Informe Técnico ICC 2011-07-01

Reducing Power Consumption of the LU Factorization with Partial Pivoting on Multi-Core Processors

Pedro Alonso, Manuel F. Dolz, Rafael Mayo, Enrique S. Quintana-Ortí

Julio de 2011

Departamento de Ingeniería y Ciencia de los Computadores

Correo electrónico: palonso@dsic.upv.es,
{dolzm,mayo,quintana}@icc.uji.es

Universidad Jaime I
Campus de Riu Sec, s/n
12.071 - Castellón
España

Reducing Power Consumption of the LU Factorization with Partial Pivoting on Multi-Core Processors

Pedro Alonso[1],
Manuel F. Dolz[2],
Rafael Mayo[3],
Enrique S. Quintana-Ortí[4]

Abstract:

In this paper we analyze the trade-off between energy and performance for a data-parallel execution of the LU factorization with partial pivoting on a multi-core processor. To improve power efficiency, we adapt the runtime in charge of controlling the concurrent execution of the algorithm so as to leverage DVFS by activating/blocking idle threads. For a CPU-bounded operation like the LU factorization, experiments on an AMD 8-core processor report an average reduction around 5% in energy consumption in exchange for a minor, in some cases negligible, increase in the execution time.

Keywords:

Power-aware computing, DVFS, linear systems, LU factorization, multi-core processors.

[1] Departamento de Sistemas Informáticos y Computación, Universidad Politécnica de Valencia
E-mail: `palonso@dsic.upv.es`

[2] Departamento de Ingeniería y Ciencia de los Computadores
E-mail: `dolzm@icc.uji.es`

[3] Departamento de Ingeniería y Ciencia de los Computadores
E-mail: `mayo@icc.uji.es`

[4] Departamento de Ingeniería y Ciencia de los Computadores
E-mail: `quintana@icc.uji.es`

Reducción del Consumo Energético de la Factorizacin LU con Pivotamiento Parcial en Procesadores Multinúcleo.

Pedro Alonso[5],
Manuel F. Dolz[6],
Rafael Mayo[7],
Enrique S. Quintana-Ortí[8]

Resumen:

En este trabajo se analiza el compromiso entre el consumo de energía y el rendimiento de la ejecución paralela del algoritmo que implementa la factorización LU con pivotamiento parcial en procesadores multiúcleo. Para mejorar la eficiencia energética, se ha adaptado el *runtime* encargado de controlar la ejecución concurrente del algoritmo con el fin de aprovechar la técnica DVFS a través de la activación y el bloqueo de los hilos que se encuentran en estado ocioso. Para un algoritmo limitado por CPU tal como la factorización LU, los experimentos con un procesador AMD de 8 núcleos proporcionan una reducción promedio de alrededor del 5% en el consumo de energía a cambio de un menor, en algunos casos insignificante, aumento del tiempo de ejecución.

Palabras clave:
Computación consciente del consumo, DVFS, sistemas lineales, factorización LU, procesadores multinúcleo.

[5] Departamento de Sistemas Informáticos y Computación, Universidad Politécnica de Valencia
E-mail: `palonso@dsic.upv.es`
[6] Departamento de Ingeniería y Ciencia de los Computadores
E-mail: `dolzm@icc.uji.es`
[7] Departamento de Ingeniería y Ciencia de los Computadores
E-mail: `mayo@icc.uji.es`
[8] Departamento de Ingeniería y Ciencia de los Computadores
E-mail: `quintana@icc.uji.es`

Reducing Power Consumption of the LU Factorization with Partial Pivoting on Multi-Core Processors

Pedro Alonso[1], Manuel F. Dolz[2], Rafael Mayo[2], and Enrique S. Quintana-Ortí[2]

[1] Depto. de Sistemas Informáticos y Computación
Univ. Politécnica de Valencia
46022–Valencia, Spain
palonso@dsic.upv.es
[2] Depto. de Ingeniería y Ciencia de los Computadores
Universidad Jaume I
12.071–Castellón, Spain
{dolzm,mayo,quintana}@icc.uji.es

Abstract In this paper we analyze the trade-off between energy and performance for a data-parallel execution of the LU factorization with partial pivoting on a multi-core processor. To improve power efficiency, we adapt the runtime in charge of controlling the concurrent execution of the algorithm so as to leverage DVFS by activating/blocking idle threads. For a CPU-bounded operation like the LU factorization, experiments on an AMD 8-core processor report an average reduction around 5% in energy consumption in exchange for a minor, in some cases negligible, increase in the execution time.

Key words: Power-aware computing, DVFS, linear systems, LU factorization, multi-core processors.

1 Introduction

As we target the Exaflop barrier, energy consumption is becoming a major concern. Reaching such an impressive performance rate using as-of-today most energy-efficient technology will require more than 590 MWatts [1], which basically amounts for 50% of the energy produced by a modern nuclear plant. Hardware designers have been highly sensible to power consumption during the past decade and, thus, state-of-the-art processors, memories and disks feature low-power modes to trade-off performance for energy. On the other hand, most current scientific, engineering and industrial applications running in high performance computing (HPC) centers are quite oblivious to the possibilities offered by the underlying hardware, in spite of the significant assets it can yield [2].

In this paper we address the power-aware solution of dense linear systems, via the LU factorization with partial pivoting, on multi-core processors. The inclusion of pivoting in the LU factorization is necessary to render the algorithm numerically stable in practice, though it hampers the degree of concurrency of the factorization.[3] To over-

[3] Although alternative pivoting schemes have been recently proposed to overcome the limited parallelism of the LU factorization with partial pivoting [8,3], this algorithm remains the method of reference for the solution of dense linear systems.

come this, the use of a *runtime* which divides the operation into work units, or *tasks*, dynamically keeps track of data dependencies among those, and schedules those tasks ready for execution to idle cores has revealed itself as a good compromise between coding complexity and performance. Runtimes for dense linear algebra include the SuperMatrix runtime for `libflame` [4] or Quark for PLASMA [7]. More general-purpose alternatives, applicable also to that particular domain, are being developed within the StarSs and StarPU frameworks.

There exist a good number of works which have analyzed the trade-off between energy and performance enabled by dynamic voltage-frequency scaling (DVFS); see, e.g., [5]. Some of these tackle the execution of a directed acyclic graph (DAG) representing tasks and data dependencies under certain conditions, in most cases reporting the theoretical gains which can be expected from this; see [6,9]. In our paper, we move one step forward, by incorporating DVFS into a production runtime for the domain of dense linear algebra, which allows us to provide experimental evidence of the impact on energy. In particular, our paper makes following contributions:

- We leverage DVFS to enable a power-efficient execution of the LU factorization, where idle threads are set into a blocking state and the corresponding cores are promoted into a low-power mode, without compromising the computational performance of the execution.
- Our power-saving strategies are integrated into the SuperMatrix runtime [8], offering a practical evaluation of the actual energy savings they can yield for dense linear algebra kernels.
- Experimental results on an 8-core AMD 6128 processor report the trade-off between energy and execution time, which can lead to significant power savings for the LU factorization with partial pivoting, around 5%, for a wide variety of problem dimensions, with a reduced impact on the time-to-response.

The remainder of this paper is structured as follows. In Section 2 we review the right-looking algorithmic variant for the LU factorization with partial pivoting and briefly summarize different alternatives to exploit the concurrency of this algorithm on a multi-threaded architecture. In Section 3 we present the power-saving techniques and our approach to accommodate them into the SuperMatrix framework. In Section 4 we report the computational and energy performances of the algorithm modulated by the power-aware runtime. Finally, a few concluding remarks close the paper in Section 5.

2 The LU Factorization with Partial Pivoting

2.1 The right-looking algorithm

The LU factorization with partial pivoting of a nonsingular matrix $A \in \mathbb{R}^{n \times n}$ is given by $PA = LU$, where $P \in \mathbb{R}^{n \times n}$ is a permutation matrix and $L/U \in \mathbb{R}^{n \times n}$ is unit lower/upper triangular.

Figure 1 illustrates the blocked right-looking algorithmic variant for the LU factorization using the FLAME notation. There, $n(\cdot)$ stands for the number of columns of its argument while $\text{TRILU}(\cdot)$ denotes the matrix consisting of the elements in the lower triangular part of its argument with the diagonal entries replaced by ones. We believe

Algorithm: $A := \text{LUP_BLK}(A)$

Partition $A \rightarrow \left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right)$
  where A_{TL} is 0×0
while $n(A_{TL}) < n(A)$ **do**
  Determine block size b
  Repartition

$$\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right) \rightarrow \left(\begin{array}{c|c|c} A_{00} & A_{01} & A_{02} \\ \hline A_{10} & A_{11} & A_{12} \\ \hline A_{20} & A_{21} & A_{22} \end{array}\right)$$

  where A_{11} is $b \times b$

$$\left(\begin{array}{c} A_{11} \\ \hline A_{21} \end{array}\right) := \text{LUP_UNB}\left(\begin{array}{c} A_{11} \\ \hline A_{21} \end{array}\right)$$
$$A_{12} := \text{TRILU}(A_{11})^{-1} A_{12} \quad (\texttt{trsm})$$
$$A_{22} := A_{22} - A_{21} A_{12} \quad (\texttt{gemm})$$

  Continue with

$$\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right) \leftarrow \left(\begin{array}{c|c|c} A_{00} & A_{01} & A_{02} \\ \hline A_{10} & A_{11} & A_{12} \\ \hline A_{20} & A_{21} & A_{22} \end{array}\right)$$

endwhile

Figure 1. Blocked algorithm for the LU factorization.

the rest of the notation is intuitive; see [8]. For simplicity, we dropped the application of pivoting from the algorithmic description. However, as argued above, pivoting determines the degree of concurrency as it dictates the panel-wise progress of the algorithm where, at each iteration, a panel of b columns is factored. The algorithm overwrites the strictly lower triangle/upper triangle of A with L/U. The factorization of the current panel, composed of A_{11} and A_{22}, can be obtained by calling an unblocked version of the algorithm ($b = 1$). Provided $1 \ll b \ll n$, the blocked algorithm in the figure performs $2n^3/3 + O(n^2)$ floating-point arithmetic operations (flops), mostly cast in terms of the matrix-matrix product (`gemm`) $A_{22} := A_{22} - A_{21}A_{12}$. This basic linear algebra operation is well known to yield high performance on current processors with a hierarchical organization of the memory as well as a highly efficient, relatively straight-forward parallelization on multi-threaded parallel architectures.

2.2 Parallelization

A trivial approach to execute the algorithm in Figure 1 in parallel is to link it with a multi-threaded implementation of `gemm`. Given that the major part of the computation is in the update of A_{22}, performance close to that of `gemm` can be expected when the number of threads/cores is relatively small. However, the factorization of the panel lies in the critical path of the algorithm so that, as the computational resources grow in number, the concurrency of this approach suffers. A simple look-ahead strategy can

be employed to tackle this problem as follows: the block to update is split as $A_{22} = \left[\bar{A}_L,\ \bar{A}_R\right]$, with $\bar{A}_L$ containing b columns, so that the update of $\bar{A}_L$ and its factorization are performed in parallel with the update of $\bar{A}_R$. In practice, this is analogous to shifting computations from the second iteration (which would compute the LU factorization of $\bar{A}_L$) into the first one. This scheme is repeated in subsequent iterations. More elaborate versions of look-ahead advance computations from iterations $i+1, i+2, \ldots$ into the i-th iteration, at the expense of greatly complicating the coding.

The SuperMatrix runtime unburdens the programmer from this complexity by automatically decomposing a `libflame` routine into tasks and keeping track of the dependencies during the execution. Thus, the operations are not executed in the order they appear in the code (control-flow parallelism) but in the order dictated by the data dependencies implicit to the algorithm (data-flow parallelism). Identification and scheduling of tasks are both done dynamically (i.e., at run time), without direct intervention from the programmer. For this purpose, SuperMatrix proceeds in two stages. During the initial stage, a symbolic execution of the code produces a directed acyclic graph (DAG) containing all tasks and dependencies. This information then dictates the feasible orderings in which tasks can be executed during the subsequent dispatch stage. To monitor progress, the SuperMatrix implementation utilizes a *pending list* which contains those tasks to be run but which depend on tasks not yet executed. At the beginning of this second stage, all tasks except for that corresponding to the factorization of the first panel are in the pending list. From this structure, a task is moved into the *ready list* when all its dependencies are fulfilled. Initially this list contains only the factorization of the first panel. Idle threads (one per core) continuously check the ready list for work (busy-wait or polling). When a thread acquires a task, it runs the corresponding job in the associated core and, upon completion, checks the tasks which were in the pending queue, moving them to the ready list in case all their dependencies are now satisfied. Details on the operation and implementation of the SuperMatrix runtime can be found in [8].

3 Accommodating Power-Aware Techniques into SuperMatrix

Modern Linux distributions leverage DVFS by providing different *governors* (`ondemand`, `powersave`, etc.) which set idle threads into power-hungry/power-save modes by increasing/reducing their operation frequency and voltage scaling. Operations as those in the level-3 BLAS (e.g., `gemm`) are highly CPU-bounded computations so that reducing the operation frequency/voltage incurs an increase in the execution time and, therefore, yields higher energy consumption, blurring all benefits of a lower-paced execution. Despite being a level-3 BLAS-based operation, the picture is different for the LU factorization with partial pivoting. Due to the existence of task dependencies, idle periods may appear during the computation of this operation. While this can be exploited by selecting a given governor for the entire application, in our case we apply a more effective approach, integrating it into the runtime system.

In particular, our first energy-saving technique works as follows: when a thread samples whether there is work in an empty ready list, the runtime immediately sets the operation frequency of the associated core to the lowest possible (system call `cpufreq`). Later, when the poll receives a positive answer, the frequency is raised back to the high-

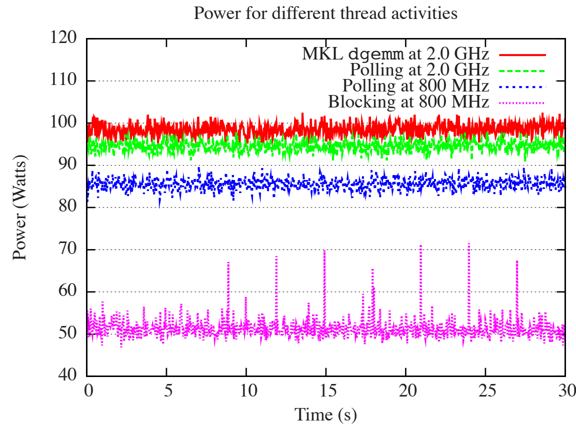

Figure 2. Power consumption of different actions performed by threads.

est, in preparation for the execution of the corresponding job. This operational mode implies a reduction in the polling rate which is beneficial (polling itself can be viewed as a waste of energy). Figure 2 illustrates the difference in energy consumption between a thread that performs polling at 2.0 GHz and one that does the same at 800 MHz on an 8-core AMD 6128 processor (when all remaining cores are idle): from around 95 Watts to less than 90 Watts. Note also the power consumption of a thread performing polling at the highest frequency is only slightly smaller than that of one performing useful work like, e.g. a matrix-matrix product (MKL `dgemm`). The results in that plot also reveal a complementary/alternative strategy. In particular, observe that a thread performing the busy-wait corresponding to polling, even at 800 MHz, still consumes a considerable amount of energy. However, when the same thread is blocked, the consumption is decreased significantly, to 50–55 Watts.

Our second power saving technique replaces the polling state of "inactive" threads by a power-friendly, blocking one. Whether these theoretical savings yield an actual gain will depend, however, in the existence/length of idle periods during the execution of the algorithm and the overhead of blocking/activating a thread. In our implementation we employ POSIX semaphores to control the active threads. Now, when a thread polling for a new job from the ready list receives a negative answer (there is no task ready for execution at the moment), it blocks itself (with the system call `sem_wait()`). When a thread completes the execution of a task, it updates the dependencies of the tasks in the pending list; besides, in case this implies moving k tasks from the pending list to the ready list, this thread will also enforce that there exist k active threads (using system call `sem_post()` to activate other threads, if necessary). This simple mechanism ensures that there is basically one active thread per task in the ready list and, key to power conservation, that no continuous polling is being done on an empty list.

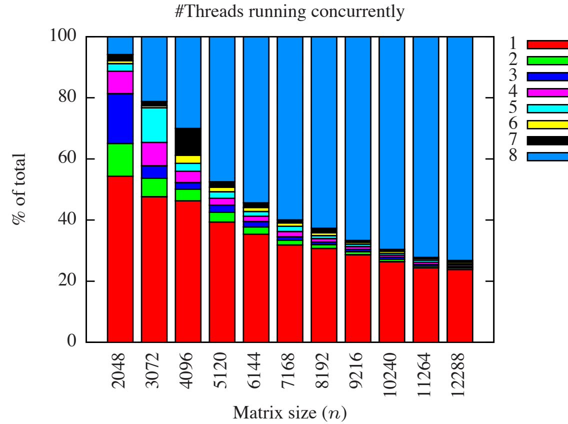

Figure 3. Thread activity during the execution of the LU factorization with partial pivoting.

4 Experimental Results

All experiments reported in this section were obtained using IEEE double-precision arithmetic on an 8-core AMD 6128 processor (2.0 GHz) with 24 Gbytes of RAM. The system runs a Linux Ubuntu 10.04 distribution. Highly tuned implementations of BLAS and LAPACK were those in MKL 10.2.4. A modified version of SuperMatrix runtime in `libflame` version 5.0–r5587 was designed to leverage the two power-saving techniques described in the previous section. Execution times/power measurements correspond to that of routine `FLASH_LU_piv` (for the blocked right-looking variant of the LU factorization with partial pivoting) from this library, linked to the original and power-aware implementations of the runtime. Matrices were generated with random entries uniformly distributed in $[0, 1]$, so that pivoting actually occurs during the computation of the triangular factors. Our evaluation includes a variety of (square) matrix dimensions ranging from 2,048 to 12,288 and the block size $b = 512$. This block dimension was close to optimal for most kernels involved in this factorization.

Power was measured using an internal DC powermeter. This is an ASIC operating with a sampling frequency of 25 Hz, directly attached to the lines connecting the power supply unit and the motherboard (chipset plus processors). All tests were repeated 30 times and average values are reported.

Our first experiment evaluates the existence and length of idle periods during the computation of the LU factorization with partial pivoting on the 8 cores of the AMD processor, with parallelism extracted by the SuperMatrix runtime. The results in Figure 3 gather the results from this evaluation. Let's examine the two extreme cases: when the problem size is (n=)2,048, 54% of the time there is a single active thread and only 6% of the time all threads are performing work. On the other hand, when the problem size is much larger, e.g. n=10,240, about 26% of the time there is one active and most of the remaining period all 8 threads are running. The conclusion from this experiment

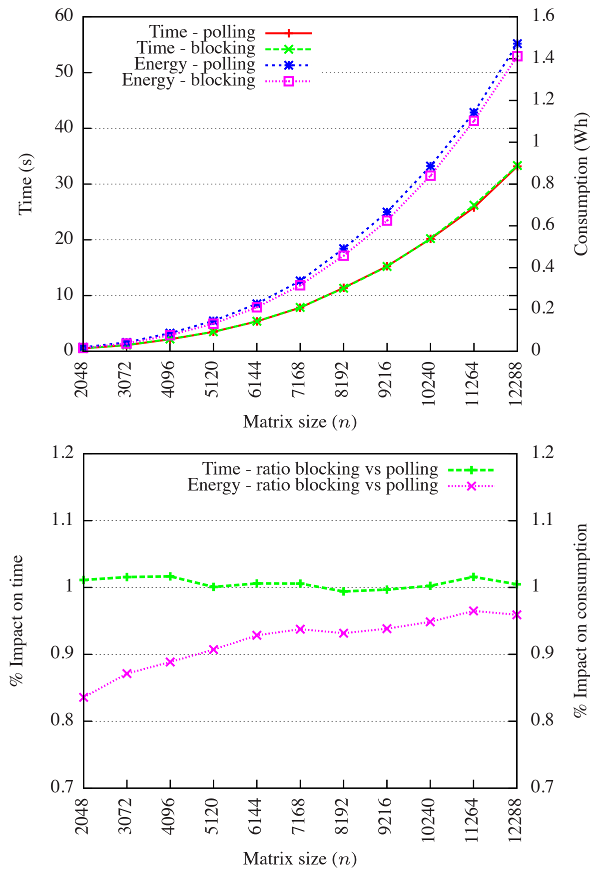

Figure 4. Evaluation of the impact on time and energy of the power-saving strategies.

is that there exists indeed the opportunity of saving energy by carefully controlling the level of activity of idle threads.

The second experiment measures the actual gains that can be attained by an energy-aware approach. In this case we compare the original SuperMatrix runtime with a modified variant that employs semaphores to block idle threads (second technique described in the previous section). We leave the operation of DVFS in the hands of the OS by setting the governor to `ondemand`[4] with the default policies to raise/lower frequency (namely, when the core load exceeds/falls below 95%, the frequency is set to the highest/lowest possible; the OS samples core activity with a frequency of 10 ms.). In this mode, a polling thread is active and, thus, the corresponding core remains at 2.0 GHz; a thread blocked in a semaphore, instead, is detected by the OS which lowers the oper-

[4] Although we evaluated several other governors (`powersave`, `conservative`, etc.), they all offered a significant increase in the execution time which resulted in a much higher power consumption.

ation frequency of the associated core to 800 MHz. We will refer to these two versions of SuperMatrix as "polling" and "blocking". Figure 4 illustrates the effect of the power-saving strategy both in the execution time and energy consumption. The impact on the execution time is small, with an increase of 1.6% at most for the smallest problem sizes while, for others, there is no appreciable difference between the two strategies. On the other hand, the effect on power efficiency is much more relevant. For the smallest problem sizes, the number of tasks is relatively low compared with the number of threads, which results in gaps (idle periods) during the execution of the algorithm, and this translates into significant energy savings. These inactive periods are reduced as the problem dimension grows, and the power savings tend to stabilize around 5%.

5 Concluding Remarks

While CPU-bounded computations like, e.g., the matrix-matrix product should be run at the highest frequency so as to reduce execution time and, therefore, energy consumption, this paper addresses this issue for complex dense linear algebra operations, where idle periods appear during the execution of the corresponding algorithm due to data dependencies. In particular, we address the LU factorization with partial pivoting, and a parallel data-flow runtime-assisted (SuperMatrix) from a production library like `libflame`, to analyze the trade-off between performance and energy on a multi-core platform. Our results show that, for large problem sizes, it is possible to leverage these inactive periods, reducing energy consumption around 5% with a negligible impact on the execution time.

Acknowledgments

The researchers from Univ. Jaume I were supported by project CICYT TIN2008-06570-C04 and FEDER.

References

1. The Green500 list, 2010. Available at `http://www.green500.org`.
2. Susanne Albers. Energy-efficient algorithms. *Commun. ACM*, 53:86–96, May 2010.
3. J. Demmel, L. Grigori, M.F. Hoemmen, and J. Langou. Communication-optimal parallel and sequential QR and LU factorizations. Technical Report UCB/EECS-2008-89, EECS Department, Univ. California, Berkeley, 2008.
4. FLAME project home page. `http://www.cs.utexas.edu/users/flame/`.
5. V.W. Freeh, David K. Lowenthal, Feng Pan, Nandini Kappiah, Rob Springer, Barry L. Rountree, and Mark E. Femal. Analyzing the energy-time trade-off in high-performance computing applications. *IEEE Trans. Parallel Distrib. Syst.*, 18:835–848, June 2007.
6. D. King, I. Ahmad, and H.F. Sheikh. Stretch and compress based re-scheduling techniques for minimizing the execution times of DAGs on multi-core processors under energy constraints. In *International Conference on Green Computing*, pages 49–60. IEEE, 2010.
7. PLASMA project home page. `http://icl.cs.utk.edu/plasma/`.

8. G. Quintana-Ortí, E.S. Quintana-Ortí, R.A. van de Geijn, F.G. Van Zee, and E. Chan. Programming matrix algorithms-by-blocks for thread-level parallelism. *ACM Trans. Math. Softw.*, 36(3):14:1–14:26, 2009.
9. V. Shekar and B. Izadi. Energy aware scheduling for DAG structured applications on heterogeneous and DVS enabled processors. In *Green Computing Conference, 2010 International*, pages 495 –502, 2010.