solver implementation still considerably outperforms the singular value solvers from open-source and vendor software packages.

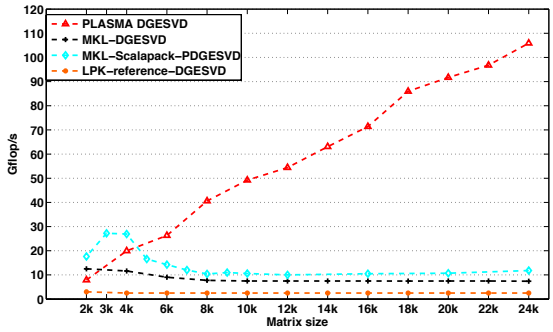


Figure 8. Performance comparisons of the singular value solver) based on our BRD implementation) against the state-of-the-art numerical libraries using eight socket hexa-core AMD Opteron Processors (48 cores total).

VIII. Power Consumption Analysis

A. Hardware/Software Setup

The experiments have been conducted on a single node of a distributed system named systemg.cs.vt.edu from Virginia Tech composed of 324 nodes with InfiniBand interconnect. Each node is a dual-socket quad-core Intel Xeon 2.8GHz (2592 cores total) with 8GB of memory. It is actually the largest power-aware compute system in the world. It has over 30 power and thermal sensors per node and relies on PowerPack [22] to obtain measurements of the major system components' power consumption using power meters attached to the hardware of the system. The PowerPack framework shown in Figure 9 is a collection of software components, including libraries and APIs, which enable system component-level power profiling correlated to application functions. PowerPack obtains measurements from power meters attached to the hardware of the system. As multicore systems evolve, the framework can be used to indicate the application parameters and the system components that affect the power consumption on the multicore unit. PowerPack allows the user to obtain direct measurements of the major system components' power consumption, including the CPU, memory, hard disk, and motherboard. This fine-grain measurement allows power consumption to be measured on a per-component basis.

B. Power Profiling

Here, we present the power consumption profiles for bidiagonal reduction of a matrix of size 5000. For tile algorithm runs, we use the tile size NB of 80 and the internal blocking parameter IB to be 20. Such profiles allow us to not only observe temporal changes of the power drawn by

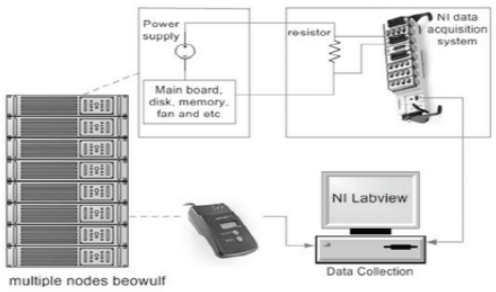


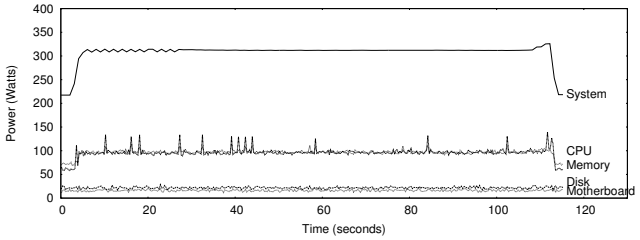
Figure 9. Conceptual flow chart of the PowerPack framework.

the hardware but, primarily, give us additional information about the usage of system components and how changes in this usage are influenced by various algorithms. However, one thing to keep in mind is the fact that the external power measurements, such as the one we were using, provide much less precise results than the built-in capabilities of modern processors in the form of hardware counters and the associated libraries such as PAPI [36].

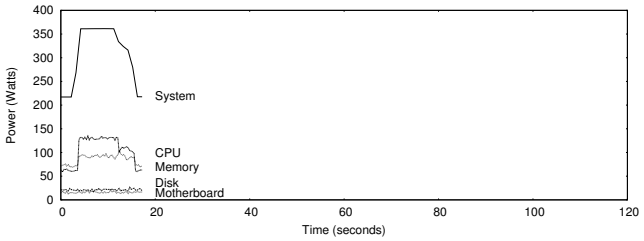
In an earlier work [37], power profiles were used to reveal varying power demand from the major components of a smaller computer system. A somewhat similar analysis may be done in the case of our variable grain bidiagonalization schemes. Figure 10 compares directly the best shared-memory implementation of the bidiagonal reduction from MKL [14] with our most efficient tile code that features the optimal task grouping size. Clearly, the time to solution is much different for both codes but we chose to use the same time scale to underscore this difference and make a statement about the energy savings brought about by the drastic reduction of computation time. Another observation is virtually constant power profile for the MKL implementation – both the CPU and the main memory consume about 100 Watts throughout the running time with a short spike at the very end. This is to be expected from a mostly memory-bound code like this. Throughout the execution, the performance is mostly limited by the memory bandwidth. And at the end, the problem size becomes small enough to fit in cache which increases the computational intensity of the run which corresponds to the spike in the CPU power draw. On the other hand, the tile algorithm reaches almost 150 Watts in CPU power usage which may be attributed to a better cache locality which leads to a higher computational rate.

C. Coalescing Task Size Impact on Power Consumption

Figures 11, 12, 13, 14, 15, 16, 17, and 18 show the effect of the task group size on the power profile of the tile algorithm. A most obvious observation is the fact that both stages are clearly delineated by two different levels of nearly constant power consumption. The reduction to band form draws almost 150 Watts while the reduction from band to bidiagonal form consumes a little bit over



(a) MKL DGEBRD implementation.



(b) PLASMA DGEBRD implementation with the optimal task group size of 6.

Figure 10. Power profiles of first stage of singular value computation: reduction to band bidiagonal form when applied on a matrix of size 5000.

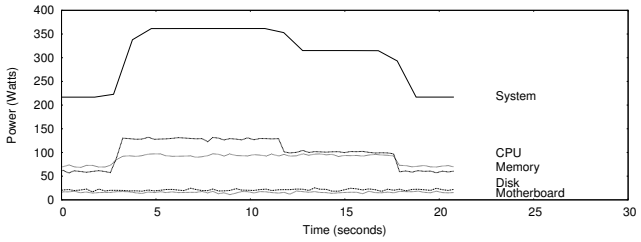


Figure 11. PLASMA DGEBRD implementation with task group size equal 1.

100 Watts. The latter phase is memory-bound which leads to pipeline stalls of each of the cores which result in decreased demand for power. In addition, we may readily observe a transition from one phase to the next by noticing a gradual descent of the power curve at the border of the stages. This transition corresponds to the overlap between the phases afforded by the dynamic runtime scheduler which mixes the tasks from the two phases as long as the data dependencies

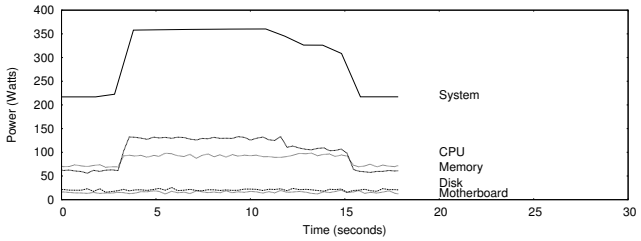


Figure 12. PLASMA DGEBRD implementation with task group size equal 2.

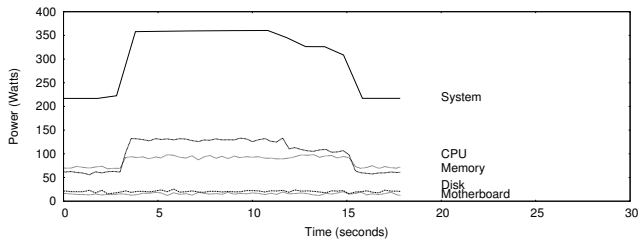


Figure 13. PLASMA DGEBRD implementation with task group size equal 4.

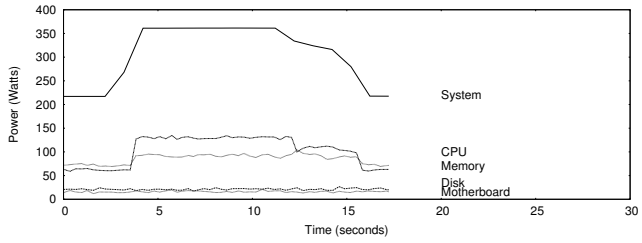


Figure 14. PLASMA DGEBRD implementation with task group size equal 6.

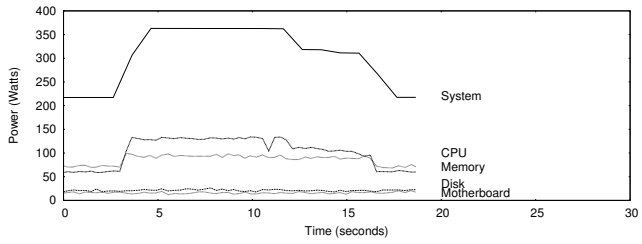


Figure 15. PLASMA DGEBRD implementation with task group size equal 12.

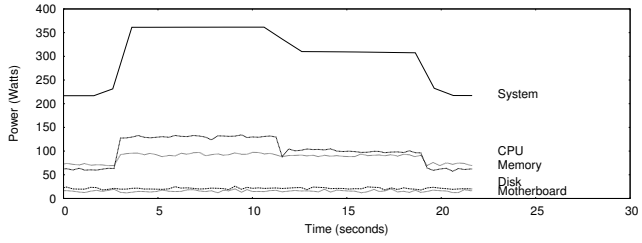


Figure 16. PLASMA DGEBRD implementation with task group size equal 24.

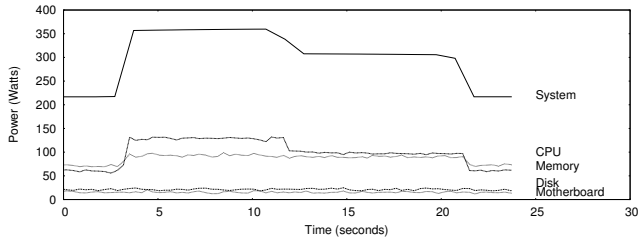


Figure 17. PLASMA DGEBRD implementation with task group size equal 32.

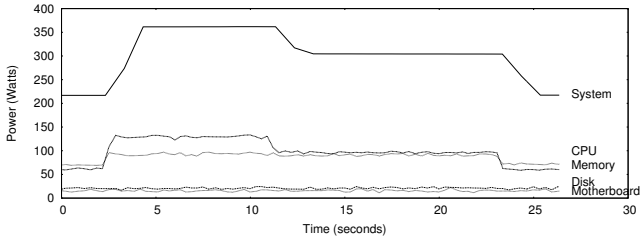


Figure 18. PLASMA DGEBRD implementation with task group size equal 64.

have been satisfied. Finally, there is a further decrease in power consumption at the end of the second phase as the bulge chasing procedure runs out of tasks which limits the available degree of parallelism and thus leaves some cores in the idle state. Finally, only the bulge chasing procedure is affected by the size of the task groups and the power profiles from the figures show clearly how the corresponding portions of the plots are the only ones that change.

IX. Summary and Future Work

We presented a study of task coalescing and its effects on performance and power consumption of a two-stage matrix bidiagonalization procedure. We have shown how the tuning of the size of task grouping can substantially improve the running time on modern multicore hardware – a many-fold speedup may be expected (up to 50X). Although only a small fraction of the system theoretical peak is achieved (25%), the performance of this memory-bound algorithm has been significantly accelerated on homogeneous multicore thanks to in-cache computations.

In our future work we will concentrate on calculation of singular vectors. This will require appropriate accumulation of all the transformations that were applied throughout our current algorithm and it may also provide additional opportunities for even larger overlap between the multiple computational stages. This could further improve the degree of parallelism and remove inherent sequential bottlenecks by hiding them behind the tasks that accumulate the unitary transformations and calculate the singular vectors.

In terms of improving the energy efficiency of our implementation, we are also looking at the Dynamic Voltage and Frequency Scaling (DVFS) feature and how it may be applied, especially in the second stage of the code. In that stage, the computational intensity of most tasks is low enough to warrant reduction in CPU frequency. We would like to further investigate under what circumstances such a scheme could yield reduction in energy consumption without adversely impacting the execution time.

Finally, we consider introducing static scheduling and thus doing away with the runtime scheduling layer. If implemented efficiently, this will have a beneficial effect of reducing overheads of the dynamic scheduler. This will

certainly alter the balance between the need to reduce these overheads and the struggle to keep the level of parallelism high. At the very least, we expect for many of the tuned parameters to no longer be optimal when carried over directly from the dynamically scheduled code to the statically scheduled one.

X. Acknowledgment

The authors would like to thank Kirk Cameron and Hung-Ching Chang from the Department of Computer Science at Virginia Tech, for granting us access to their experimental platforms for the power profiling study.

References

[1] G. H. Golub and C. Reinsch, "Singular value decomposition and least squares solutions," *Numer. Math.*, vol. 14, pp. 403–420, 1970.

[2] G. H. Golub and C. F. Van Loan, *Matrix Computations*, 3rd ed., ser. John Hopkins Studies in the Mathematical Sciences. Baltimore, Maryland: Johns Hopkins University Press, 1996.

[3] L. N. Trefethen and D. Bau, *Numerical Linear Algebra*. Philadelphia, PA: SIAM, 1997. [Online]. Available: http://www.siam.org/books/OT50/Index.htm

[4] H. Ltaief, P. Luszczek, and J. Dongarra, "High Performance Bidiagonal Reduction using Tile Algorithms on Homogeneous Multicore Architectures," *ACM TOMS*, 2011, Accepted.

[5] G. Ballard, J. Demmel, and I. Dumitriu, "Communication-optimal parallel and sequential eigenvalue and singular value algorithms," EECS University of California, Berkeley, CA, USA, Technical Report EECS-2011-14, February 2011, LAPACK Working Note 237.

[6] G. W. Stewart, "The decompositional approach to matrix computation," *Computing in Science & Engineering*, vol. 2, no. 1, pp. 50–59, Jan/Feb 2000, iSSN: 1521-9615; DOI 10.1109/5992.814658.

[7] A. YarKhan, J. Kurzak, and J. Dongarra, "QUARK Users' Guide: QUeueing And Runtime for Kernels," *University of Tennessee Innovative Computing Laboratory Technical Report ICL-UT-11-02*, 2011.

[8] *PLASMA Users' Guide, Parallel Linear Algebra Software for Multicore Architectures, Version 2.3*, University of Tennessee Knoxville, November 2010.

[9] A. Haidar, H. Ltaief, and J. Dongarra, "Parallel reduction to condensed forms for symmetric eigenvalue problems using aggregated fine-grained and memory-aware kernels," in *SC11: International Conference for High Performance Computing, Networking, Storage and Analysis*, Seattle, WA, USA, November 12-18 2011.

[10] P. Luszczek, H. Ltaief, and J. Dongarra, "Two-stage tridiagonal reduction for dense symmetric matrices using tile algorithms on multicore architectures," in *IPDPS 2011: IEEE International Parallel and Distributed Processing Symposium*, Anchorage, Alaska, USA, May 16-20 2011.

[11] E. Agullo, B. Hadri, H. Ltaief, and J. Dongarra, "Comparative study of one-sided factorizations with multiple software packages on multi-core hardware," *SC '09: Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, pp. 1–12, 2009.

[12] A. S. Householder, "Unitary triangularization of a nonsymmetric matrix," *Journal of the ACM (JACM)*, vol. 5, no. 4, October 1958, DOI 10.1145/320941.320947.

[13] E. Anderson, Z. Bai, C. Bischof, S. L. Blackford, J. W. Demmel, J. J. Dongarra, J. D. Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. C. Sorensen, *LAPACK User's Guide*, 3rd ed. Philadelphia: Society for Industrial and Applied Mathematics, 1999.

[14] Intel, "Math Kernel Library," Available at http://software.intel.com/en-us/articles/intel-mkl/.

[15] R. G. Grimes and H. D. Simon, "Solution of large, dense symmetric generalized eigenvalue problems using secondary storage," *ACM Transactions on Mathematical Software*, vol. 14, pp. 241–256, September 1988. [Online]. Available: http://doi.acm.org/10.1145/44128.44130

[16] B. Lang, "Efficient eigenvalue and singular value computations on shared memory machines," *Parallel Computing*, vol. 25, no. 7, pp. 845–860, 1999.

[17] C. H. Bischof, B. Lang, and X. Sun, "Algorithm 807: The SBR Toolbox—software for successive band reduction," *ACM Transactions on Mathematical Software*, vol. 26, no. 4, pp. 602–616, 2000.

[18] L. G. Valiant, "A bridging model for parallel computation," *Communications of the ACM*, vol. 33, no. 8, Aug. 1990, dOI 10.1145/79173.79181.

[19] L. Karlsson and B. Kågström, "Parallel two-stage reduction to Hessenberg form using dynamic scheduling on shared-memory architectures," *Parallel Computing*, 2011, dOI:10.1016/j.parco.2011.05.001.

[20] B. Kågström, D. Kressner, E. Quintana-Orti, and G. Quintana-Orti, "Blocked Algorithms for the Reduction to Hessenberg-Triangular Form Revisited," *BIT Numerical Mathematics*, vol. 48, pp. 563–584, 2008.

[21] "Performance Application Programming Interface (PAPI). Innovative Computing Laboratory, University of Tennessee. Available at http://icl.cs.utk.edu/papi/."

[22] R. Ge, X. Feng, S. Song, H.-C. Chang, D. Li, and K. W. Cameron, "Powerpack: Energy profiling and analysis of high-performance systems and applications," *IEEE Transactions on Parallel and Distributed Systems*, vol. PDS-21, no. 5, pp. 658–671, May 2010.

[23] H. Ltaief, J. Kurzak, and J. Dongarra, "Parallel band two-sided matrix bidiagonalization for multicore architectures," *IEEE Transactions on Parallel and Distributed Systems*, vol. 21, no. 4, April 2010.

[24] R. Schreiber and C. Van Loan, "A storage efficient WY representation for products of householder transformations," *SIAM J. Sci. Statist. Comput.*, vol. 10, pp. 53–57, 1989.

[25] R. D. Blumofe, C. F. Joerg, B. C. Kuszmaul, C. E. Leiserson, K. H. Randall, and Y. Zhou, "Cilk: An efficient multithreaded runtime system," in *Principles and Practice of Parallel Programming, Proceedings of the fifth ACM SIGPLAN symposium on Principles and Practice of Parallel Programming, PPOPP'95*. Santa Barbara, CA: ACM, July 19-21 1995, pp. 207–216.

[26] *OpenMP Application Program Interface, Version 3.0*, OpenMP Architecture Review Board, 2008.

[27] "Intel Threading Building Blocks," http://www.threadingbuildingblocks.org/.

[28] *SMP Superscalar (SMPSs) User's Manual, Version 2.0*, Barcelona Supercomputing Center, 2008.

[29] C. Augonnet, S. Thibault, R. Namyst, and P. Wacrenier, "StarPU: A unified platform for task scheduling on heterogeneous multicore architectures," *Concurrency Computat. Pract. Exper.*, 2010, (to appear).

[30] J. Kurzak, H. Ltaief, J. J. Dongarra, and R. M. Badia, "Scheduling dense linear algebra operations on multicore processors," *Concurrency Computat.: Pract. Exper.*, vol. 21, no. 1, pp. 15–44, 2009.

[31] 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," *J. Phys.: Conf. Ser.*, vol. 180, no. 1, 2009.

[32] L. S. Blackford, J. Choi, A. Cleary, E. F. D'Azevedo, J. W. Demmel, I. S. Dhillon, J. J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. W. Walker, and R. C. Whaley, *ScaLAPACK Users' Guide*. Philadelphia: Society for Industrial and Applied Mathematics, 1997.

[33] J. Choi, J. J. Dongarra, S. Ostrouchov, A. Petitet, D. W. Walker, and R. C. Whaley, "The design and implementation of the ScaLAPACK LU, QR, and Cholesky factorization routines," *Scientific Programming*, vol. 5, pp. 173–184, 1996.

[34] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, and V. Sunderam, *PVM: Parallel Virtual Machine. A Users' Guide and Tutorial for Networked Parallel Computing*. Cambridge, MA: MIT Press, 1994.

[35] M. P. I. Forum, "MPI-2: Extensions to the Message-Passing Interface," 18 Jul. 1997, available at http://www.mpi-forum.org/docs/mpi-20.ps.

[36] S. Browne, J. J. Dongarra, N. Garner, G. Ho, and P. Mucci, "A portable programming interface for performance evaluation on modern processors," *The International Journal of High Performance Computing Applications*, vol. 14, no. 3, pp. 189–204, 2000.

[37] H. Ltaief, P. Luszczek, and J. Dongarra, "Profiling High Performance Dense Linear Algebra Algorithms on Multicore Architectures for Power and Energy Efficiency," in *EnA-HPC 2011: Second International Conference on Energy-Aware High Performance Computing*, Hamburg, Germany, Sept 7-9 2011.