Informe Técnico ICC 2011-05-09

DVFS-Technique for Dense Linear Algebra Operations on Multi-Core Processors

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

Mayo 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

DVFS-Technique for Dense Linear Algebra Operations on Multi-Core Processors

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

Abstract:

This paper addresses the efficient explotation of task-level parallelism, present in many dense linear algebra operations, from the point of view of both computational performance and energy consumption. In particular, we consider a procedure, the Slack Reduction Algorithm (SRA), to optimize the execution frequency of a collection of tasks (in which many dense linear algebra algorithms can be decomposed) on multi-core architectures. The results from this procedure are modulated by an energy-aware simulator, which is in charge of scheduling/mapping the execution of these tasks to the cores, leveraging dynamic frequency voltage scaling featured by current technology. Simultaneously, the simulator evaluates the performance benefits of the solution. Experiments with these tools show significant energy gains for two key dense linear algebra operations: the Cholesky and QR factorizations.

Keywords:

Dense linear algebra, scheduling, power consumption, multi-core processors, DVFS.

[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

Técnica DVFS para Algoritmos de Álgebra Lineal Densa en Procesadores Multinúcleo

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

Resumen:

En este trabajo se aborda la explotación eficiente del paralelismo a nivel de tareas, presente en muchas de las operaciones de álgebra lineal densa desde el punto de vista del rendimiento y el consumo de energía. En particular, se presenta el Algoritmo de Reduccin de Holguras, para optimizar la frecuencia de ejecución de un conjunto de tareas (en la que muchos algoritmos de álgebra lineal densa pueden ser descompuestas) en las arquitecturas multinúcleo. Los resultados de este procedimiento son procesados por un simulador de consciente del consumo, encargado de la planificación/asignación a ejecución de estas tareas en los núcleos, aprovechando la escalado dinámico del voltaje y la frecuencia presente en los procesadores actuales. Al mismo tiempo, el simulador evalúa las ventajas de rendimiento del algoritmo presentado. Los experimentos con estas herramientas muestran significativos ahorros de energía para dos importantes operaciones de álgebra lineal densa: la factorización de Cholesky y QR.

Palabras clave:

Álgebra lineal densa, planificación, consumo energético, procesadores multinúcleo, DVFS.

[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`

DVFS-Technique for Dense Linear Algebra Operations 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 This paper addresses the efficient explotation of task-level parallelism, present in many dense linear algebra operations, from the point of view of both computational performance and energy consumption. In particular, we consider a procedure, the Slack Reduction Algorithm (SRA), to optimize the execution frequency of a collection of tasks (in which many dense linear algebra algorithms can be decomposed) on multi-core architectures. The results from this procedure are modulated by an energy-aware simulator, which is in charge of scheduling/mapping the execution of these tasks to the cores, leveraging dynamic frequency voltage scaling featured by current technology. Simultaneously, the simulator evaluates the performance benefits of the solution. Experiments with these tools show significant energy gains for two key dense linear algebra operations: the Cholesky and QR factorizations.

Key words: Dense linear algebra, scheduling, power consumption, multi-core processors, DVFS.

1 Introduction

Large-scale HPC facilities are big consumers of energy, which is employed to operate the computing resources as well as auxiliary systems like backup equipment, air cooling, etc. [1,2,3,4,5]. Power[3] consumption has a direct impact on the operation and maintenance costs of these centers, compromising their existence and impairing the installation of new facilities. But the electricity cost is not the only problem; in general, energy consumption results in carbon dioxide emission, a hazard for the environment and public health, and heat, which reduces reliability of hardware components [3].

The pressure from HPC centers, and especially that of the embedded and mobile market segments, has forced hardware manufacturers to improve the energy efficiency

[3] Note the difference between *energy* and *power*, with the former referring to the power consumed over a period of time (*Energy* = *Power* $\times$ *Time*). For simplicity, in this document we will use both terms indistinctively, though in most cases the meaning is that of energy.

of their designs: CPU, memory and disks usually feature low-power modes to trade-off performance for energy by decreasing frequency and voltage operation (DVFS or *Dynamic Voltage Frequency Scaling* [6,7]). While being a mature subject in other segments, the development of power-aware software for HPC applications, which optimizes both execution time and energy conservation, is still in its beginnings, in spite of the tremendous assets it can yield [8,3].

Recent work has demonstrated the benefits of exploiting task-level parallelism in multi-core processors; see, e.g., [9,10,11], among many others. Following this trend, numerical dense linear algebra libraries like `libflame` [12] and LAPACK [13] are being rewritten to accommodate task-level parallelism for this class of architectures. In these projects, (blocked) dense linear algebra algorithms are statically (PLASMA [14]) or dynamically (`libflame`) decomposed into a collection of tasks (or kernel operations), identifying the dependencies among them. The result is a directed acyclic graph (DAG) with the information implicit in the algorithm, which is then passed to a scheduler in charge of issuing tasks to the computational resources. As a result, tasks are executed in the order dictated by dependencies (data-flow parallelism) instead of the order they appear in the code (control-flow parallelism), which unleashes a richer degree of concurrency. Unfortunately, as of today, these projects address high performance but ignore power consumption.

In this paper we investigate how to leverage DVFS in the execution of dense linear algebra algorithms on state-of-the-art multi-core processors. In particular, we apply CPM to a DAG that represents a collection of tasks and data dependencies among these, corresponding to the computation of a dense linear algebra operation. Our goal is to detect which tasks lie in non-critical paths, in order to adjust the frequency/voltage (DVFS) of the processor cores in charge of their execution and thus save energy. As the execution time depends linearly on the frequency, but the power consumption is a function of the square of the voltage times the frequency, we expect that reducing the operation frequency (and, therefore, the associated voltage) results in significant energy savings while maintaining the execution time. However, we note that reducing the frequency during the execution of a task does not always lead to energy savings because doing so, surely results in a longer execution time and, sometimes, more power usage. Since, in general, this is not the case for dense linear algebra kernels, we will not comment on this issue further.

The main contributions of the paper include:

– We apply the Slack Reduction Algorithm (SRA), which aims at exploiting the *slacks* (idle times) existing in the DAG that represents a dense linear algebra operation by carefully tuning frequency execution of certain tasks.
– In addition, we develop a simulator that validates the theoretical energy gain, calculating a schedule of the tasks which takes into account practical constraints like actual number of resources (processor cores), the cost of varying processor frequency, the discrete range of frequencies, etc.
– We analyze the energy performance of the solution using highly efficient blocked algorithms for the Cholesky and QR factorizations, two important operations required for the solution of (symmetric positive definite) linear systems and linear least squares problems, respectively.

The article is organized as follows. After a brief discussion of related work in the next subsection, Section 2 reviews the Critical Path Method and its application to identify tasks which can be delayed without negatively affecting the total execution time of a project, represented by a collection of dependent tasks. The two main contributions of this work, the application of the Slack Reduction Algorithm (SRA) and the energy-aware simulator, are next introduced in Sections 3 and 4, respectively. Experiments are reported in Section 5, and a few concluding remarks as well as a summary of future work close the paper in Section 6.

1.1 Related work

There exist a number of related investigations. In [15], the authors model a scheduler for clusters that can map tasks and adjust node frequencies, depending on the number of pending jobs. In [16,17] the authors discuss scheduling of independent tasks (jobs) in a DVFS-enabled processor, while in [18,19] this technology is used to schedule tasks with dependencies in a multiprocessor setup. The authors of [20,21,22] introduce several real-time, power-aware schedulers for tasks with dependencies. The work in [23] describes a platform that combines real-time mapping with DVFS to reduce power usage of dependent tasks. The algorithm LPHM in [24] dynamically adjusts the execution time of noncritical tasks using DVFS.

In [25] new heuristics are proposed for an energy-aware task scheduler in an heterogeneous cluster. In [26] a strategy is employed to stretch or reduce the execution time of noncritical jobs. In [27], the authors perform a similar investigation, but frequency is statically tuned at the beginning of the algorithm, and fixed for its complete duration. The authors of [28] also follow the same strategy, with stretch/compress stages, which are iteratively applied until the consumption of power is below a certain threshold. Finally, algorithm LPHEFT (low power HEFT algorithm) is presented in [29] as a means to reduce power consumption, based on scheduling of idle time-slots (or gaps).

2 The Critical Path Method

The *Critical Path Method* (CPM) is commonly used in the field of management and project planning [30] to control the duration of the project by carefully scheduling so-called “critical” tasks (that is, those tasks which are likely to delay the project execution time in case of a late start/finish). We next discuss how to apply CPM to detect slacks in a DAG using a simple case study from linear algebra.

2.1 Demonstration example

We illustrate our goal using the Cholesky factorization of a symmetric positive definite matrix $A \in \mathbb{R}^{n\times n}$, which renders the decomposition $A = LL^T$, where the Cholesky factor $L \in \mathbb{R}^{n\times n}$ is lower triangular. Consider a partitioning of this matrix into blocks of size $b \times b$ and denote the (i, j) block in this partitioning as A_{ij}. (For simplicity, we will assume hereafter that n is an integer multiple of the block size b; i.e., there exists an integer s such that $n = s \cdot b$.) Algorithm 1 presents a blocked (right-looking) procedure to compute the Cholesky factorization of A, overwriting the lower triangular part of the matrix with the contents of the lower triangular factor L. Each operation (i.e., task) in the algorithm is annotated to the right with its theoretical cost in floating-point arithmetic operations, or flops. (Hereafter we neglect lower order terms in the evaluation of theoretical costs.)

Algorithm 1 Blocked (right-looking) Cholesky factorization procedure.

```
1:  for $k = 1, 2, \ldots, s$ do
2:    $A_{kk} = L_{kk} L_{kk}^T$                      CHOLESKY FACTORIZATION       $b^3/3$ flops
3:    for $i = k+1, k+2, \ldots, s$ do
4:      $A_{ik} \leftarrow A_{ik} L_{kk}^{-T}$           TRIANGULAR SYSTEM SOLVE      $b^3$ flops
5:    end for
6:    for $i = k+1, k+2, \ldots, s$ do
7:      for $j = k+1, k+2, \ldots, i-1$ do
8:        $A_{ij} \leftarrow A_{ij} - A_{ik} A_{jk}^T$   MATRIX-MATRIX PRODUCT        $2b^3$ flops
9:      end for
10:     $A_{ii} \leftarrow A_{ii} - A_{ik} A_{ik}^T$     SYMMETRIC RANK-$b$ UPDATE    $b^3$ flops
11:   end for
12: end for
```

Figure 1 shows the dependency graph corresponding to the computation of the Cholesky factorization of a matrix consisting of 3×3 blocks ($s = 3$) using Algorithm 1. In the graph, nodes stand for tasks and edges identify dependencies. Each task is labelled with a first letter representing its type ("P" for Cholesky, "T" for triangular system solve, "G" for general matrix-matrix product, and "S" for symmetric rank-b update); followed by a number that uniquely identifies the task in the graph; and, inside parenthesis, its theoretical computational cost, expressed in *units of time*, or u.t., with the equivalence 1 u.t. = b^3 flops. For simplicity, we assume that there is a direct relation between time and theoretical cost, which is quite realistic for the execution of (Level 3 BLAS-based) dense linear algebra computations on current high performance architectures.

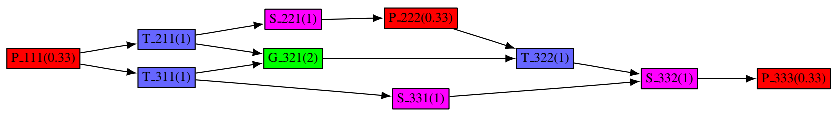

Figure 1. (Task-node) DAG capturing the data dependencies in the computation of the Cholesky factorization of a matrix consisting of 3×3 blocks using Algorithm 1.

We define the (total) *slack* as the amount of time that a task can be delayed without increasing the total execution time of the algorithm. Note that, although in the following we often refer to time, in our approach the slack will be computed in terms of theoretical cost, because of the equivalence that we introduced between u.t. and flops. In order to apply CPM to calculate the slack of the algorithm tasks, we first transform the *task-node* graph into a *task-edge* graph, where tasks are represented by edges instead of nodes. The DAG that results from this transformation is shown in Figure 2. There, new edges/dummy tasks, labelled as `NULL`, are introduced to preserve the dependencies from the original *task-node* graph.

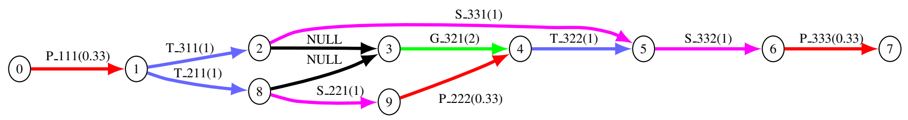

Figure 2. Task-edge DAG capturing the data dependencies in the computation of the Cholesky factorization of a matrix consisting of 3×3 blocks using Algorithm 1.

CPM can now be applied to the task-edge graph in order to detect slacks, which yields the profile in Table 1. The column labelled as *Task* lists the task name. Column $i - j$ specifies the initial and final nodes of the task, i and j respectively; and column $C_{i,j}$ its cost (in u.t.). The information computed by the CPM is contained in the next three columns. For each task $i - j$, the first column, labelled as ES_i, represents the *earliest time* at which the tasks leaving from node i can start their execution. This value can be computed as:

$$ES_i = \max_k (ES_k + C_{ki}) \tag{1}$$

for all node k connected to node i by an edge from k to i. The earliest execution time for node 0 is, obviously, 0; while, e.g., ES_5 is the maximum between $ES_4 + C_{4,5}$ and $ES_2 + C_{2,5}$. The next column, LF_j, indicates for each task $i - j$ the *latest time* at which the tasks that reach node j can finish execution without increasing the total time of the algorithm. This is given by:

$$LF_j = \min_k (LF_k - C_{jk}) \tag{2}$$

for all node k connected to node j by an edge from j to k. In this case, e.g., LF_1 is the minimum between $LF_2 - C_{1,2}$ and $LF_8 - C_{1,8}$, and LF_7 equals the length of the critical path, this is, 5.67 u.t. Finally, the last column, $S_{i,j}$, indicates the *slack* of the corresponding task, and can be obtained from:

$$S_{i,j} = LF_j - ES_i - C_{i,j}. \tag{3}$$

CPM identifies slack times, but does not provide an explicit strategy (procedure) to exploit them. In the following section, we introduce an algorithm that conveniently tailors execution frequency, to tune the slack of those tasks with $S_{i,j} > 0$, yielding a lower power usage. In an ideal case where the cores can operate at an infinite (continuous) range of frequencies, and the transition time (overhead) between any two frequencies is zero, the slack could be adjusted very accurately. In a real case, processor cores run at a limited (discrete) number of frequencies, and changing the between any two given frequencies is not immediate, so that the slack can only be adjusted sub-optimally.

Task	$i-j$	$C_{i,j}$	ES_i	LF_j	$S_{i,j}$
P_111	0-1	0.33	0	0.33	0
T_211	1-8	1	0.33	1.33	0
T_311	1-2	1	0.33	1.33	0
NULL	2-3	0	1.33	1.33	0
S_221	8-9	1	1.33	3	0.67
G_321	3-4	2	1.33	3.33	0
S_331	2-5	1	1.33	4.33	2
P_222	9-4	0.33	2.33	3.33	0.67
T_322	4-5	1	3.33	4.33	0
S_332	5-6	1	4.33	5.33	0
P_333	6-7	0.33	5.33	5.67	0
NULL	8-3	0	1.33	1.33	0

Table 1. Application of CPM to the task-edge DAG capturing the data dependencies in the computation of the Cholesky factorization of a matrix consisting of 3×3 blocks using Algorithm 1.

3 Slack Reduction Algorithm

We next describe the *Slack Reduction Algorithm* (SRA) to set and assign a tentative operation frequency to each task, among a discrete number of these, at which it will be executed. The algorithm is preceded by an initialization stage that decomposes a given dense linear algebra algorithm into a collection of tasks, and identifies the dependencies among these, resulting in a task-node DAG (alike that in Figure 1). In this preliminary stage, this information is then transformed into a task-edge DAG (see, e.g., Figure 2). Recent work on efficient execution of dense linear algebra operations on multi-core processors has shown that automatic extraction of this information can be easily done [31]. Transformation between task-node and task-edge graphs is a simple and systematic process for which efficient algorithms exist.

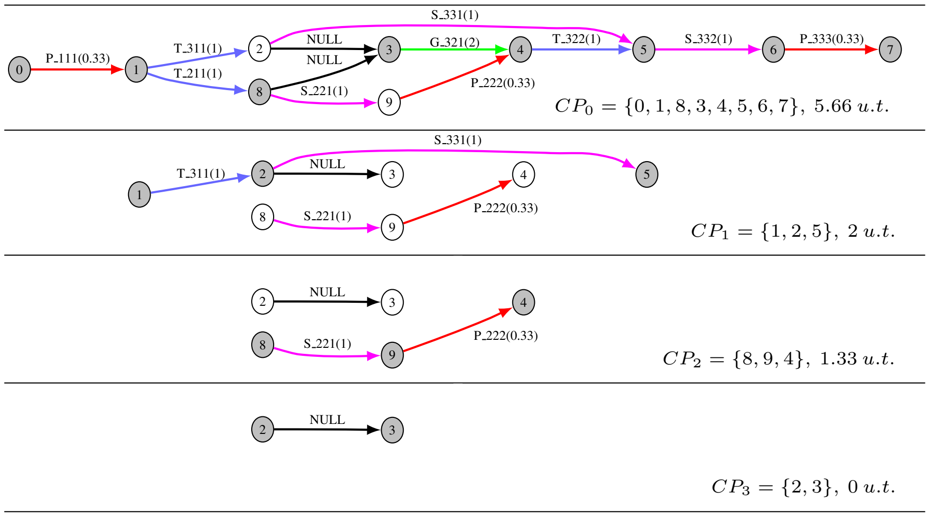
P_111(0.33)
T_311(1)
T_211(1)
NULL
NULL
S_221(1)
S_331(1)
G_321(2)
P_222(0.33)
T_322(1)
S_332(1)
P_333(0.33)
$CP_0 = \{0,1,8,3,4,5,6,7\},\ 5.66\ u.t.$
$CP_1 = \{1,2,5\},\ 2\ u.t.$
$CP_2 = \{8,9,4\},\ 1.33\ u.t.$
$CP_3 = \{2,3\},\ 0\ u.t.$

Figure 3. Critical subpath decomposition using DAG that captures the data dependencies in the Cholesky factorization of a matrix consisting of 3×3 blocks using Algorithm 1.

The SRA is composed of three stages, with the second and third stages being iterative procedures. To illustrate the algorithm, in the following discussion we will refer to the DAG in Figure 2. We will also consider the discrete collection of frequencies $FR = \{2.27, 2.13, 2.00, 1.87, 1.73, 1.60\}$, (in GHz, conformal with current values, e.g., in Intel Xeon E5520).

Frequency assignment: Initially, all tasks except those marked as `NULL` are assigned to be run at the maximum frequency.
Critical subpath extraction: In this step the graph is decomposed into a number of critical subpaths. First, the critical path is identified. Next, the graph edges that belong to the critical path (as well as the nodes, if they have no other edge arising at or leaving from them) are eliminated from the graph. A new critical subpath is then extracted for this subgraph; and the process is repeated until the graph is empty. Figure 3 details the application of the procedure to the DAG contained in Figure 2. For each iteration of the extraction procedure, we indicate the sequence of nodes in the critical path/subpaths ($CP_0/CP_1, CP_2, CP_3$) and the execution time in the bottom right corner. Note that this decomposition automatically sorts critical subpaths according to descending execution time.
SRA: This is the key stage which calculates the (recommended) operation frequency and, therefore, the execution time of tasks. The algorithm starts by processing critical subpath $CP_1 = \{1, 2, 5\}$, trying to adjust the execution time of the tasks embedded in this subpath with a nonzero slack: this includes only `S_331`. The execution rate of the remaining tasks will not be modified during this iteration. The slack for `S_331` is 2 u.t. which, under ideal conditions, would allow us to run this task as $f_{\max}/2$; however, given the values in FR, the best one can do is to assign $f = 1.60$ to this task, which increases its execution time from 1 to 1.42 u.t.

The process continues next with $CP_2 = \{8, 9, 4\}$. Both `S_221` and `P_222` lie in this subpath and have a slack of 0.67 u.t. which matches that of the critical subpath. In other words, the slack for this subpath, 0.67 u.t., can be split between the two tasks or assigned to only one of them. Given the discrete ranges of frequencies available, we may decide to run both tasks at f =1.73, so that the execution times of `S_221` and `P_222` becomes, respectively, 1.31 and 0.44 u.t.

As a result tasks `S_331`, `S_221` and `P_222` are run at frequencies 1.60, 1.73 and 1.73, respectively, and the total slack reflected in Table 1 (taking into account the slacks that are associated with the same subpath) is reduced from 2.0+0.67=2.67 to 2.0-(1.42-1.0)+(0.67-(1.31-1)-(0.44-0.33))=1.83 u.t.; see Figure 4. Note again that a decrease in the slack value is achieved by decreasing the operation frequency and, therefore, yields the sought-after energy savings.

The third critical subpath, $CP_3 = \{2, 3\}$ only contains a dummy task, and therefore, requires no processing.

Under mild conditions, a user can be interested in trading-off execution time for energy consumption. To accommodate this, the SRA utilizes a user-defined *excess ratio* which specifies the maximum increase in execution time that is acceptable. This makes the previous algorithm slightly more complicated, but the basic structure remains the same. For simplicity, we skip the exposition of the modified algorithm, and refer to the experimental evaluation for a practical demonstration of its effects.

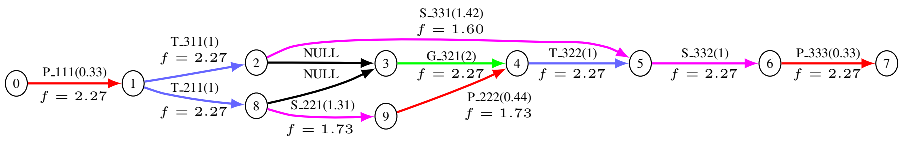

Figure 4. Frequency assignment and cost of the task-edge DAG capturing the data dependencies in the computation of the Cholesky factorization of a matrix consisting of 3 × 3 blocks using Algorithm 1.

4 Simulator

In order to evaluate the performance of our strategy, we have developed a flexible, energy-aware simulator which uses the information obtained with the SRA to produce a schedule (in graphical form), for a particular target architecture. It also records all frequency variations occurred during the execution, and displays statistics on energy saving in terms of percentage of time that each computing resource has operated at a given frequency. Therefore, this tool can help to analyze the theoretical performance and energy savings produced by the application of the SRA on different scenarios: DAGs associated with different task-based algorithms, platform setups, excess ratios, frequency ranges etc. In general, the static (*a priori*) schedule produced by the simulator is not applicable in practice as even tiny variations during task execution may render it useless. However, it serves as a demonstrator of the benefits that a technique like DVFS can yield for the execution of dense linear algebra kernels.

In the following, we describe the possibilities and properties of the simulator in more detail.

4.1 Input parameters

The input parameters for the simulator include:

– A DAG capturing tasks and dependencies implicit in the blocked algorithm as well as the frequencies recommended by the SRA to execute tasks (corresponding to the output of this procedure).
– A simple description of the target architecture that specifies the *number of sockets* (or physical processors) and the *number of cores per socket*. To mimic current technology from Intel (*Enhanced Intel®SpeedStep Technology®*), the simulator can only change the frequency at socket level, but not for individual cores. Furthermore, the socket frequency cannot be changed if there is a task running on it at the moment.
– A discrete range of processor frequencies, FR, and associated voltages.
– The cost (overhead) required to perform frequency changes.

4.2 Scheduler

As starting point, we have chosen a static priority list scheduler [32,33]. The reason for this is twofold. First, the approximate duration of the tasks (that is, their theoretical cost) is known in advance (as, in general, is the case for dense linear algebra). Second, the execution of tasks that lie on the critical path must be prioritized. In particular, tasks with higher recommended execution frequency are given raised priority. Among the tasks which have to be run at the same frequency, those which belong to a critical subpath (CP_i) with smaller index (i) are sorted first.

Consider the execution of a task T_i with recommended frequency f_i (by the SRA). The scheduling algorithm maps execution of T_i to the first idle core that satisfies one of the following conditions, checking them in the order they appear next:

1. The core socket is operating at frequency f_i.
2. The core socket is varying its operation frequency to $f_o = f_i$. (The task will commence execution when the change is completed.)
3. The core socket is operating at a frequency $f_o > f_i$.
4. The core socket is varying its operation frequency to $f_o > f_i$.
5. All cores in the same socket are idle. If the socket is operating at a frequency $f_o \neq f_i$, a change of frequency to f_i is requested. The socket is reserved so that T_i will be the first task that will run on it when the change is completed.

If none of the above conditions is satisfied, the task remains in the pending queue, waiting for variations in system. This strategy will ensure that the execution time of a task does not require longer than recommended by the SRA. For that purpose, the task is run in a core running at the desired frequency or, if not possible, a higher one.

5 Results

In this section, we evaluate the performance of the SRA combined with the energy-aware scheduler using state-of-the-art blocked algorithms for two important dense linear algebra operations. In the experiments we consider two scenarios: one with strict execution time (excess ratio=1.0) and an alternative one where the user can accept up to a 50% increase in the time-to-solution (excess ratio=1.5) for more aggressive power reduction.

5.1 Benchmark algorithms

In our experiments, we analyze the traditional right-looking blocked algorithmic variant for the Cholesky factorization (see Algorithm 1) and the incremental QR factorization introduced in [34]. These two algorithms represent the state-of-the-art to attain high performance on current multi-core processors [31]. Each of these operations is evaluated for a variety of (square) matrix dimensions (n) and block sizes (b). Specifically, in our experiments n varies from 576 to 2,112, with $b = 192$. (Although we carried out the evaluation for other problems sizes, we do not report the results here. The chosen block size is known to be close to optimal for most kernels from many current implementations of BLAS.) For simplicity, we assume that the execution time of the tasks as well as that of the whole algorithm are proportional to their theoretical cost. The usual flop counts of $n^3/3$ and $4n^3/3$ are considered, respectively, for the Cholesky and

QR factorizations (though the incremental QR factorization performs a slightly higher number of flops).

The scheme of the incremental QR factorization of a matrix A consisting of $s \times s$ blocks, of dimension $b \times b$ each is shown in Algorithm 2.

Algorithm 2 Algorithm-by-blocks for the incremental QR factorization.

```
1:  for $k = 1, 2, \ldots, s$ do
2:    $A_{kk} = Q_{kk} R_{kk}^T$                                   QR FACTORIZATION             $4b^3/3$ flops
3:    for $j = k+1, k+2, \ldots, s$ do
4:      $A_{kj} \leftarrow Q_{kk}^T A_{kj}$                         APPLY TRANSFORMS             $2b^3$ flops
5:    end for
6:    for $i = k+1, k+2, \ldots, s$ do
7:      $\begin{pmatrix} A_{kk} \\ A_{ik} \end{pmatrix} = \begin{pmatrix} Q_{kk} \\ Q_{ik} \end{pmatrix} R_{ik}$      $2 \times 1$ QR FACTORIZATION      $2b^3$ flops
8:      for $j = k+1, k+2, \ldots, s$ do
9:        $\begin{pmatrix} A_{kj} \\ A_{ij} \end{pmatrix} \leftarrow \begin{pmatrix} Q_{kk} & 0 \\ Q_{ik} & I \end{pmatrix}^T \begin{pmatrix} A_{kj} \\ A_{ij} \end{pmatrix}$      $2 \times 1$ APPLY TRANSFORMS      $4b^3$ flops
10:       end for
11:     end for
12: end for
```

Figure 5 illustrates the tasks and dependencies obtained for the QR factorization of a blocked 3×3 matrix using Algorithm 2. There, "Q" denotes the QR factorization, "O" the application of (orthogonal) transforms, "Q2" the 2×1 QR factorization, and "O2" the 2×1 application of (orthogonal) transforms.

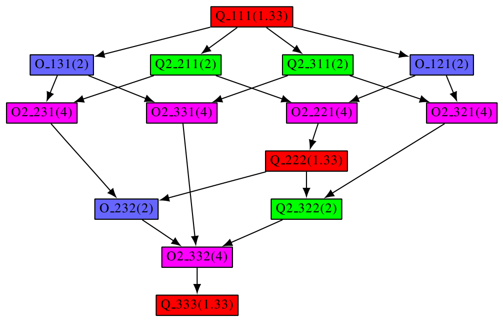

Figure 5. (Task-node) DAG capturing the data dependencies in the computation of the QR factorization of a matrix consisting of 3×3 blocks using Algorithm 2.

5.2 Environment setup

The following architectural considerations are assumed:

– **Frequency change latency:** Varying the frequency of a socket between any two values incurs an overhead of 0.1 u.t.

– **Target architecture:** The simulated platform consists of four quad-core sockets (a total of 16 cores) with a discrete range of frequencies $FR = \{2.27, 2.13, 2.00, 1.87, 1.73, 1.60\}$ GHz. Associated voltages vary from 0.75 to 1.35 V, with a linear relation between these and the frequency. These values are representative featured, e.g., by Intel Xeon E5520 processors [35].

5.3 Metrics

In order to assess the benefits of the proposed solution we will employ the following metrics:

Execution time stands for the estimated total execution time (in u.t.). This parameter is in general higher than the execution time (length) of the critical path as, in a real setup, the number of computational resources (cores) is bounded and, therefore, the execution time of the algorithm will obviously increase.

Impact of the SRA on time (T_{SRA}) measures the ratio between the algorithm execution times using the frequencies determined by the SRA and the maximum frequency. Ideally, this ratio should be 1.0.

Consumption is the energy usage, in consumption units (u.c.), for the execution of the algorithm. We assume that power is proportional to the square of the voltage times the frequency. We estimate the usage of energy as a function of the aggregated time during which the processor cores have been operating at the different frequencies.

Impact of the SRA on consumption (C_{SRA}) measures the ratio between the algorithm power consumption using the frequencies recommended by the SRA and the maximum frequency. Ideally, this ratio should be close to 0.

Results for the Cholesky factorization Figure 6 reports the results for the Cholesky factorization. Let us comment on the results with excess ratio=1.0 first (red and blue bars respectively for energy and time, in %). The plot shows that, for small matrices, the energy required to execute the algorithm at the frequencies dictated by the SRA is only 25% of that required by an execution at the original (maximum) frequency. The plot also shows that this ratio raises up to 70% for the largest problem sizes. The reason underlying this growth is that, due to limitations of the algorithm to adjust the slack, and the limited number of frequencies, the length of alternative subpaths does not always conform perfectly to the reference critical subpaths. Nevertheless, this is all attained with an increase in the execution time that is below 10% in all cases. The results when the excess ratio is set to 1.5 (green and magenta bars for energy and time, respectively) clearly show the trade-off between execution time and energy: The lowest energy savings are around 60%, for the largest problem sizes, but they come at the expense of an increase in the execution time of up to 20% in one of the cases.

A closer inspection of the dependency tree reveals the strong impact of the degree of parallelism of the algorithm. In particular, when the number of task that can be run in parallel is large compared with the number of cores, savings become practically negligible. On the other hand, in a situation where the number of tasks that can be run in parallel is limited, energy savings will grow, with basically no (negative) effect on the total execution time.

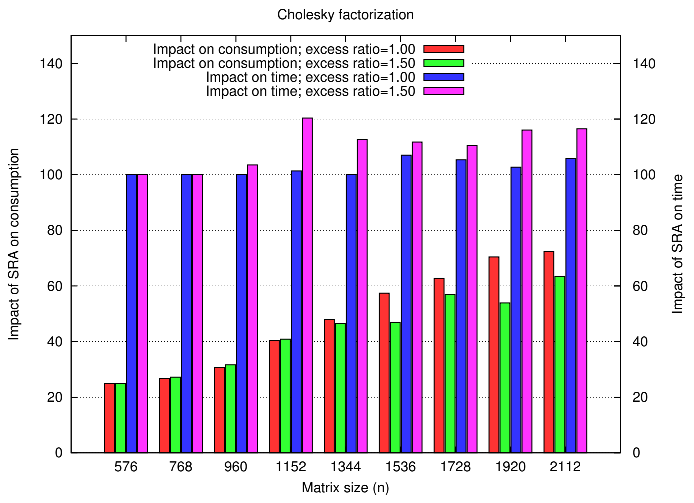

Figure 6. Impact of the SRA on energy and time for the Cholesky factorization.

Results for the QR factorization Figure 7 shows a similar behaviour for the QR factorization: when an excess in the execution times is not desirable, the energy required by the SRA requires between 25% (smallest problem dimension) and 76% of that consumed by the original frequency (largest problem dimension), and the time grows at most by 10%. The excess ratio allows to increase energy savings, but at the expense of longer execution time.

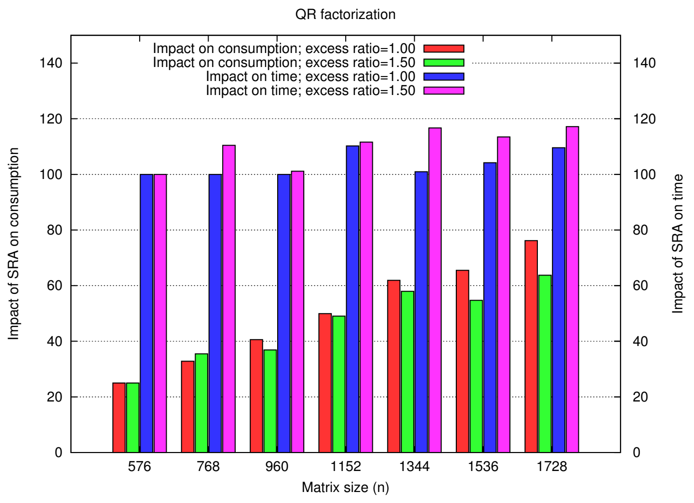

Figure 7. Impact of the SRA on energy and time for the QR factorization.

6 Conclusions and Future Work

In this paper, we provide evidence that, in theory, it is possible to improve energy consumption in the execution of dense linear algebra algorithms while delivering high performance. Following the current trend for multi-core parallelization (adopted, e.g., in libraries `libflame` and PLASMA), our algorithms exploit task-level parallelism, considering that dense linear algebra operations are partitioned into a number of tasks with dependencies among them. Our efforts towards power conservation start from the DAG representing the operation and are based on two key observations. First, if all the tasks run at full speed, idle times appear during their execution. Second, present processors include efficient mechanisms to dynamically adjust frequency/voltage (DVFS) and hence the energy consumed.

Our approach to the problem is inspired by concepts and methods of project planning theory. Specifically, we first apply CPM to determine the total slack of tasks, and then employ the SRA to conveniently adjust the frequency at which tasks must run. The results from this process are then fed to a simulator, which is used to further tune the frequencies of the tasks to a particular target architecture and assess the theoretical benefits that can be obtained for a given operation.

We have evaluated these tools using two representative algorithms of dense linear algebra, namely, the Cholesky and QR factorizations. The results of this experimental analysis under realistic conditions show a significant reduction in power consumption in both cases and some interesting insights. In particular, we observed that a higher ratio between the number of computational resources and number of tasks yields a more reduced power consumption. Second, even a small stretch of the total time may result in significant energy savings. Although not reported, the results for the LU factorization (with incremental pivoting), the third key algorithm in the numerical solution of linear systems, show similar behaviour to that of the incremental QR factorization.

We plan to address several open questions as part of future work. First, scheduling heterogeneous tasks (with dependencies among them) in environments with limited number of resources is known to be an NP problem; therefore, efficient new heuristics, tuned for the particular conditions of our problem, can have a considerable impact on the results. Second, our frequency variation strategy is static, deciding the task frequencies in advance; this should be changed into a dynamic policy, which operates at run-time, dynamically adapting to variations on the conditions. Third, our ultimate goal is to integrate the results from this research with a practical run-time scheduler for dense linear algebra operations.

Acknowledgments

The authors from the Universidad Jaume I were supported by project CICYT TIN2008-06570-C04-01 and FEDER.

References

1. J. Dongarra, P. Beckman, T. Moore, P. Aerts, G. Aloisio, J.C. Andre, D. Barkai, J.Y. Berthou, T. Boku, B. Braunschweig, et al. The international exascale software project roadmap. *International Journal of High Performance Computing Applications*, 25(1):3, 2011.
2. M. Duranton et al. The HiPEAC vision, 2010. Available from `http://www.hipeac.net/roadmap`.
3. Wu-chun Feng, Xizhou Feng, and Rong Ce. Green supercomputing comes of age. *IT Professional*, 10(1):17 –23, jan.-feb. 2008.
4. Ralf Gruber and Vincent Keller. One Joule per GFlop for BLAS2 Now! In Simos Theodore E., Psihoyios George, and Tsitouras Ch, editors, *AIP Conf. Proceedings*, volume 1281, pages 1321–1324. American Institute of Physics, 2010.
5. Thomas Ludwig. Editorial for the First International Conference on Energy-Aware High Performance Computing. *Computer Science - Research and Development*, 25(3):123–124, 2010.
6. C. Hsu and W. Feng. A feasibility analysis of power awareness in commodity-based high-performance clusters. *Cluster 2005*, 2005.
7. C. Hsu and W. Feng. A power-aware run-time system for high-performance computing. In *Proceedings of the 2005 ACM/IEEE conference on Supercomputing*, page 1. IEEE Computer Society, 2005.
8. Susanne Albers. Energy-efficient algorithms. *Commun. ACM*, 53:86–96, May 2010.
9. Cilk project home page. `http://supertech.csail.mit.edu/cilk/`.
10. SMP superscalar project home page. `http://www.bsc.es/plantillaG.php?cat_id=385`.
11. StarPU project home page. `http://runtime.bordeaux.inria.fr/StarPU/`.
12. Field G. Van Zee. `libflame`*: The Complete Reference*. `www.lulu.com`, 2009.
13. E. Anderson, Z. Bai, C. Bischof, L. S. Blackford, J. Demmel, Jack J. Dongarra, J. Du Croz, S. Hammarling, A. Greenbaum, A. McKenney, and D. Sorensen. *LAPACK Users' guide*. SIAM, 3rd edition, 1999.
14. PLASMA project home page. `http://icl.cs.utk.edu/plasma/`.
15. Maja Etinski, Julita Corbalan, Jesus Labarta, and Mateo Valero. Utilization driven power-aware parallel job scheduling. *Computer Science - Research and Development*, 25:207–216, 2010.
16. F. Yao, A. Demers, and S. Shenker. A scheduling model for reduced cpu energy. In *Proceedings of the 36th Annual Symposium on Foundations of Computer Science*, FOCS '95, pages 374–, Washington, DC, USA, 1995. IEEE Computer Society.
17. A. Manzak and C. Chakrabarti. Variable voltage task scheduling for minimizing energy or minimizing power. In *Proceedings of the Acoustics, Speech, and Signal Processing, 2000. on IEEE International Conference - Volume 06*, pages 3239–3242, Washington, DC, USA, 2000. IEEE Computer Society.
18. Gu yeon Wei, Jaeha Kim, Dean Liu, Stefanos Sidiropoulos, and Mark A. Horowitz. A variable-frequency parallel I/O interface with adaptive power-supply regulation. *IEEE J. Solid-State Circuits*, 35:1600–1610, 2000.
19. Flavius Gruian and Krzysztof Kuchcinski. Lenes: task scheduling for low-energy systems using variable supply voltage processors. In *Proceedings of the 2001 Asia and South Pacific Design Automation Conference*, ASP-DAC '01, pages 449–455, New York, NY, USA, 2001. ACM.
20. Steven M. Martin, Krisztian Flautner, Trevor Mudge, and David Blaauw. Combined dynamic voltage scaling and adaptive body biasing for lower power microprocessors under dynamic workloads. In *Proceedings of the 2002 IEEE/ACM international conference on Computer-aided design*, ICCAD '02, pages 721–725, New York, NY, USA, 2002. ACM.

21. Jiong Luo and Niraj K. Jha. Power-efficient scheduling for heterogeneous distributed real-time embedded systems. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 26(6):1161–1170, 2007.
22. Jiong Luo, Li-Shiuan Peh, and Niraj Jha. Simultaneous dynamic voltage scaling of processors and communication links in real-time distributed embedded systems. In *Proceedings of the conference on Design, Automation and Test in Europe - Volume 1*, DATE '03, pages 11150–, Washington, DC, USA, 2003. IEEE Computer Society.
23. Yumin Zhang, Xiaobo Sharon Hu, and Danny Z. Chen. Task scheduling and voltage selection for energy minimization. In *Proceedings of the 39th annual Design Automation Conference*, DAC '02, pages 183–188, New York, NY, USA, 2002. ACM.
24. Yves Robert, Manish Parashar, Ramamurthy Badrinath, and Viktor K. Prasanna, editors. *High Performance Computing - HiPC 2006, 13th International Conference, Bangalore, India, December 18-21, 2006, Proceedings*, volume 4297 of *Lecture Notes in Computer Science*. Springer, 2006.
25. Y.C. Lee and A.Y. Zomaya. Minimizing energy consumption for precedence-constrained applications using dynamic voltage scaling. In *Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid-Volume 00*, pages 92–99. IEEE Computer Society, 2009.
26. H. Kimura, M. Sato, Y. Hotta, T. Boku, and D. Takahashi. Empirical study on reducing energy of parallel programs using slack reclamation by DVFS in a power-scalable high performance cluster. In *Cluster Computing, 2006 IEEE International Conference on*, pages 1–10. IEEE, 2007.
27. V. Shekar and B. Izadi. Energy aware scheduling for DAG structured applications on heterogeneous and DVS enabled processors. In *International Conference on Green Computing*, pages 495–502. IEEE, 2010.
28. 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.
29. K. Palli. Scheduling DAGs for Minimum Finish Time and Power Consumption on Heterogeneous Processors. 2005.
30. L.R. Shaffer, JB Ritter, and WL Meyer. *The critical-path method*. McGraw-Hill, 1965.
31. Gregorio Quintana-Ortí, Enrique S. Quintana-Ortí, Robert A. van de Geijn, Field G. Van Zee, and Ernie Chan. Programming matrix algorithms-by-blocks for thread-level parallelism. *ACM Transactions on Mathematical Software*, 36(3):14:1–14:26, 2009.
32. R. Li and H.C. Huang. List scheduling for jobs with arbitrary release times and similar lengths. *Journal of scheduling*, 10(6):365–373, 2007.
33. A. Mtibaa, B. Ouni, and M. Abid. An efficient list scheduling algorithm for time placement problem. *Computers & Electrical Engineering*, 33(4):285–298, 2007.
34. Brian C. Gunter and Robert A. van de Geijn. Parallel out-of-core computation and updating the QR factorization. *ACM Transactions on Mathematical Software*, 31(1):60–78, 2005.
35. V. Pallipadi. Enhanced Intel speedstep technology and demand-based switching on Linux. *Intel Developer Service*.