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

Power Profiling of Cholesky and QR Factorizations on Distributed Memory Systems

George Bosilca · Hatem Ltaief · Jack Dongarra

Received: date / Accepted: date

Abstract This paper presents the power profile of two high performance dense linear algebra libraries on distributed memory systems, ScaLAPACK and DPLASMA. From the algorithmic perspective, their methodologies are opposite. The former is based on block algorithms and relies on multithreaded BLAS and a two-dimensional block cyclic data distribution to achieve high parallel performance. The latter is based on tile algorithms running on top of a tile data layout and uses fine-grained task parallelism combined with a dynamic distributed scheduler (DAGuE) to leverage distributed memory systems. We present performance results (Gflop/s) as well as the power profile (Watts) of two common dense factorizations needed to solve linear systems of equations, namely Cholesky and QR. The reported numbers show that DPLASMA surpasses ScaLAPACK not only in terms of performance (up to 2X speedup) but also in terms of energy efficiency (up to 62%).

Keywords Power Profile Analysis · Dense Linear Algebra · Distributed Memory System · Dynamic Scheduler

1 Introduction and Motivation

More than eight years after traditional processor designs hit the edge of their power envelope, the path of extreme scale computational science to a 100 Petaflop (Pflop/s) system, which researchers had hoped to be using by the middle of the coming decade, has never looked steeper. On current high performance computing (HPC) systems, the application-architecture performance gap, i.e. the gap between theoretical peak performance and the performance realized by full applications, is already substantial. But with clock frequencies now capped below 4 GHz and trending downward, latencies in key areas (e.g. memory, bus, interconnect, power usage) expected to remain relatively stagnant, and software parallelism required to increase by at least three orders of magnitude, it is now reasonable to worry that a widening application-architecture performance gap will make such systems costly, unproductive to use and therefore irrational to build.

Moreover, as highlighted in the DOE report about Exascale Computing [9], the current power usage of most of the computing components is significantly higher than acceptable. Scaling today's systems, based on current technologies, to an exaflop level will require over one gigawatt of power. That amount of power is nearly equivalent to the output of Hoover Dam, and symbolizes an economical barrier that will be difficult to overcome. Therefore, reducing the power requirement by a significant factor, estimated at least 100 times, is one of the most important challenges for future hardware and software technologies.

However, future exascale machines are not the only opportunities to get more traction from more efficient hardware or software solutions. Already, large scale data centers, installed either at government institutions or private corporations selling them as services, are using a significant amount of power. Any reduction in their energy consumption will directly translate to lower usage costs and a more carbon-friendly footprint. At the

G. Bosilca and J. Dongarra
Innovative Computing Laboratory
University of Tennessee, Knoxville, USA
E-mail: bosilca,dongarra@eecs.utk.edu

H. Ltaief
Supercomputing Laboratory
KAUST, Thuwal, Saudi Arabia
E-mail: Hatem.Ltaief@kaust.edu.sa

hardware level, Google set a trend by building most of its data centers from scratch, from the building itself and the solar panels to feed it, down to the servers using energy-saving chips. At the software level, the picture is less clear. Most of the current efforts seem to be centralized around taking advantage of the power management techniques implemented in today's processors.

In this paper, we will focus on credible scenarios happening inside a High Performance Computing Center, where highly computational intensive parallel applications and simulations are the most common. Such environments are mainly government funded institutions, places where the next generation large scale environments are nurtured. Our motivation is to investigate the power usage and identify research directions on synchronous and asynchronous approaches for dense linear algebra numerical algorithms, using as benchmarks two of the most widely used one-sided factorizations for solving dense linear systems of equations for symmetric and non symmetric matrices [12,18]. This class of algorithms includes the Cholesky and QR factorizations, which actually correspond to the most computationally intensive step in solving various linear systems of equations. While they do not represent the most common pattern in data centers, they are still critical pieces as a large number of other software solutions extracts their parallel performance from an efficient dense linear algebra solvers. Additionally, due to their regularity and inherent load balancing characteristic, they offer a straightforward way to link energy consumption to algorithmic behaviors.

This is in this context that the authors propose to analyze the power efficiency of both factorizations using the distributed version of these two algorithms, as implemented in the high performance dense linear algebra libraries ScaLAPACK [7] and DPLASMA [4]. The aforementioned libraries employ diametrically opposite approaches in terms of algorithms, data layout and scheduling to effectively exploit the available hardware resources.

The remainder of this paper is organized as follows: Section 2 gives a detailed overview of previous research works in this area. Section 3 recalls the main features of the ScaLAPACK library. Section 4 describes the DPLASMA framework based on tile algorithms. Section 5 highlights the dynamic runtime engine used within DPLASMA. Section 6 outlines the experimental settings. Section 7 demonstrates the impact of the block/tile size as well as the number of cores on power consumption. Finally, Section 8 summarizes the results of this paper and presents some future work.

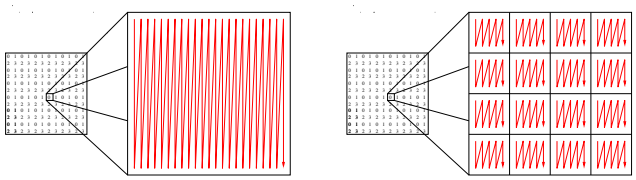
2 Background

During the last decade, the hardware evolution toward multicore architecture has pressed the scientific community to redesign their software stack in order to take advantage of the fine-grained parallelism. In the context of dense linear algebra, libraries such as LAPACK [3] for shared-memory platforms have shown their limitations on such architecture. One of the main reasons is that LAPACK relies on optimized multithreaded BLAS to exploit the underlying hardware. With the multicore era, parallelism has to be brought to the fore and exposed as much as possible through fine-grained computational tasks. The PLASMA [19] and FLAME [20] frameworks have anticipated the hardware landscape change by deeply rethinking the previous dense algorithms for solving linear systems of equations. The main concepts can be described in twofold: (a) reformulate the original algorithms using fine-grained computational kernels; the algorithms can then be depicted as a directed acyclic graph (DAG), where nodes represent tasks and edges represent data dependencies between them, and (b) a dynamic runtime system, which will schedule the different tasks, with respect to their data dependencies, on the available computing resources. These concepts have been successfully applied in solving linear systems of equations [6,17,2] as well as solving the symmetric eigenvalue problem and computing the singular value decompositions [13,14]. In a previous work by some of the authors [16], the power profile of the major routines of LAPACK and PLASMA libraries have been studied on a single shared-memory node. The authors were able to identify the critical algorithmic phases in both libraries thanks to the power profile. In this paper, the authors propose to extend the power analysis to the distributed memory system environment in the context of the ScaLAPACK [7] and DPLASMA [4] libraries. It is particularly noteworthy to mention that this is the first power usage study in the context of the distributed linear algebra libraries[1].

3 The ScaLAPACK Library

ScaLAPACK is a library of high-performance linear algebra routines for distributed-memory message passing MIMD computers and networks of workstations supporting PVM [11] and/or MPI [1]. It is a continuation of the LAPACK project, which designed and produced analogous software for workstations, vector supercomputers, and shared-memory parallel computers. The other extension to LAPACK is that ScaLAPACK uses a two-dimensional block cyclic distribution, which

[1] Up to our knowledge.



(a) Column-major data layout format within a block.

(b) Tile data layout format within a block.

Fig. 1 Two-Dimensional Block Cyclic Data Distribution.

improves the memory locality. Figure 1(a) shows an example of a two-dimensional block cyclic distribution using four processes. The data is contiguous in memory with each block following the column-major data layout format. The goal of this data distribution is to optimize the work load among each process, such that the idle time is minimized.

4 The DPLASMA Framework

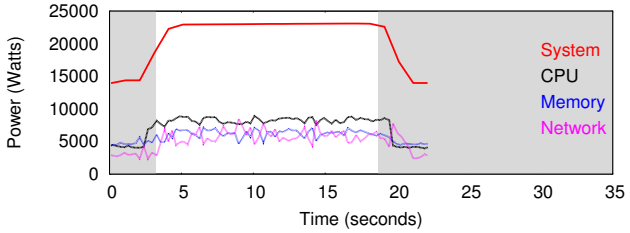
DPLASMA is a new library providing high-performance dense liner algebra routines for distributed-memory environments. Developed based on the mathematical algorithms implemented in the context of the PLASMA library, augmented with different layers of compiler technology to translate sequential tile-based dense linear algebra code into highly specialized DAGs, and executes the resulting DAG of tasks using the DAGuE runtime depicted in Section 5. In contrast to the ScaLAPACK library described in Section 3 and similarly to the PLASMA library, the DPLASMA library is using a **tile data layout** (TDL), where each tile data is contiguous in memory in order to improve the memory locality. In fact, due to the TDL local data distribution, the data storage in memory is similar to what is sketched in Figure 1(b). Moreover, the data can be distributed across the processes in any type of distribution, regular or irregular, and the algorithms will adapt dynamically to the data layout. For the algorithms studied in this paper, using a two-dimensional block cyclic distribution is common as this avoids load imbalance.

5 The DAGuE Engine

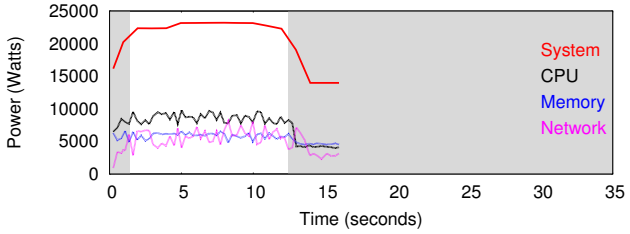
DAGuE [5] is a distributed runtime system designed to achieve extreme computational scalability by exploiting an algebraic representation of Direct Acyclic Graphs that efficiently captures the totality of tasks involved in a computation and the flow of data between them. Its primary goal is to maximize parallelism while automatically orchestrating task execution so as to minimize both data movements and load imbalance. Unlike other available DAG-based runtimes, the concise symbolic representation [8] of the algorithm that DAGuE uses minimizes the memory footprint required to completely express the total map of tasks; at the same time, it provides extreme flexibility during the scheduling process. This algebraic representation allows the DAG to be traversed at very high speed, while tracking any flow of data from task to task. By combining this underlying mechanism with an understanding of the specific hardware capabilities of the target architecture, DAGuE is able schedule tasks in ways that creatively adapt alternative work-stealing strategies to the unique features of the system. These capabilities enable DAGuE to optimize data movement between distributed memory computational resources, including both different nodes of the full system and different accelerators on the same node.

6 Experimental Settings

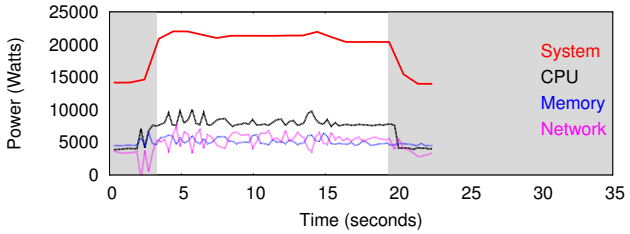
All experiments have been conducted on an x86-based cluster named systemg.cs.vt.edu from Virginia Tech, composed of 324 nodes with an InfiniBand network interconnect. Each node is composed of a dual-socket quad-core Intel Xeon 2.8GHz (2592 cores total) with 8GB of memory. It has over thirty power and thermal sensors per node, and relies on PowerPack [10] to obtain measurements of the major system components' power consumption, using power meters attached to the system's hardware. The PowerPack framework is a collection of software components, including libraries and APIs, which enable system component-level power profiling correlated to application functions. PowerPack allows the user to obtain direct measurements of the system's power consumption on a per-component basis, including the CPU, memory, hard disk, and motherboard. Furthermore, the efficiency of the AC/DC conversion happening at the power supply level is around 80%, which explains the gap between the overall system's power consumption and the sum of the different hardware components's power consumption (see Figures in Section 7). It is noteworthy to mention that there are no power sensors on the network interconnect cards, and it is not possible to monitor its power consumption through PowerPack. However, it is captured within the entire system's power consumption and it can therefore be extracted by subtracting the recorded power rate of the system components from the whole system power rate. In fact, this is one of the main contributions of this paper compared to previous work done on a single shared-memory system [16]. Furthermore, the Cholesky and QR factorizations are compute-bound numerical algorithms with a ratio flops per byte close to the matrix-matrix multiplication kernel (DGEMM). One of the main way to reduce their overall power con-



(a) block size = 32.

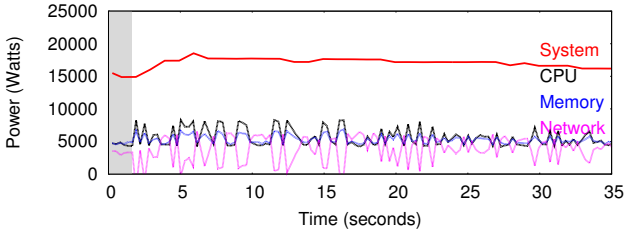


(b) block size = 128.

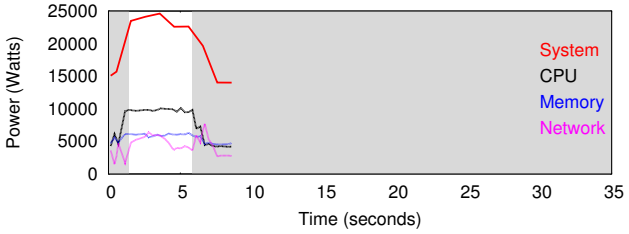


(c) block size = 512.

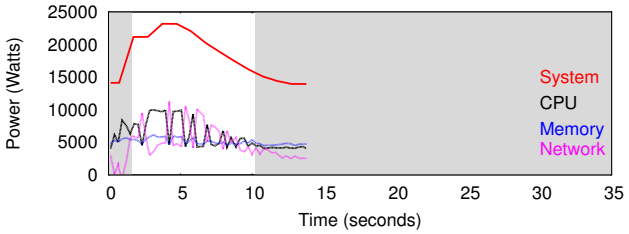
Fig. 2 Impact of the block size on the power profiles (Watts) of the ScaLAPACK Cholesky Factorization.



(a) tile size = 48.



(b) tile size = 192.



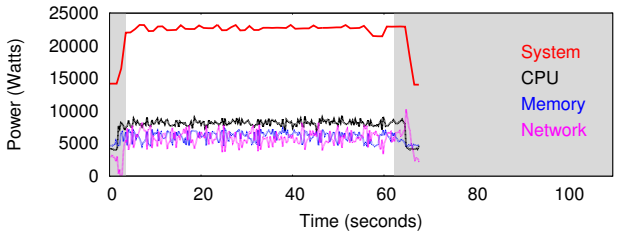
(c) tile size = 768.

Fig. 3 Impact of the tile size on the power profiles (Watts) of the DPLASMA Cholesky Factorization.

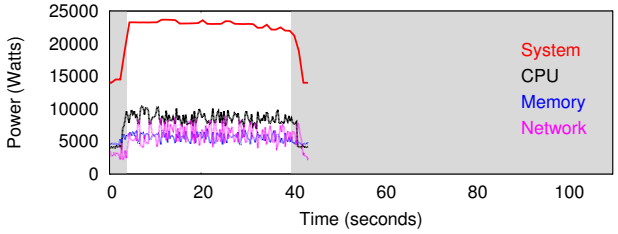
sumption is to increase the degree of parallelism such that there is enough concurrency to feed the available cores and therefore, to reduce the time to solution. And this is exactly what tile algorithms are able to achieve by breaking the computation into fine-grained numerical kernels. Consequently, there is no need to enable processor throttling in such algorithms through the commonly used technique of dynamic voltage frequency scaling (DVFS) [15]. These tile factorizations require rather the full processing power of the available cores until the end of the execution. However, this might not be always true for the other dense factorizations e.g., tridiagonal and bidiagonal reductions [13, 14] needed for eigenvalue problems and singular value decomposition, respectively, which are characterized by memory-bound computational stages. DVFS for these two-sided reductions may in fact play a major role in reducing their energy footprints. Their distributed version implementations and their actual integrations into the DPLASMA library are very challenging and still under work in progress. Finally, ScaLAPACK V2.0.1 and DPLASMA 1.0-rc1 have been compiled against the sequential version of the Intel MKL library V10.3 and OpenMPI V1.4.4.

7 Power Profiling Analysis

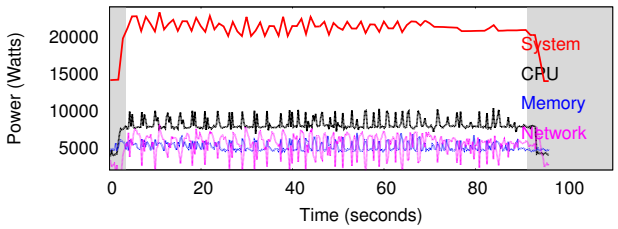
Understanding The Graphs As described in [5], the parallel performance of the mathematical algorithms are highly dependent on the blocking factor for ScaLAPACK and the tile size for DPLASMA. In both cases, these sizes impact the cache reuse and the balance between the computations and communications. In the context of the DPLASMA library, the tile size has a tremendous impact on the available parallelism and the number of tasks to be executed, as well as on the size and number of communications. A smaller tile size will generate more tasks, potentially exacerbating the overhead of the runtime due to low arithmetic intensity, while increasing the potential for late execution due to missing remote data (delayed communications). On the other side, a larger tile will limit the degree of parallelism in the algorithm and increase the load imbalance. A second parameter investigated is the number of cores, and its impact on the overall power consumption. For the sake of simplicity, the size of the matrix has



(a) block size = 32.

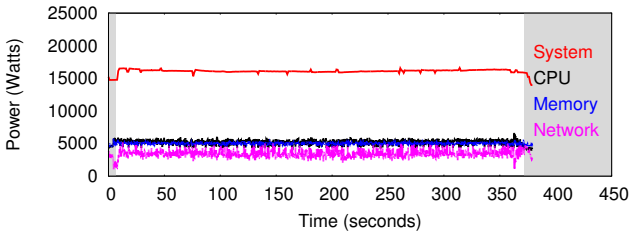


(b) block size = 128.

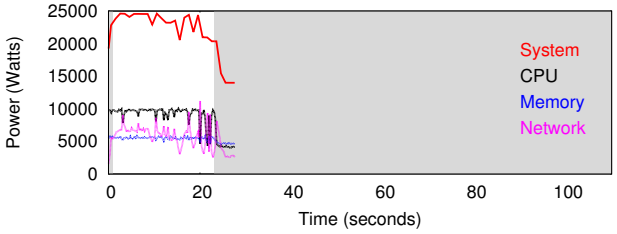


(c) block size = 512.

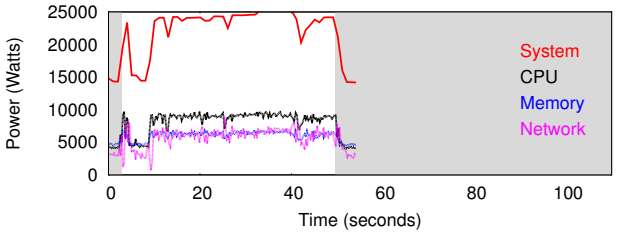
Fig. 4 Impact of the block size on the power profiles (Watts) of the ScaLAPACK QR Factorization.



(a) tile size = 48.



(b) tile size = 192.



(c) tile size = 768.

Fig. 5 Impact of the tile size on the power profiles (Watts) of the DPLASMA QR Factorization.

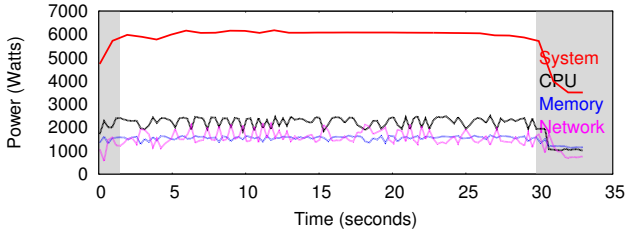
been set to $N = 40000$ for all experiments presented below. We have also reduced the power consumption monitors to only two components using the PowerPack framework i.e., CPU and memory, in addition to monitoring the entire system power consumption. We have disregarded the disk and the motherboard since they are not relevant. We have also provided an estimation of the network interconnect power consumption using the collected data from PowerPack power sensors. All power consumption measurements presented in the Figures below are for the entire allocated resource partition power consumption (up to 64 allocated nodes). In addition, areas non essential to the understanding of the algorithms pictured in the graphs (e.g., matrix allocation and initialization and different validity checks) have been grayed out.

7.1 Impact of Block/Tile Size on Power Profiles

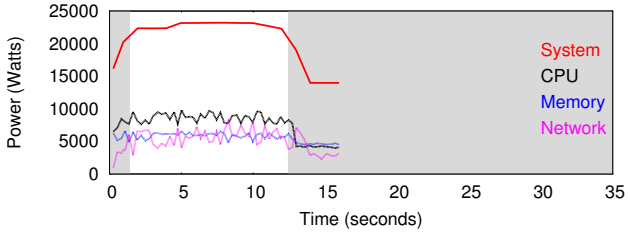
Here, we present the power consumption profiles for the Cholesky and QR factorizations implemented within ScaLAPACK and DPLASMA, while varying the blocking factor and the tile size, respectively. Such profiles allow us not only to observe temporal changes of the power drawn by the hardware, but, primarily, give us additional information about the usage of system components and how changes in this usage are influenced by various algorithms or steps in the algorithms.

ScaLAPACK. Figures 2 and 4 show the power rate of ScaLAPACK Cholesky and QR. Small block sizes affect the power consumption as the algorithm performance is mostly driven by the extensive communication load. In the same manner, a large block size hinders the performance due to a low degree of generated parallelism. Choosing a block size $b = 128$ seems to be the appropriate number to get the best out of ScaLAPACK for the matrix size considered. The power curve of the network interconnect is more dense for the QR factorization than the Cholesky factorizations mainly due to the number of global communications required when it comes to reducing non-symmetric matrices.

DPLASMA. Figures 3 and 5 underline the power profiles of DPLASMA QR and Cholesky. In particular, Figures 3(a) and 5(a) show the negative impact of using a very small tile size on the computation time. The large number of communications and their latencies, along with the low computation intensity of the kernels

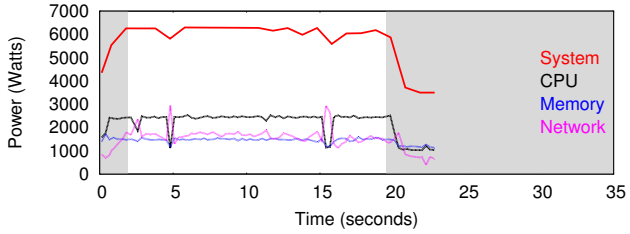


(a) Number of cores = 128.

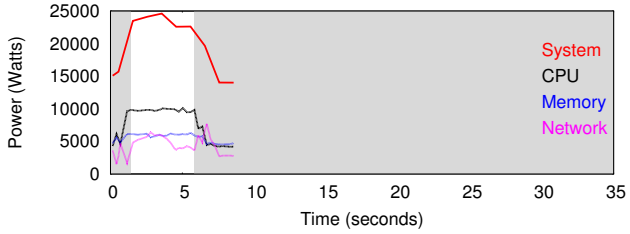


(b) Number of cores = 512.

Fig. 6 Impact of the number of cores on the power profiles (Watts) of the ScaLAPACK Cholesky Factorization.

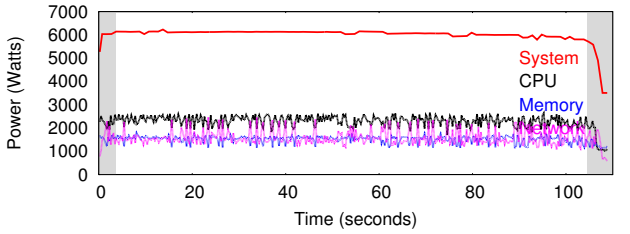


(a) Number of cores = 128.

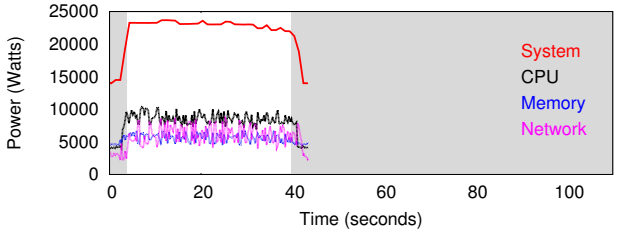


(b) Number of cores = 512.

Fig. 7 Impact of the number of cores on the power profiles (Watts) of the DPLASMA Cholesky Factorization.

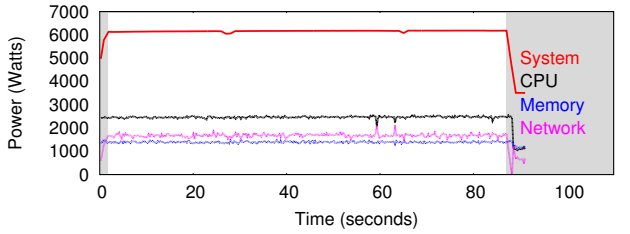


(a) Number of cores = 128.

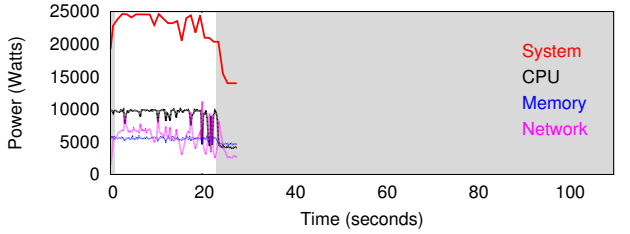


(b) Number of cores = 512.

Fig. 8 Impact of the number of cores on the power profiles (Watts) of the ScaLAPACK QR Factorization.



(a) Number of cores = 128.



(b) Number of cores = 512.

Fig. 9 Impact of the number of cores on the power profiles (Watts) of the DPLASMA QR Factorization.

(see the low CPU power rate), induce considerable overhead, more than what can be dealt with by the dynamic scheduler in the runtime, limiting its efficiency. In this particular case, a more power-aware type of scheduler may improve the performance and power usage, to a level similar to ScaLAPACK. While Figure 3(a) shows only a snapshot of the Cholesky power profile due to its excessive execution time, Figure 5(a) presents the entire snapshot of QR with an elapsed time of around 375 seconds. The power curves from the subsequent Figures shrink as the tile size increases, and again, as seen above for ScaLAPACK, after passing a certain threshold $t = 192$, the power consumptions start widening and decreasing at the same time, since processes are running out of work for such a coarse-grain tile size. Increasing the size of the matrix will certainly alleviate this problem, but not solve it entirely. The power curves of the network interconnect are also smoother than those from ScaLAPACK. The fine-grained computations in DPLASMA generate fine-grained communications, which can be better overlapped, compared to ScaLAPACK block algorithms.

7.2 Impact of the Core Number on Power Profiles

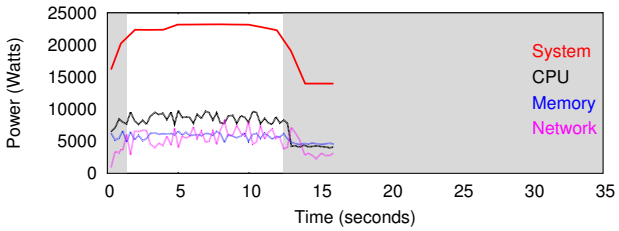
This Section describes the impact of the core number on the power profiles of ScaLAPACK and DPLASMA Cholesky/QR factorizations, in the context of a strong scaling analysis, i.e. the matrix size remains constant while the number of computational resources varies.

ScaLAPACK. Figures 6 and 8 show the power evolution of ScaLAPACK Cholesky/QR when changing the total number of cores. As the number of cores increases, the time to solution is shortened accordingly and the power consumption is therefore enhanced. However, the correlation between the power usage and the number of cores is not linear.

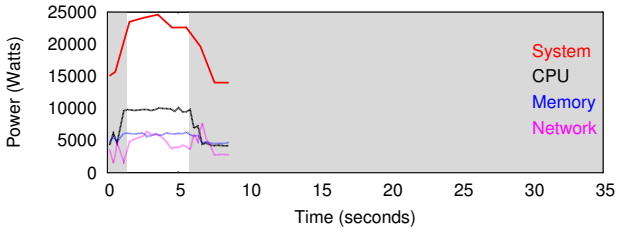
DPLASMA. Figures 7 and 9 show a somehow similar power profile behavior, as the number of cores increases. As the tile size has been optimally chosen based on the previous measurements, a significant amount of parallelism is available. Adding more cores would only improve the time to solution accordingly.

7.3 Power Profile Comparisons

Figures 10 and 11 compare the power profiles of both algorithms, as implemented in ScaLAPACK and DPLASMA. DPLASMA Cholesky consumes up to 62% less energy than ScaLAPACK Cholesky and DPLASMA QR consumes up to 40% less energy than ScaLAPACK QR for the matrix size considered. Furthermore, Figure 12 reports the total amount of energy needed for each test depending on the number of cores used. The total energy consumption of ScaLAPACK increases as more cores are used to compute the Cholesky/QR factorizations. However, we see a different behavior for DPLASMA total power consumption. For example, for the DPLASMA Cholesky factorization, the total energy consumption decreases when doubling the number of cores from 128 to 256, due to the super linear speedup seen in Figure 7(a), where the elapsed time starts at 20 seconds and goes down to 8 seconds. Doubling further the number of cores to 512 does not actually bring more performance (the scalability is limited due to the small workload per cores for the matrix size considered since only the upper/lower part of the symmetric matrix is factorized) but does increase the total energy consumption compared to the experiments with 256 cores. Regarding the DPLASMA QR factorization, the total energy consumption stays roughly the same regardless of the number of cores used. Indeed, the QR factorization is very compute intensive (four times the Cholesky algorithmic complexity), besides the high degree of parallelism for such a matrix size.

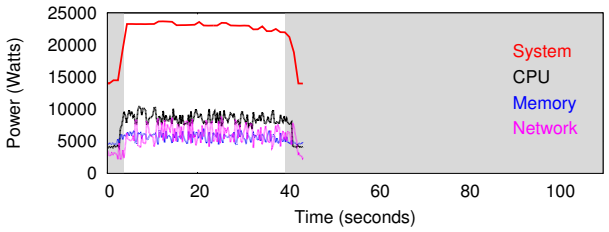


(a) ScaLAPACK.

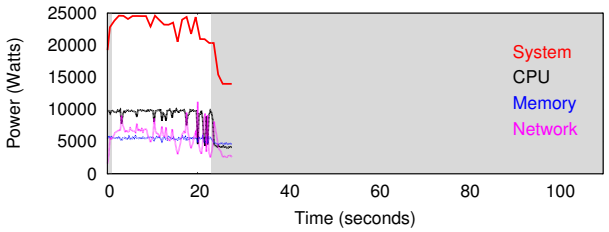


(b) DPLASMA.

Fig. 10 Power Profiles of the Cholesky Factorization.



(a) ScaLAPACK.



(b) DPLASMA.

Fig. 11 Power Profiles of the QR Factorization.

8 Conclusion

This paper presents the power profiles of two high performance one-sided dense linear algebra factorizations, namely Cholesky and QR, on distributed memory systems in the context of ScaLAPACK and DPLASMA libraries. From a pure performance point of view, the results demonstrate that a completely asynchronous approach, such as the data-flow based approach in DPLASMA, achieve significantly higher efficiencies, delivering up to two-fold speedup. However, due to the asynchronous behavior of the algorithms and a scheduler targeted toward high efficiency, all available hardware components are in use during the entire execution, lead-

# Cores	Library	Cholesky	QR
128	ScaLAPACK	192000	672000
	DPLASMA	128000	540000
256	ScaLAPACK	240000	816000
	DPLASMA	96000	540000
512	ScaLAPACK	325000	1000000
	DPLASMA	125000	576000

Fig. 12 Total amount of energy (joule) used for each test based on the number of cores

ing to a higher power usage. Nevertheless, combining the execution time decrease with the power usage increase, the DPLASMA algorithms decrease the energy consumption up to 62% compared to ScaLAPACK. This is interesting evidence that the shift to more hardware challenging environments, if accompagnied by asynchronous execution runtime and adapted algorithms can lead to significantly improved efficiencies and power saving.

In the future, the authors plan to extend this study with other dense reductions (*two-sided* factorizations) required for computing the eigenvalues and singular values. Based on a two-stage approach, the original matrix is first reduced to a band using compute intensive kernels. Then, the band matrix is further reduced to the corresponding condensed form using memory-bound kernels where the degree of parallelism is extremely low. Therefore, the DAGuE runtime could exploit the nature of the algorithms and detect that the second stage, being less compute intensive, requires a smaller number of cores. The engine could then decide to dynamically adapt the frequencies of the core and eventually turn them off using Dynamic Voltage Frequency Scaling [15], a commonly used technique with which it is possible to achieve reduction of energy consumption.

Acknowledgment

The authors would like to thank Pr. Kirk Cameron from the Department of Computer Science at Virginia Tech, for granting access to his platform.

References

1. MPI-2: Extensions to the message passing interface standard. http://www.mpi-forum.org/ (1997)
2. Agullo, E., Hadri, B., Ltaief, H., Dongarra, J.: 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)
3. 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, 3rd edn. Society for Industrial and Applied Mathematics, Philadelphia (1999)
4. Bosilca, G., Bouteiller, A., Danalis, A., Faverge, M., Haidar, A., Herault, T., Kurzak, J., Langou, J., Lemarinier, P., Ltaief, H., Luszczek, P., YarKhan, A., Dongarra, J.: Flexible Development of Dense Linear Algebra Algorithms on Massively Parallel Architectures with DPLASMA. In: PDSEC-11. ACM (2011)
5. Bosilca, G., Bouteiller, A., Herault, T., Lemarinier, P., Dongarra, J.: DAGuE: A generic distributed DAG engine for high performance computing. HIPS (2011)
6. 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)
7. Choi, J., Demmel, J., Dhillon, I., Dongarra, J., Ostrouchov, S., Petitet, A., Stanley, K., Walker, D., Whaley, R.C.: ScaLAPACK, a portable linear algebra library for distributed memory computers-design issues and performance. Computer Physics Communications **97**(1-2)
8. Cosnard, M., Jeannot, E.: Compact DAG representation and its dynamic scheduling. Journal of Parallel and Distributed Computing **58**, 487–514 (1999)
9. Dongarra, J., Beckman, P.: The International Exascale Software Roadmap. International Journal of High Performance Computer Applications **25**(1) (2011)
10. 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 **21**(5), 658–671
11. Geist, A., Beguelin, A., Dongarra, J., Jiang, W., Manchek, R., Sunderam, V.: PVM: Parallel Virtual Machine. A Users' Guide and Tutorial for Networked Parallel Computing. MIT Press, Cambridge, MA (1994)
12. Golub, G.H., Van Loan, C.F.: Matrix Computations, third edn. John Hopkins Studies in the Mathematical Sciences. Baltimore, Maryland (1996)
13. Haidar, A., Ltaief, H., Dongarra, J.: Parallel Reduction to Condensed Forms for Symmetric Eigenvalue Problems using Aggregated Fine-Grained and Memory-Aware Kernels. In: SC11. Seattle, WA, USA
14. Haidar, A., Ltaief, H., Luszczek, P., Dongarra, J.: A Comprehensive Study of Task Coalescing for Selecting Parallelism Granularity in a Two-Stage Bidiagonal Reduction. In: IPDPS'12. Shanghai, China (2012)
15. 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)
16. Ltaief, H., Luszczek, P., Dongarra, J.: 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 HPC. Hamburg, Germany (2011)
17. Quintana-Ortí, G., Quintana-Ortí, E.S., Chan, E., van de Geijn, R.A., F. G. Van Zee: Scheduling of QR factorization algorithms on SMP and multi-core architectures. In: PDP, pp. 301–310. IEEE Computer Society (2008)
18. Trefethen, L.N., Bau, D.: Numerical Linear Algebra. SIAM, Philadelphia, PA (1997). URL http://www.siam.org/books/OT50/Index.htm
19. University of Tennessee Knoxville: PLASMA Users' Guide, Parallel Linear Algebra Software for Multicore Architectures, Version 2.4 (2011)
20. Zee, F.G.V., Chan, E., van de Geijn, R.A., Quintana-Orti, E.S., Quintana-Orti, G.: The `libflame` library for dense matrix computations. Computing in Science and Engineering **11**(6), 56–63 (2009)