Noname manuscript No.
(will be inserted by the editor)

Profiling High Performance Dense Linear Algebra Algorithms on Multicore Architectures for Power and Energy Efficiency

Hatem Ltaief
Piotr Luszczek
Jack Dongarra

Received: date / Accepted: date

Abstract This paper presents the power profile of two high performance dense linear algebra libraries i.e., LAPACK and PLASMA. The former is based on block algorithms that use the fork-join paradigm to achieve parallel performance. The latter uses fine-grained task parallelism that recasts the computation to operate on submatrices called tiles. In this way tile algorithms are formed. We show results from the power profiling of the most common routines, which permits us to clearly identify the different phases of the computations. This allows us to isolate the bottlenecks in terms of energy efficiency. Our results show that PLASMA surpasses LAPACK not only in terms of performance but also in terms of energy efficiency.

Keywords Power Profile · Energy Efficiency · Dense Linear Algebra · Tile Algorithms · Multicore Architectures

1 Introduction

After the processor industry underwent the transition from sequential to multicore processors [19] the dense linear algebra software transitioned from block algorithms of LAPACK [2] to the tile algorithms of PLASMA [21]. The former stresses an efficient use of deep memory hierarchies while the latter focuses on introducing a sufficient amount of parallelism to keep all the newly introduced cores busy. The shift to multicore designs at the hardware level was primarily driven by the prohibitive power consumption and the corresponding power dissipation needs of the ever increasing clock frequencies of superscalar processors. On the other hand, the tile algorithms were introduced to keep the new processor cores busy at all times in pursuit of sustained performance levels. The obvious question of energy efficiency at the software level was never adequately answered until now.

Our focus is on two main classes of numerical linear algebra algorithms: one-sided and two-sided factorizations. The former class includes Cholesky, QR, and LU factorizations which are the most computationally intensive step in solving various linear systems of equations. The latter class includes tridiagonal (TRD) and bidiagonal (BRD) reductions which consume the most amount of computation time in solving the symmetric eigenvalue problem and in obtaining the singular value decomposition of a matrix, respectively [12,20]. With the experiments reported here we are able to conclude that PLASMA is not only a high performance library for dense linear algebra but it also is more energy efficient than LAPACK.

The remainder of this document is organized as follows. Section 2 gives an overview of related work in this area. Section 3 recalls the block algorithms as they are implemented in LAPACK [2]. Section 4 explains the tile algorithms and their implementation in PLASMA [21]. Section 5 describes the power profiling tool called PowerPack [11] and the experimental plat-

This work was supported in part by the National Science Foundation under grant No. 0910899.

H. Ltaief
KAUST Supercomputing Laboratory
Thuwal, Saudi Arabia
Tel.: +966-2808-8001
Fax: +966-2802-1194
E-mail: Hatem.Ltaief@kaust.edu.sa

P. Luszczek and J. Dongarra
Department of Electrical Engineering and Computer Science
University of Tennessee, Knoxville
Tel.: +1-865-974-8295
Fax: +1-865-974-8296
E-mail: [luszczek,dongarra]@eecs.utk.edu

form. Section 6 outlines the power consumption of the different block and tile algorithms. Finally, Section 7 summarizes the results of this paper and presents the ongoing work.

2 Related Work

Dynamic Voltage and Frequency Scaling (DVFS) is a commonly used technique with which it is possible to achieve reduction of energy consumption [10,14]. One example is the use of DVFS to reduce the energy usage of MPI applications by determining different phases for the application [10]. The authors utilized a brute-force approach to determine the optimal energy and performance settings for each phase and then executed the application according to the new settings. In particular, the NAS BT benchmark is divided into two phases that are executed at multiple gear points (voltage and frequency level). The BT is executed at gears 1 and 2 giving an energy saving of 10% with a time penalty of 5%.

A system called Jitter [14] was introduced with the goal of exploiting the MPI wait time for load-imbalanced applications. The frequency and voltage of nodes with less computational time were reduced to save overall energy while other compute-intensive nodes completed.

The scaling of the voltages and frequencies was used to reduce overall energy consumption by targeting the processors that are not on the critical path of the calculation [9,8]. This approach is oriented towards saving power without incurring performance penalties. The experiments yielded very promising results. Namely, for matrices extracted from real applications, the observed savings in power were close to the optimal values. This lended credibility to the authors' recursive strategies and their usefulness in saving power.

A different example of power-aware computing is exploitation of mixed-precision iterative refinement [6]. By taking advantage of numerical properties of the input data it is possible to lower the working precision of the calculation and thus reap benefits of shorter execution time and lower power requirements, which together result in lower energy consumption [3].

The selection of the most efficient algorithm and its implementation is determined by first specifying a criterion – preferably a quantitative one. The Gflop/s per Watt is one such metric, but alternatives are also being considered [4]. We partially side step this issue here by studying temporal energy and power characteristics of various linear algebra algorithms rather then reducing them to a single power-aware metric.

3 Block Algorithms

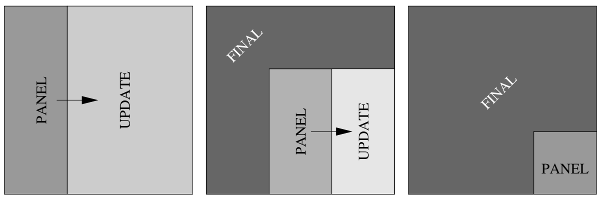
This section recalls the notion of block algorithms in LAPACK [2] and describes, in particular, the one-sided factorizations (QR, LU, and Cholesky) and the two-sided transformations (TRD and BRD).

3.1 Description of the Block Algorithms

LAPACK implements block algorithms to solve linear systems of equations as well as eigenvalue problems and singular value decompositions. Block algorithms are characterized by two successive phases: panel factorization and update of the trailing submatrix. During the panel factorization, the transformations are only applied within the panel. The panel factorization is very rich in Level 2 BLAS operations because the transformations are singly applied. Once accumulated within the panel, those transformations are applied to the rest of the matrix (the trailing submatrix) in a blocking manner leading to Level 3 BLAS operations. While the update of the trailing submatrix is compute-bound and very efficient, the panel factorization is memory-bound and may appear to be a bottleneck for some numerical linear algebra algorithms, especially for the two-sided factorizations. Last but not least, the parallelism within LAPACK occurs only at the level of the BLAS routines, which follows the expensive fork-join model. Basically, all processing units need to synchronize before and after each call to BLAS kernels.

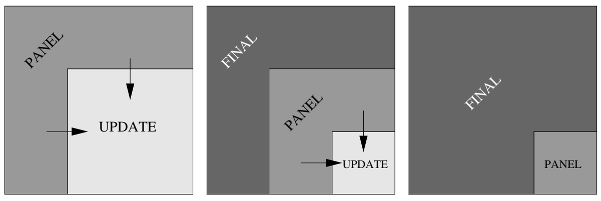
3.2 LAPACK One-Sided Factorizations

The one-sided factorizations i.e., QR, LU, and Cholesky are the first step toward solving linear systems of equations. While the Cholesky factorization is used for symmetric matrices, QR and LU solve linear systems of equations, where the matrices are non-symmetric and general. A three *panel-update* sequence is illustrated in Figure 1. The transformations from the panel are applied to the left of the matrix and the factorization proceeds until the two final factors are computed. Since the matrix has to be (among other things) symmetric for the Cholesky factorization, only the upper or the lower part of the matrix needs to be referenced. It is noteworthy that the computation of the panel is self-contained in the sense that it does not involve data located outside of the specified panel region.



(a) First step. (b) Second step. (c) Third step.

Fig. 1 Panel-update sequences for the LAPACK one-sided factorizations.



(a) First step. (b) Second step. (c) Third step.

Fig. 2 Panel-update sequences for the LAPACK two-sided transformations.

3.3 LAPACK Two-Sided Transformations

The BRD and TRD are the two-sided algorithms studied in this paper. As opposed to one-sided factorizations (i.e., QR, LU, and Cholesky), the computed transformations are applied from the left as well as from the right side of the matrix. Figure 2 shows a three *Panel-Update* sequence until the matrix is reduced to the appropriate form. In the TRD case, since the matrix is symmetric, only the lower/upper part of the matrix is referenced. Additionally, the accumulation of the left and right transformations during the panel computation requires loading into memory the whole unreduced part of the matrix (i.e., the trailing submatrix) at each single reduction step, as opposed to the one-sided factorizations, where only the data located in the current panel is accessed. In fact, the panel factorization is clearly the bottleneck phase for the two-sided algorithms.

Moreover, this successive sequence of *panel-update* in LAPACK has shown strong limitations on multicore architectures. Indeed, the LAPACK framework is not capable of performing any lookahead computations, where the panel or update tasks from multiple steps could significantly overlap. Although, in practice, lookahead techniques would be algorithmically possible only for one-sided factorizations. For two-sided transformations, the one-stage approach (reduction to the final corresponding condensed form without intermediary step) necessitates the panel computational step to be atomic because, as mentioned above, it requires access to the entire trailing submatrix using expensive Level 2 BLAS operations (memory-bound).

The next section describes the concept of tile algorithms and explains how these new algorithms are able to supersede the standard one-sided and two-sided block algorithms.

4 Tile Algorithms

This section recalls the general principles of tile algorithms and describes how the block algorithms had to be redesigned to effectively exploit parallelism from multicore architectures.

4.1 Tile Algorithms

The general idea of tile algorithms is to transform the original matrix to **tile data layout** (TDL) where each data tile is contiguous in memory as in Figure 3. This may demand a complete reshaping of the standard numerical algorithm. The panel factorization as well as the update of the trailing submatrix are then decomposed into several fine-grained tasks, which better fit the memory of the small core caches. The parallelism

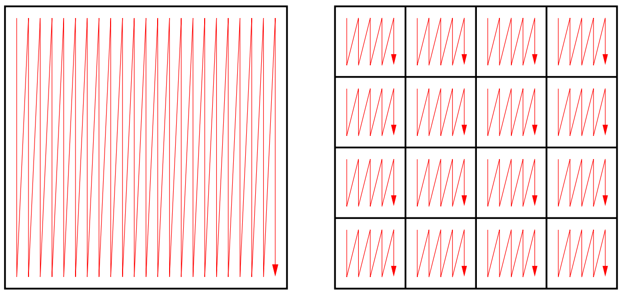


Fig. 3 Translation from LAPACK Layout (column-major) to Tile Data Layout

is then no longer hidden inside the BLAS routines but rather is brought to the fore. The whole computation can then be represented with a directed acyclic graph (DAG), where nodes are computational tasks and edges represent the data dependencies among them. Next, it becomes critical to efficiently schedule the sequential fine-grained tasks across the processing units. A dynamic runtime environment system is used to distribute the tasks as soon as the data dependencies are satisfied.

4.2 Tile One-Sided Transformations

The block formulation of one-sided factorizations had to be redesigned accordingly in order to handle the computations by tiles. This may engender extra flops, which are compensated by the increase of the degree of parallelism [7]. The tile computation of the panel enhances the data locality and therefore, cuts off the number of TLB misses seen for the one-sided block algorithms, due to the large stride access. Tile algorithms [21] have already shown promising results for the one-sided factorizations, as compared to LAPACK and vendor libraries on various multicore architectures and we refer to [1] for a comprehensive performance comparison.

4.3 Tile Two-Sided Transformations

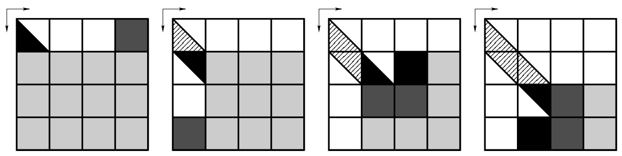
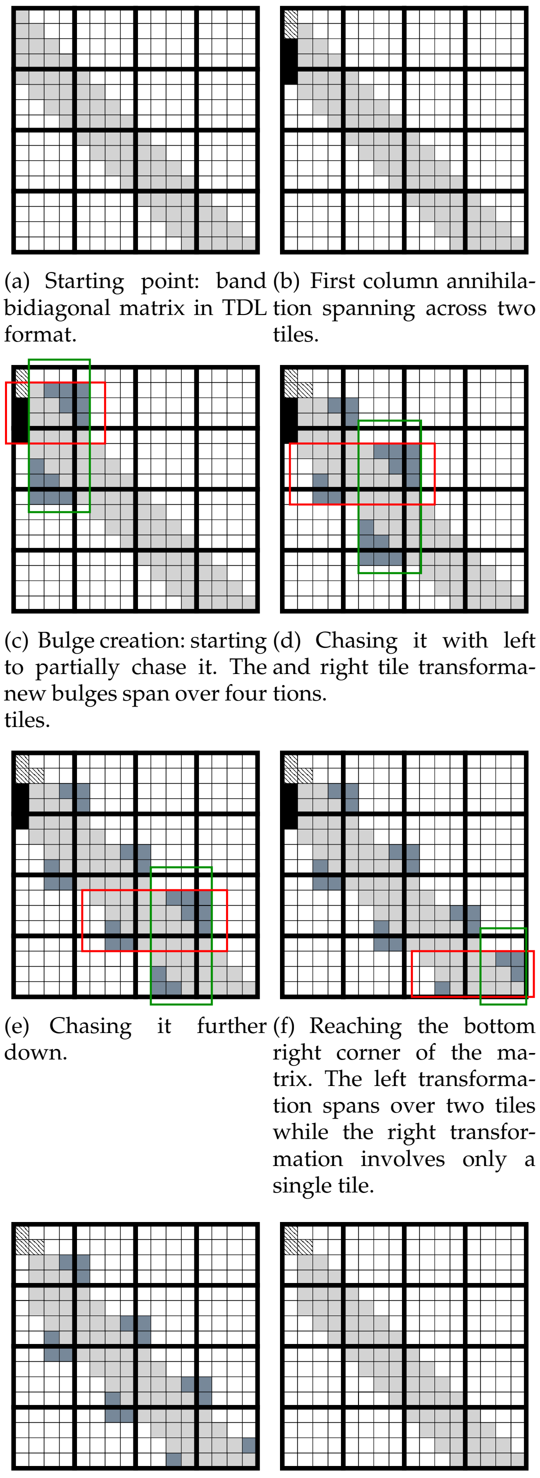

(a) Right Reduction Step 1. (b) Left Reduction Step 1. (c) Right Reduction Step 2. (d) Left Reduction Step 2.

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

A complete new methodology has to be developed to drastically decrease the negative impact of the panel computation of both, TRD and BRD. The standard one-stage approach for the two-sided block reductions has been replaced by a two-stage approach. Two-stage approaches have recently proven to be an interesting solution in achieving high performance in the context of two-sided reductions [5,13,16,17]. The Figures shown in this section are related to the BRD case, whose Figures are similar to the TRD case to some extent. Therefore, for simplicity, we purposefully omit the extra TRD figures. The first stage comprises a reduction of the original matrix to band form. Figure 4 recalls how the band bidiagonal structure is obtained from a 4-by-4 tile matrix. The matrix is reduced to band form by interleaving QR (left transformations) and LQ (right transformations) factorizations. The light gray tiles correspond to transient data, which still need to be reduced. The black and dark gray tiles are being reduced and the dashed tiles are final data tiles. Four interleaved QR/LQ factorization steps are needed to achieve the band bidiagonal form. The overhead of the Level 2 BLAS operations dramatically decreases and most of the computation is performed in Level 3 BLAS, which makes this stage run closer to the theoretical peak of the machine.


(a) Starting point: band bidiagonal matrix in TDL format. (b) First column annihilation spanning across two tiles.

(c) Bulge creation: starting to partially chase it. The new bulges span over four tiles. (d) Chasing it with left and right tile transformations.

(e) Chasing it further down. (f) Reaching the bottom right corner of the matrix. The left transformation spans over two tiles while the right transformation involves only a single tile.

(g) Final Tile Matrix after the first column annihilation: the tile band structure is restored. (h) Final Tile Matrix after the first column annihilation: the tile band structure is restored.

Fig. 5 Execution breakdown of the bulge chasing procedure on a band bidiagonal matrix of size N=16 and NB=4 with **tile data layout** (TDL) after the first column annihilation (black elements). The bright and pale rectangles show the left and right transformations, respectively. The dark grey elements represent the fill-in elements, which eventually need to be chased down to the bottom right corner of the matrix. The dashed elements are the final elements of the bidiagonal structure of the matrix.

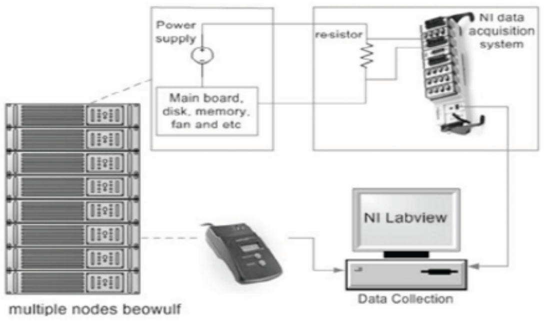


Fig. 6 Description of the PowerPack Framework.

The second stage further reduces the band matrix to the corresponding compact form. A challenging bulge chasing procedure using orthogonal transformations annihilates the off-diagonal elements column-wise and hunts down the fill-in elements to the bottom right corner of the matrix. Figure 5 shows the dimension of its complexity. The bulge chasing procedure on the tile matrix creates bulges, which could span over multiple tiles, and therefore, they are not contiguous in memory anymore. Special computational kernels obviously need to be implemented to handle the various cases, depending on the number of tiles involved in one particular task. Besides the development of new kernels, a layer of abstraction is required to map the bulge chasing algorithm running on top of column-major data layout (CDL) format into tile data layout (TDL) format. This layer is a crucial component of the two-stage tile BRD and TRD algorithms as it homogenizes the layout format across both stages. We refer to [16,17] for more details on those reductions using a two-stage approach.

The next Section describes the experimental platforms used to perform the power profiling of block and tile algorithms.

5 Experimental Settings

LAPACK and PLASMA are numerical dense linear algebra libraries for shared-memory systems. All of our experiments utilize a single node of an eight-node distributed multicore system named *Dori*, which is available in the Department of Computer Science at Virginia Tech. Each node of the system consists of two dual-core AMD Opteron processors 265 (1.8 GHz) and six 1 GB memory modules per node, each being DDR2 SDRAM clocked at 600 MHz. We used PowerPack [11], which provides power profiling information for advanced execution systems, to measure the power consumption of our applications running on the cluster. The PowerPack framework shown in Figure 6 is a collection of software components, including libraries and APIs, which enables system component-level power profiling correlated to application functions. PowerPack obtains measurements from power meters attached to the hardware of a 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. The next Section highlights the power profiling of block and tile algorithms.

6 Power Profiling

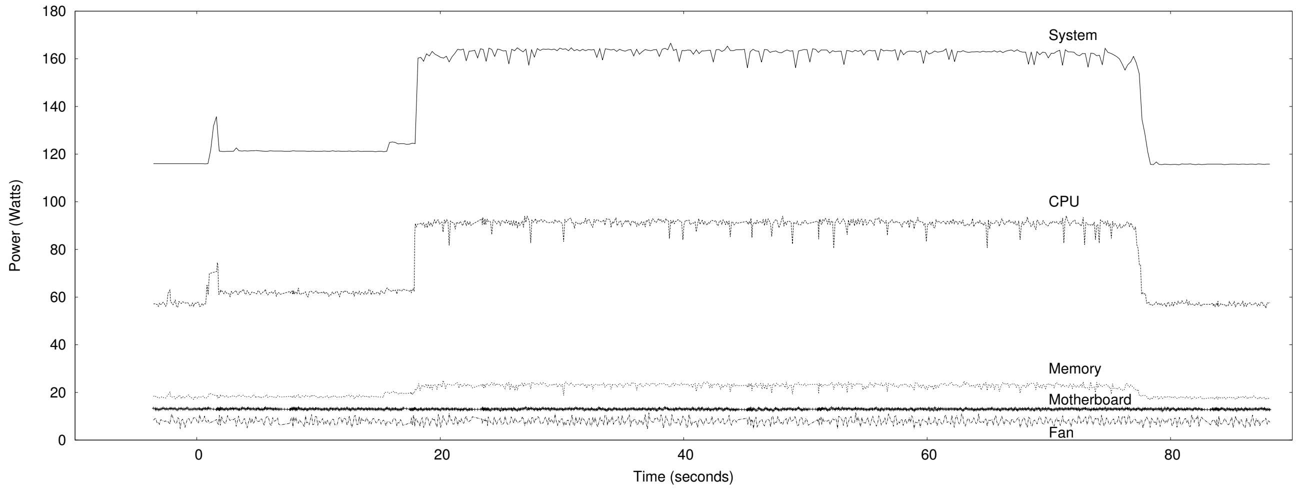
We refrain from publishing a comprehensive set of performance charts for the aforementioned factorizations as we have done elsewhere [1]. Instead, we focus on a detailed study of temporal energy and power characteristics. To the best of our knowledge, it is the first such comprehensive study in the context of dense linear algebra libraries.

6.1 One-Sided Factorizations

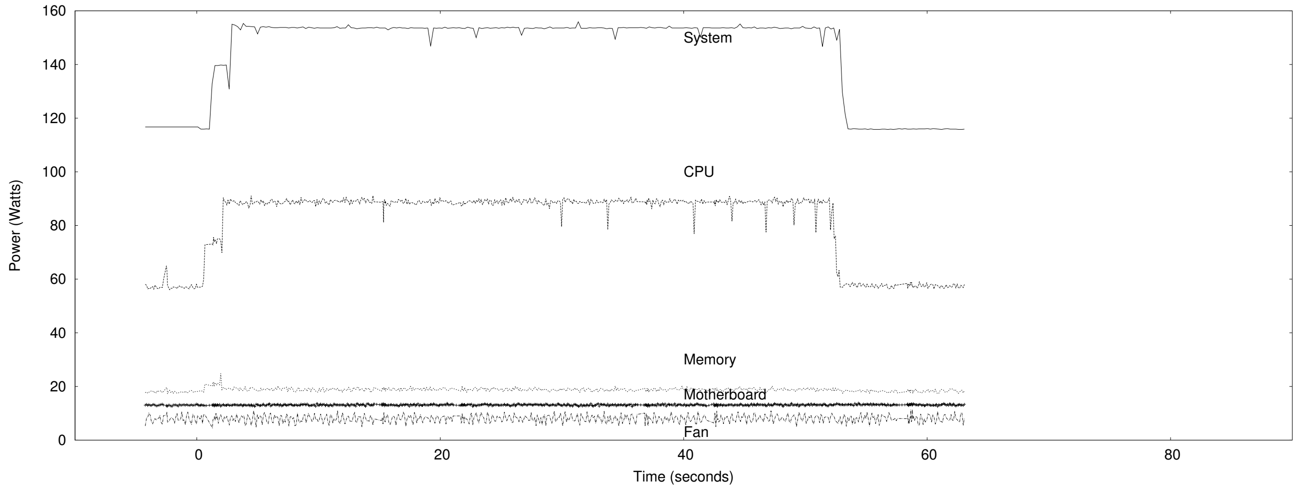
The next six figures present the power consumption during different factorizations of a matrix size $N = 10000$ linked with GotoBLAS2 on the test machine *Dori* (dual-socket dual-core AMD Opteron), as previously described in Section 5.

Figures 7, 8, and 9 show power consumption during the Cholesky, QR, and LU factorizations, respectively. The beginning and the end of computations are readily visible in the figures even though the time period for performing each of the factorizations is not marked explicitly on either of the figures. This is because the CPU power consumption and, consequently, the overall system consumption increases by nearly 100% during the factorization period. As mentioned earlier, we would like to deemphasize the raw execution time (which overall favors the tile approach) and focus on the comparison between the power profiles of LAPACK and PLASMA.

One aspect of the power profiles presented in the figures is the maximum power draw. LAPACK draws visibly more instantaneous power over the course of a run. In particular, it is 3.3% for Cholesky, 5.4% for LU,



(a) LAPACK Cholesky.



(b) PLASMA Cholesky.

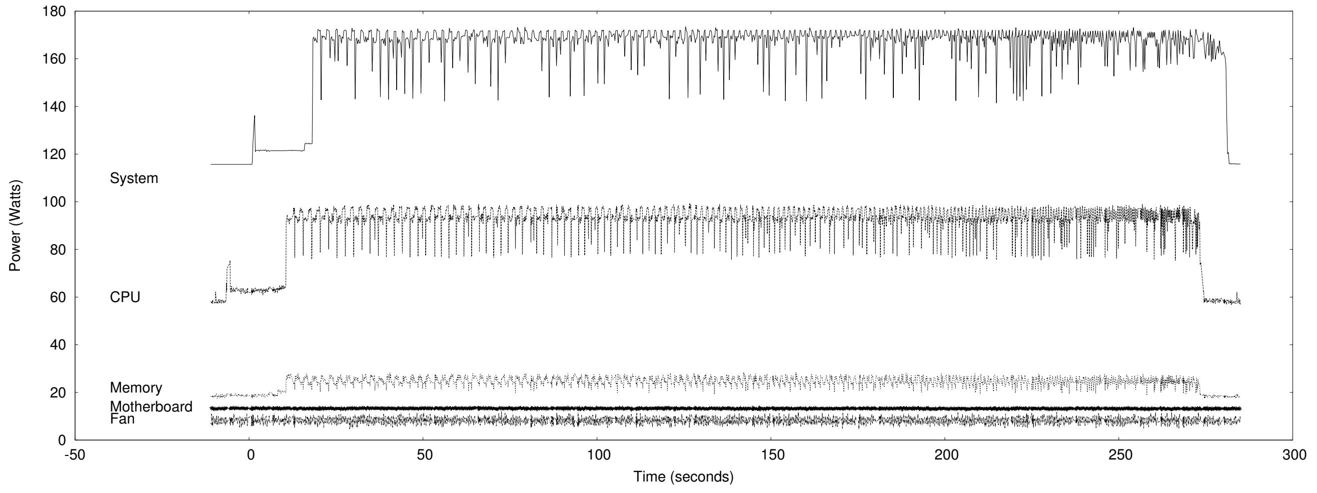
Fig. 7 Power consumption of LAPACK and PLASMA Cholesky on the test system.

and 10.4% for QR factorization, respectively. A similar comparison for a mean power draw of the main memory shows a more staggering difference: 21.1% for Cholesky, 17.3% for LU, and 34.3% for QR. This latter comparison is more pertinent as the memory is bound to overtake CPU as the main consumer of energy on a motherboard [15].

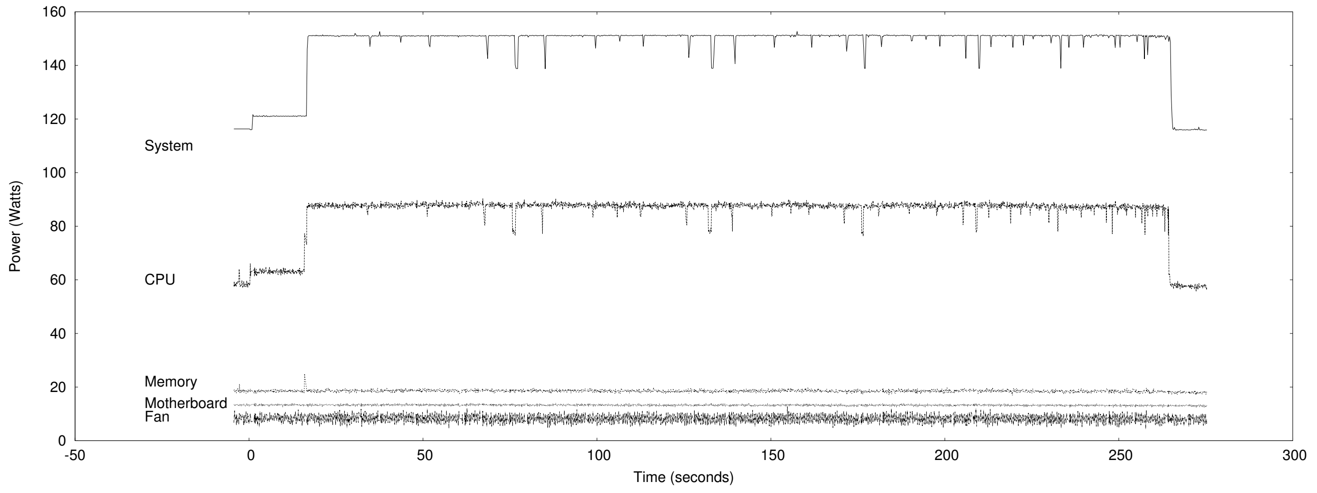
Another feature that might be observed in the figures is the high variability of power draw for LAPACK's QR and LU algorithms, which is not present for the tile algorithms. By counting the number of variations and including the information on the size of the matrix, we can determine that the low points correspond to the power consumption of the panel computation which is memory-bound. For such a computation, the main memory is stressed much more than the processor. The high points of the power draw correspond to the updates of the trailing submatrix: a compute-bound code. The processor is busy inside a matrix-matrix multiply routine and the data is only occasionally fetched from the main memory thanks to the high data reuse. The Cholesky factorization does not show this behavior as it is mostly based on Level 3 BLAS: the panel computation is negligible. On the contrary, the PLASMA power profiling curves for one-sided factorizations are very smooth, which is due to the data locality improvement from the use of tile algorithms.

6.2 Two-Sided Factorizations

Figures 10 and 11 show power consumption during the tridiagonal and bidiagonal reductions, respectively. The observations made for the one-sided factorizations still hold for the two-sided ones. We can, however, note



(a) LAPACK QR.



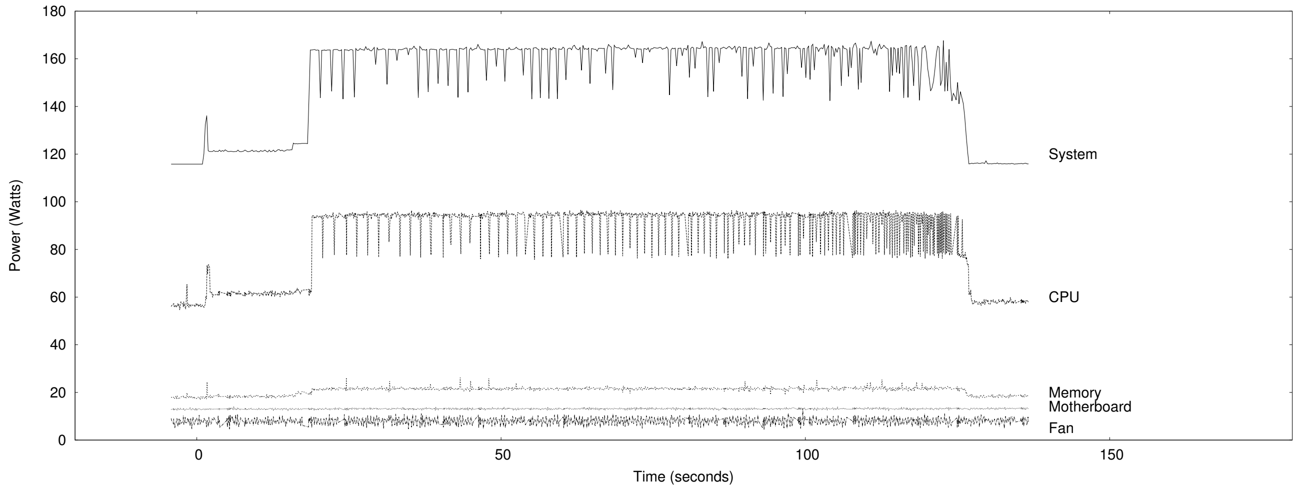
(b) PLASMA QR.

Fig. 8 Power consumption of LAPACK and PLASMA QR on the test system.

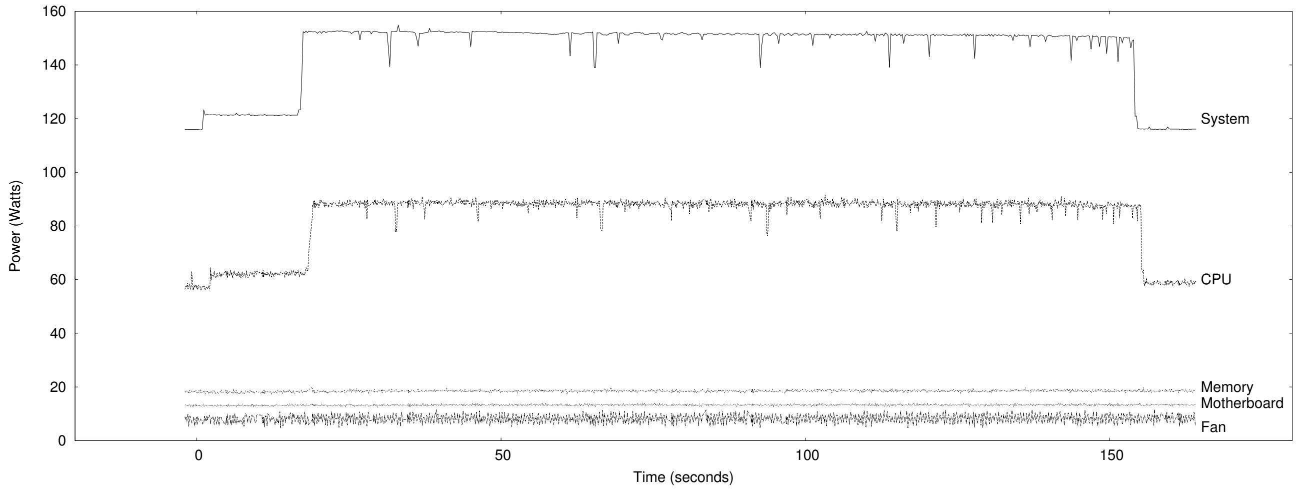
additional phenomena that are specific to the two-sided codes. The two-sided reductions in PLASMA are by design conducted in two stages and it clearly shows in the power profile in the figures. There is a 20% to 30% drop in power draw by the CPU when going into the second stage: a testament to the computationally less-intensive nature of that stage. Correspondingly, the main memory draws more power in the second stage as this stage is more demanding in terms of bandwidth. Therefore, tile algorithms for two-sided reductions allow us to distinctly split level 3 BLAS operations (high power consumption due to intense CPU usage) from level 2 BLAS ones (low power consumption due to intense memory usage).

6.3 Analysis of Results

The results presented in the previous section allow us to draw a number of interesting conclusions and make predictions for the linear algebra research in the context of energy-conscious computing. A result that surprised us the most is the fact that optimizing for performance does not necessarily conflict with low energy consumption. In fact, the PLASMA library is able to outperform LAPACK in terms time-to-solution while at the same time draw less power. As a result, energy consumption win of PLASMA comes from two sources: reduced time and demand for power. The explanation of this phenomenon seems to be related to the use of on-chip resources. For simplicity sake, we can assume that each core has the computational and memory components. The former is responsible for performing the computations and could be claimed to work at the same



(a) LAPACK LU.



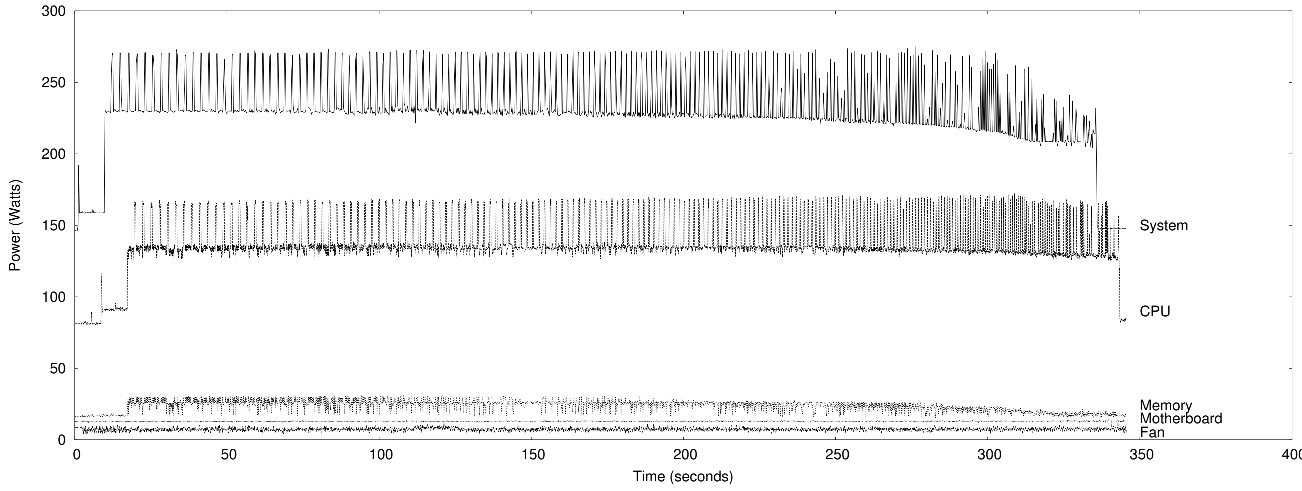
(b) PLASMA LU.

Fig. 9 Power consumption of LAPACK and PLASMA LU on the test system.

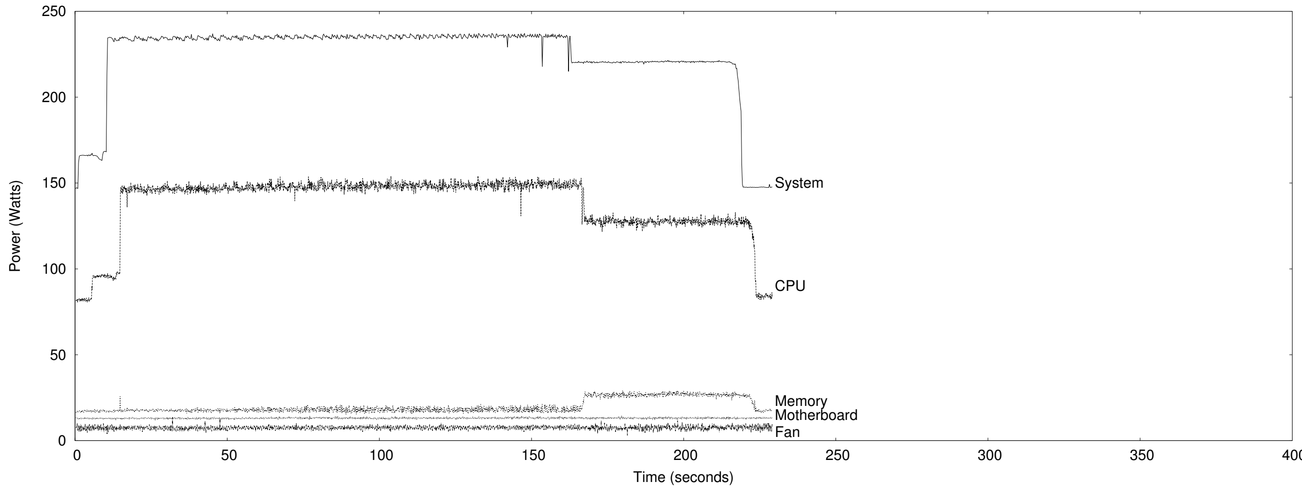
intensity with both PLASMA and LAPACK. On the other hand, the memory components seem to be working in much higher rate for LAPACK as the memory traffic is much more intensive due to a less efficient block storage of data. We believe that this extra work of handling data movement between cache hierarchy and the main memory contributes to the higher power draw of the processors when running LAPACK.

7 Summary and Future Work

This paper presented a detailed power profiling of block algorithms from LAPACK [2] as well as tile algorithms from PLASMA [21]. This energy efficiency analysis allows us to clearly identify the bottlenecks of the different computational steps involved with the block algorithms as well as the major improvements brought by the tile algorithms (power efficiency and high performance computing). Although performed on only four cores, those preliminary results are very encouraging, and we expect an even better power consumption for systems with a larger number of cores, since this is where PLASMA will benefit the most. In the future, we plan to use the results to further optimize the power profile of PLASMA by focusing on the specific energy requirements of the various factorization algorithms and their stages. The power profile optimizations we have in mind would cap the temporal power draw as well as reduce overall energy consumption. We would also like to study how these changes, dictated by power and energy constraints, influence the performance that was achieved and the total running time of the numerical algorithms from both libraries.



(a) LAPACK TRD.



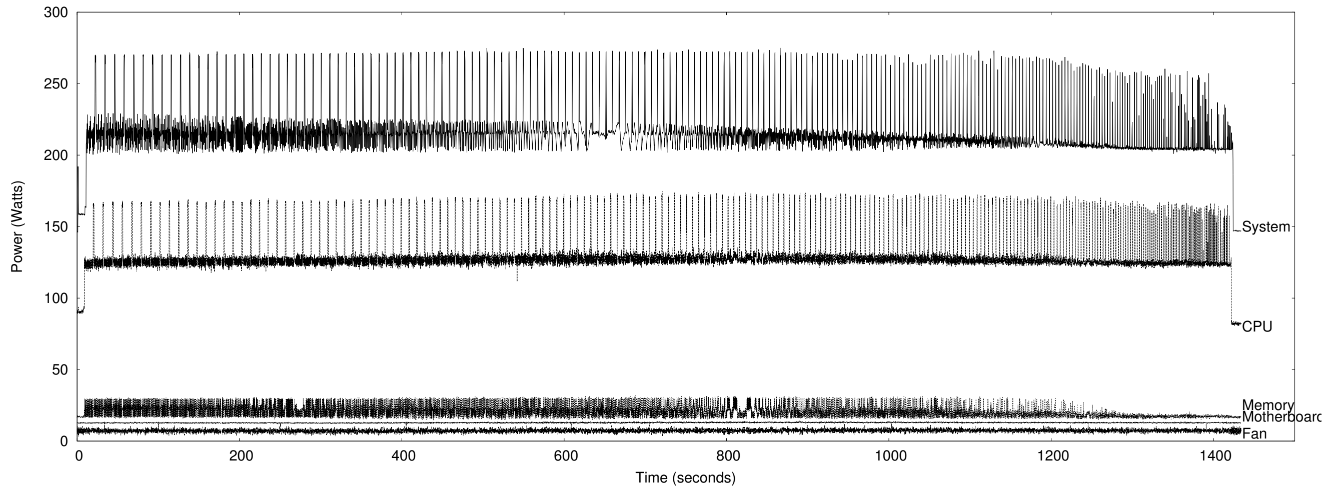
(b) PLASMA TRD.

Fig. 10 Power consumption of LAPACK and PLASMA TRD with a matrix size $N = 10000$ linked with GotoBLAS2 on the test system.

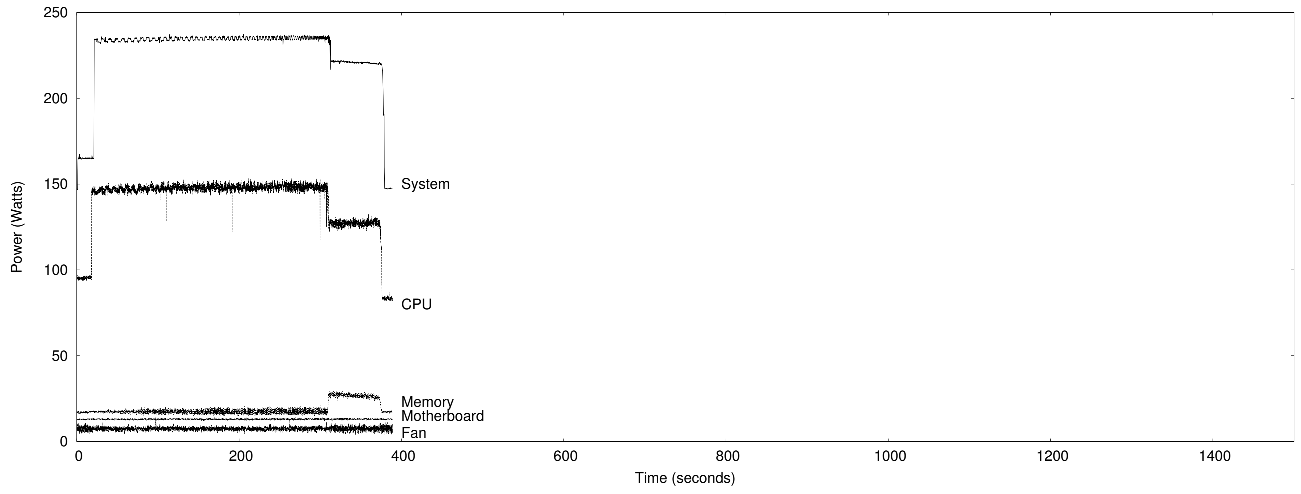
Acknowledgements The authors would like to thank Shirley Moore from the University of Tennessee, and Kirk Cameron and Hung-Ching Chang from the Department of Computer Science at Virginia Tech , for granting us access to their experimental platforms, in the context of the MuMI (Multicore application Modeling Infrastructure) project [18].

References

1. Agullo, E., Hadri, B., Ltaief, H., Dongarrra, J.: Comparative study of one-sided factorizations with multiple software packages on multi-core hardware. In: SC '09: Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis, pp. 1–12. ACM, New York, NY, USA (2009). DOI http://doi.acm.org/10.1145/1654059.1654080
2. Anderson, E., Bai, Z., Bischof, C., Blackford, S.L., Demmel, J.W., Dongarra, J.J., Croz, J.D., Greenbaum, A., Hammarling, S., McKenney, A., Sorensen, D.C.: LAPACK User's Guide, Third edn. Society for Industrial and Applied Mathematics, Philadelphia (1999)
3. Anzt, H., Rocker, B., Heuveline, V.: Energy efficiency of mixed precision iterative refinement methods using hybrid hardware platforms - An evaluation of different solver and hardware configurations. Computer Science - R&D **25**(3-4), 141–148 (2010). URL `http://dx.doi.org/10.1007/s00450-010-0124-2`
4. Bekas, C., Curioni, A.: A New Energy Aware Performance Metric. Computer Science - R&D **25**(3-4), 187–195 (2010). URL `http://dx.doi.org/10.1007/s00450-010-0119-z`
5. Bischof, C.H., Lang, B., Sun, X.: Algorithm 807: The sbr toolbox—software for successive band reduction. ACM Trans. Math. Softw. **26**(4), 602–616 (2000). DOI http://doi.acm.org/10.1145/365723.365736
6. Buttari, A., Dongarra, J., Langou, J., Langou, J., Luszczek, P., Kurzak, J.: Mixed precision iterative refinement techniques for the solution of dense linear systems. IJHPCA **21**(4), 457–466 (2007). URL `http://dx.doi.org/10.1177/1094342007084026`
7. Buttari, A., Langou, J., Kurzak, J., Dongarra, J.: A class of parallel tiled linear algebra algorithms for multicore architectures. Parallel Computing **35**(1), 38–53 (2009)



(a) LAPACK BRD.



(b) PLASMA BRD.

Fig. 11 Power consumption of LAPACK and PLASMA BRD with a matrix size $N = 10000$ linked with GotoBLAS2 on the test system.

8. Chen, G., Malkowski, K., Kandemir, M.T., Raghavan, P.: Reducing power with performance constraints for parallel sparse applications. In: IPDPS. IEEE Computer Society (2005). URL `http://doi.ieeecomputersociety.org/10.1109/IPDPS.2005.378`
9. Ding, Y., Malkowski, K., Raghavan, P., Kandemir, M.T.: Towards energy efficient scaling of scientific codes. In: IPDPS, pp. 1–8. IEEE (2008). URL `http://dx.doi.org/10.1109/IPDPS.2008.4536217`
10. Freeh, V.W., Lowenthal, D.K.: Using multiple energy gears in MPI programs on a power-scalable cluster. In: K. Pingali, K.A. Yelick, A.S. Grimshaw (eds.) Proceedings of the ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (10th PPOPP'2005), ACM SIGPLAN Notices, pp. 164–173. Chicago, IL, USA (2005). Published as Proceedings of the ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (10th PPOPP'2005), ACM SIGPLAN Notices, volume 40, number 6
11. Ge, R., Feng, X., Song, S., Chang, H.C., Li, D., Cameron, K.W.: Powerpack: Energy profiling and analysis of high-performance systems and applications. IEEE Transactions on Parallel and Distributed Systems **PDS-21**(5), 658–671 (2010)
12. Golub, G.H., Van Loan, C.F.: Matrix Computation, third edn. John Hopkins Studies in the Mathematical Sciences. Johns Hopkins University Press, Baltimore, Maryland (1996)
13. Kågström, B., Kressner, D., Quintana-Ortí, E., Quintana-Ortí, G.: Blocked Algorithms for the Reduction to Hessenberg-Triangular Form Revisited. BIT Numerical Mathematics **48**, 563–584 (2008)
14. Kappiah, N., Freeh, V.W., Lowenthal, D.K.: Just In Time Dynamic Voltage Scaling: Exploiting Inter-Node Slack to Save Energy in MPI Programs. In: SC, p. 33. IEEE Computer Society (2005). URL `http://doi.acm.org/10.1145/1105760.1105797`
15. Kogge, P., Bergman, K., Borkar, S., Campbell, D., Carlson, W., Dally, W., Denneau, M., Franzon, P., Harrod, W., Hill, K., Hiller, J., Karp, S., Keckler, S., Klein, D., Lucas, R., Richards, M., Scarpelli, A., Scott, S., Snavely, A., Sterling, T., Williams, R.S., Yelick, K.: Exascale computing study: Technology challenges in achieving exascale systems. Tech. Rep. TR-2008-13, Department of Computer Science and Engineering, University of Notre Dame (2008)

16. Ltaief, H., Luszczek, P., Dongarra, J.: High performance bidiagonal reduction using tile algorithms on homogeneous multicore architectures. Submitted to ACM Transactions on Mathematical Software (2011)
17. Luszczek, P., Ltaief, H., Dongarra, J.: Two-stage tridiagonal reduction for dense symmetric matrices using tile algorithms on multicore architectures. In: Proceedings of IPDPS 2011. ACM, Anchorage, AK USA (2011)
18. Multicore application Modeling Infrastructure (MuMI) Project. http://www.mumi-tool.org
19. Sutter, H.: The free lunch is over: A fundamental turn toward concurrency in software. Dr. Dobb's Journal **30**(3) (2005). http://www.ddj.com/184405990
20. Trefethen, L.N., Bau, D.: Numerical Linear Algebra. SIAM, Philadelphia, PA (1997). URL http://www.siam.org/books/OT50/Index.htm
21. University of Tennessee Knoxville: PLASMA Users' Guide, Parallel Linear Algebra Software for Multicore Architectures, Version 2.3 (2010). Available electronically at http://icl.cs.utk.edu/projectsfiles/plasma/pdf/users_guide.pdf.