Energy-aware strategies for task-parallel sparse linear system solvers

José I. Aliaga | María Barreda | Asunción Castaño

1 | INTRODUCTION

Large sparse systems of linear equations are regularly found in diverse scientific and engineering applications, such as the numerical solution of partial differential equations and big-data analytics. The interest of these applications and the fact that the solution of linear systems is usually a significant time-consuming stage has motivated, over the past decades, the development of highly tuned algorithms and libraries to efficiently tackle sparse instances of these linear algebra problems in general-purpose processors, following the evolution of computer architectures.

High Performance Computing (HPC) architectures enable the solution of complex applications by aggregating a number of multicore processors. As a consequence, developers face the challenge of implementing parallel algorithms that efficiently exploit the concurrency of the hardware. However, the advances in the number of transistors that can be integrated in a circuit have not enjoyed a proportional reduction of the power dissipated by the CMOS technology, turning the power wall into a crucial challenge that the HPC community needs to address. Therefore, if we want to render Exascale systems feasible by 2020, a holistic power/energy-aware approach is needed. Unfortunately, in an era where power has become the key factor that constrains both the design and performance of current computer architectures, few software developers take it into account in their implementations.

ILUPACK (Incomplete LU decomposition PACKage) offers an assorted variety of Krylov subspace-based methods, enhanced with a sophisticated ILU-type preconditioner, for the iterative solution of sparse linear systems. The cost of computing and applying ILUPACK’s preconditioner has sparked several recent efforts to develop parallel versions of this solver, for multicore processors, graphics accelerators, and clusters of computer nodes.

Task-parallel versions of ILUPACK have also been used as a case study to explore the energy consumption and optimization of iterative solvers. Concretely, in the work of Aliaga et al., the impact of C-States and P-States (idle states and operational states defined in the Advanced Configuration and Power Interface [ACPI] Specification) on ILUPACK is analyzed on two distinct platforms with multicore technology from AMD and Intel. On both architectures, the race-to-halt/race-to-idle strategy, on which the C-states are exploited, was the key to develop energy-aware implementations of ILUPACK. Moreover, the authors conclude that the use of static Voltage Frequency Scaling (VFS) is only useful in one of these platforms. More recently, in our other work, a preliminary algorithmic-based energy-saving technique is introduced to improve the energy efficiency, in which the P-states of the cores are dynamically changed for some preconditioned Conjugate Gradient (PCG) operations. In this paper, we extend that work,
exploring different new approaches (using both P-states and C-states) to save energy in the task-parallel version of ILUPACK’s PCG on an Intel Xeon Haswell-EP processor. In order to yield an energy-efficient execution, we have analyzed the following strategies.

- We explore the benefits of the race-to-halt/race-to-idle (RTH) strategy, which reduces the energy-consumption of ILUPACK transforming the busy-wait behavior of idle threads into an idle-wait.
- In addition, we leverage the iterative nature of the method to progressively adjust the frequency of the processor cores in order to reduce idle periods and harvest energy, implementing a dynamic VFS (DVFS)-based approach.11 In rough detail, this strategy detects the threads, which experience longer idle periods, and adjusts the best P-state for the execution of each task in order to reduce the waiting time of these threads. The DVFS technique has been successfully applied in the works of Wang et al,12 Alonso et al,13 and Kolodziej et al.14
- We also implement a memory-bound aware (MBA) methodology that adjusts the P-state of all threads taking into account the energy consumption instead of the execution time. This technique is useful to adapt the frequency of the cores to the current arithmetic intensity of the executed codes, reducing the energy consumption. We will show that there are several alternatives to achieve the objective of minimizing energy consumption.

All these techniques have been combined to analyze their impact on the implementation of the OpenMP task-parallel version of ILUPACK, showing relevant energy gains on the Haswell-EP processor, particularly when applying the MBA technique.

The rest of this paper is structured as follows. In Section 2, we introduce the multilevel preconditioned iterative solver in ILUPACK, and its task-parallel variant. In Section 3, we explain the different energy-aware strategies implemented to improve the energy efficiency in the OpenMP task-parallel version of ILUPACK. We evaluate the impact on the application of these strategies in Section 4. Finally, we close this paper with a few concluding remarks in Section 5.

2 OVERVIEW OF ILUPACK

2.1 Sequential ILUPACK

Consider the linear system \( Ax = b \), where \( A \in \mathbb{R}^{n \times n} \) is a sparse coefficient matrix, \( b \in \mathbb{R}^n \) is the right-hand side vector, and \( x \in \mathbb{R}^n \) is the sought-after solution. The solution of these kinds of systems can be performed using ILUPACK, a software library, written in C and Fortran, for the iterative solution of large sparse linear systems. For symmetric positive definite (SPD) linear systems, the PCG method implementation in ILUPACK integrates an "inverse-based approach" into the ILU factorization of matrix \( A \), in order to obtain an efficient preconditioner (\( A \approx LL^T = M \)). This method combines dropping with pivoting to bound the norm of the inverse triangular factor \( L \), obtaining a numerical multilevel hierarchy of partial inverse-based approximations.15

The algorithmic description of the PCG method implemented in ILUPACK is presented in Figure 1. The first step of the solver is the computation of the preconditioner \( PrCo \). The following iterative loop comprises a matrix-vector product (SpMV), the preconditioner application (PrAp), and several vector operations (Dot products, Axpy operations, 2-norm). The computation and application of the preconditioner centralize most of the computational cost of the solver. For this reason, in the remainder of this section, we mainly focus on these operations.

Computation of the preconditioner. ILUPACK relies on the commonly named inverse-based approach to compute the preconditioner. This approach enhances the robustness of classical incomplete LDL\(^T\) factorizations by restricting the growth of the entries in the inverses of the triangular factors. To verify this, consider the factorization

\[
A = LDL^T + R \rightarrow L^{-1}AL^{-T} = D + L^{-1}RL^{-T},
\]

where \( L \) is unit lower triangular matrix, \( D \) is diagonal, and \( R \) is the error matrix that accumulates those entries dropped during the factorization. The preconditioned matrix is obtained by applying the preconditioner \( M = LDL^T \) on the original matrix.

Moreover, ILUPACK accommodates pivoting during the factorization to bound the norm of the inverse triangular factors, creating a multilevel hierarchy of partial inverse-based approximations (see Figure 2). When the multilevel method is employed over multiple levels, a cascade of factors
is usually acquired. Besides, the computed multilevel factorization is adapted to the structure of the subjacent system. Consequently, in this case, the multilevel preconditioner can be formulated recursively, at a given level \( i \), as

\[
M_i \approx \tilde{D}^{-1} \tilde{P} \left[ \begin{array}{ccc} L_B & 0 & 0 \\ L_F & I & 0 \\ 0 & 0 & M_{i+1} \end{array} \right] \left[ \begin{array}{ccc} L^T & L^T & 0 \\ 0 & 0 & I \end{array} \right] \tilde{P}^T \tilde{D}^{-1}. \tag{2}
\]

where \( P \) defines the inverse-based permutation, \( L_B, L_F, \) and \( D_B \) are blocks of the factors of the multilevel \( LDL^T \) preconditioner (with \( L_B \) unit lower triangular and \( D_B \) diagonal), whereas \( \tilde{P} \) and \( \tilde{D} \) are, respectively, related to filling reduction and scaling matrices; and \( M_{i+1} \) is the preconditioner computed at level \( i + 1 \).

**Application of the preconditioner.** The application of the preconditioner in level \( l \) (ie, computing \( z := M_l^{-1} r \)) requires solving a system of linear equations

\[
\left[ \begin{array}{ccc} L_e & 0 & 0 \\ L_e & I & 0 \\ 0 & 0 & M_{i+1} \end{array} \right] \left[ \begin{array}{ccc} L^T & L^T & 0 \\ 0 & 0 & I \end{array} \right] \tilde{P}^T \tilde{D}^{-1} z = \tilde{P}^T \tilde{D}^{-1} r. \tag{3}
\]

After applying several transformations to the initial system \( \hat{r} := Dr \) and \( \hat{r} := P^T \hat{P} \hat{r} \), we obtain a new system \( 16 \)

\[
\left[ \begin{array}{ccc} L_B & 0 & 0 \\ L_e & I & 0 \\ 0 & 0 & M_{i+1} \end{array} \right] \left[ \begin{array}{ccc} L^T & L^T & 0 \\ 0 & 0 & I \end{array} \right] w = r.
\]

which is solved for \( w(= P^T \hat{P} \hat{D}^{-1} z) \) in the following three steps.

1. **Lower triangular solver (\( \text{LwTSv} \)):**

   \[
   \left[ \begin{array}{ccc} L_B & 0 & 0 \\ L_e & I & 0 \\ 0 & 0 & M_{i+1} \end{array} \right] \left[ \begin{array}{ccc} \gamma_B & \gamma_C \\ \gamma_C & \gamma_C \\ \gamma_C & \gamma_C \end{array} \right] = \left[ \begin{array}{ccc} \hat{r}_B \\ \hat{r}_C \end{array} \right] \Rightarrow L_B \gamma_B = \hat{r}_B; \gamma_C := \hat{r}_C - L_e \gamma_B.
\]

2. **Recursive step:**

   \[
   \left[ \begin{array}{ccc} D_B & 0 & 0 \\ 0 & 0 & M_{i+1} \end{array} \right] \left[ \begin{array}{ccc} x_B \\ x_C \\ x_C \end{array} \right] = \left[ \begin{array}{ccc} \gamma_B \\ \gamma_C \\ \gamma_C \end{array} \right] \Rightarrow x_B := D_B^{-1} \gamma_B; x_C := M_{i+1}^{-1} \gamma_C.
\]

3. **Upper triangular solver (\( \text{UpTSv} \)):**

   \[
   \left[ \begin{array}{ccc} L^T & L^T & 0 \\ 0 & 0 & I \end{array} \right] \left[ \begin{array}{ccc} w_B \\ w_C \\ w_C \end{array} \right] = \left[ \begin{array}{ccc} x_B \\ x_C \\ x_C \end{array} \right] \Rightarrow w_C := x_C; L_B^T w_B = x_B - L_e^T w_C.
\]

Therefore, the \( \text{PTAP} \) in ILUPACK operates two sparse matrix-vector multiplications and solves two linear systems of the form \( LDL^T x = b \). Additionally, it performs three other types of operations: diagonal scaling, vector permutation, and vector updates of the form \( x := a - b \).

### 2.2 Task-parallel ILUPACK

**Nested dissection.** The parallel version of ILUPACK can be exposed by means of nested dissection orderings, which reveal parallelism exploiting the connection between sparse matrices and adjacency graphs. In particular, nested dissection algorithm partitions the adjacency graph \( G(A) \) associated to the approximate factorization of \( A \) into a hierarchy of vertex separators and independent subgraphs.\(^{17}\) For example, in Figure 3, \( G(A) \) is partitioned after two levels of recursion into four independent subgraphs. This hierarchy can be constructed using METIS software,\(^{18}\) which minimizes the size of the vertex separators balancing the size of the independent subgraphs.\(^{19}\) Therefore, relabeling the nodes of \( G(A) \) according to the levels in the hierarchy produces a reordered matrix, \( A := P^T A P \), with a structure sensitive to efficient parallelization. Concretely, the leading diagonal blocks of \( P^T A P \) associated with the independent subgraphs can be first computed independently; after that, \( S_{2,2} \) and \( S_{3,3} \) can be processed in parallel, and finally, the separator \( S_{1,1} \) is calculated. This type of concurrency can be expressed as a binary task-dependency graph (TDG), where the nodes represent simultaneous tasks and the edges dependencies among them.

**Computation of the preconditioner.** In order to design a task-parallel version of ILUPACK, we decouple the computation of the preconditioner into tasks, identifying the dependencies among them, and mapping the tasks to the execution nodes. For that purpose, the task-parallel version
exploits the connection between sparse matrices and adjacency graphs, extracting parallelism via nested dissection, as explained before. Consider, for example, the TDG in Figure 4 such that ILUPACK partitions the coefficient matrix $A$ into four independent submatrices. The reordered matrix $(P^T A P)$ can be decomposed in submatrices as follows:

$$
\begin{align*}
A_{00} & = A_{00} + A_{24}, \\
A_{11} & = A_{11} + A_{25}, \\
A_{22} & = A_{22} + A_{36}, \\
A_{33} & = A_{33} + A_{46}, \\
A_{44} & = A_{44} + A_{66}, \\
A_{55} & = A_{55} + A_{66}.
\end{align*}
$$

where

$$
\begin{align*}
A_{00} & = A_{00} + A_{24}, \\
A_{11} & = A_{11} + A_{25}, \\
A_{22} & = A_{22} + A_{36}, \\
A_{33} & = A_{33} + A_{46}, \\
A_{44} & = A_{44} + A_{66}, \\
A_{55} & = A_{55} + A_{66}.
\end{align*}
$$

After this reorganization, the leading blocks of these four submatrices can be factorized in parallel, whereas the blocks of the other levels have dependencies with the ancestor tasks. These blocks will start the factorization when the ancestor tasks have finalized. This process continues traversing the TDG, until the root task factorizes its local submatrix.

**Application of the preconditioner.** As we stated in the previous section, this operation in ILUPACK lacks the solution of two triangular systems (lower and upper triangular factors). The TDG has to be traversed twice per solve $z_{k+1} := M^{-1} r_{k+1}$ at each iteration of the PCG. Hence, although the TDG associated with the first triangular system ($LwTrSv$) presents the same structure and dependencies as that related to the preconditioner computation, in the latter triangular solve ($UpTrSv$), the dependencies are reversed (from the root to the leaves). For that reason, the amount of parallelism expands/reduces as we progress toward/away from the leaves.

**Other kernels in the PCG iteration.** The vectors involved in the PCG are partitioned conformally to matrix $A$, see (4). Accordingly, the $SpMV$, $Axpy$, and $Axpy$-like kernels only operate with the data related to the leaves of the TDG. For example, for a matrix partitioned as in (4), the $SpMV$ is decomposed into four matrix-vector products, so that the processing of each one of these four leaves is totally independent from the others. The
procedure in the \texttt{Axpy} operations is the same, and thus, this operation can be fully computed in parallel. On the other hand, the \texttt{Dot} and the \texttt{2-norm} require a reduction of the results computed in the leaf nodes, which needs a synchronization point.

\textbf{Mapping tasks to cores.} In this paper, we explore a data-flow version of ILUPACK using an ad-hoc runtime based on OpenMP\textsuperscript{21} developed in the work of Aliaga et al\textsuperscript{22} in which each operation appearing in ILUPACK’s PCG is decomposed into a number of tasks that are then mapped on the cores to be executed. The mapping is dynamically defined during the \texttt{PrCo}, through the use of a shared task queue, on which the active tasks are inserted and the cores access to demand new work. Initially, only the leaf nodes of the TDG are stored in the queue, whereas the intermediate nodes are introduced when their children have been completed. Later, the same mapping is used during the completion of the operations included in each PCG iteration. To do this, task queues are created in each core to manage the tasks that were mapped in the core during \texttt{PrCo}. For \texttt{LwTrSv}, the management of the tasks is the same as in \texttt{PrCo}, but for \texttt{UpTrSv}, the TDG is traversed in reverse, and therefore, the task queues are initiated empty and the leaf nodes are added to the task queues when their parents complete their execution. Although there are no intermediate nodes during the other PCG operations, task queues, initiated by the leaf nodes, are also managed to complete these operations, in order to use the same mapping in all the computations.

In practice, the number of tasks of each operation exceeds the number of cores, since this produces a more balanced workload during the execution. To improve it, the nodes are sorted in the shared task queue. In this way, for \texttt{LwTrSv}, the leaf nodes are processed before the intermediate nodes, and the nodes with a longer number of non-zeros are processed first. However, for \texttt{UpTrSv}, an opposite methodology is applied because the intermediate nodes should be computed as soon as possible. Although for \texttt{LwTrSv} and \texttt{UpTrSv}, most of the computational work is concentrated in the leaf nodes of the TDG, the processing of the intermediate nodes introduces some overhead, and therefore, the practical number of leaf nodes are limited. Moreover, the sizes of the operands in the \texttt{SpMV} and the vector operations also grow along with the number of levels in the TDG, causing additional overhead. The take-away from this discussion is that, when deciding the number of levels/leaf nodes of the TDG, there is a trade-off between workload balancing and cost of processing the intermediate nodes of the TDG.

Figures 5 and 6 display, respectively, the \texttt{Extrae}\textsuperscript{23} traces for the preconditioner computation and one iteration of the PCG solver, executed on an Intel Xeon 8-core processor using 8 threads. The TDG in this example is composed of 32 leaves (4 leaf tasks per thread) and 6 levels. The color of each area in the trace determines the operation (see legends). These traces show that, even with 4x more leaf nodes than threads, there still appear significant idle times due to the implicit barrier at the end of parallel regions for \texttt{PrCo}, \texttt{SpMV}, \texttt{LwTrSv}, and \texttt{UpTrSv}. This behavior motivates the development of different approaches to save energy, which are described in the next section.

\section{ENERGY-AWARE TECHNIQUES ON ILUPACK}

The reduction of the energy consumption in ILUPACK can be tackled tuning the C-states and P-states using different approaches. In this section, we consider three energy-aware strategies.

\subsection{Race-to-halt/race-to-idle}

This strategy reduces the energy-consumption during the concurrent execution of ILUPACK transforming the “busy-wait” behavior (threads are polling until a new task is available) to “idle-wait” (threads are blocked until a new task is ready). Benefits should be obtained because the operating system promotes the hardware into a more energy-efficient C-state.
The analysis of Figures 5 and 6 reveals two types of wait-time energy-drains.

- Red areas indicate an implicit OpenMP barrier at the end of parallel regions.
- Blue areas denote idle time related to task queues management.

The length of the first type can be reduced by assigning the value `PASSIVE` to the environment variable `OMP_WAIT_POLICY`. In this way, PCG operations where the computation only affects the leaf nodes (SpMV and vector operations) are optimized. For the second type, the solution is to synchronize the access to the task queues using mutex and condition variables. This solution can be also applied to the operations that traverse the TDG, that is, PrCo and PrAp.

### 3.2 Dynamic VFS

The main objective of this strategy is to apply a frequency-tuning policy, reducing the waiting time as soon as possible. To complete this, the code integrates a mechanism to detect which threads experience waits. Then, it adjusts the best P-state for the execution of each task in order to reduce the waiting time of those threads. Taking into account that the execution time of the tasks has to be measured several times to take the best decision, it does not make sense to apply this technique to the preconditioner, but it can be perfectly included in the PCG iteration. The overhead related to this technique suggested to apply it only to the most expensive operations of the iterative solver: SpMV, LwTrSv, and UpTrSv.

In Figure 6, it is easy to identify the slowest thread of the SpMV, but its calculation requires to measure the computational time of each task \( \text{time}(th) \) and, then, to locate the thread with the largest execution time \( th_{\text{max}} \). During this computation, the initial P-state of all the tasks is P0 and, after the thread \( th_{\text{max}} \) is identified, the remaining threads increase the P-state of its last executed task. Afterwards, the execution time of each thread is measured again. If the measured value for a thread is greater than the current execution time for \( th_{\text{max}} \), the P-state of its last task is decreased, and the thread is removed from the procedure. Otherwise, a new increment on the P-state of the last executed task is made. When the last task of a thread reaches the maximum P-state and a new increment is due, the next-to-last task has to be modified; in fact, the detection of the last task of a thread always excludes the tasks whose P-state has arrived to the maximum. Figure 7 shows an algorithmic formulation of the described process.

The SpMV computation only involves leaf nodes. Therefore, the implementation of the DVFS approach is direct. In contrast, LwTrSv and UpTrSv also involve intermediate nodes. Taking into account that the weight of these nodes is relatively small in the global computation, intermediate nodes related to both operations are not considered in this strategy. In LwTrSv, the execution of the DVFS approach is quite stable because all the threads start the processing of the leaf nodes at the same time, whereas the technique ensures that all the threads finalize their computation at about the same time, reducing the idle time (blue area in the traces). On the other hand, the beginning of the leaf nodes in UpTrSv is unknown because the execution of the intermediate nodes can be delayed in some scenarios (see Figure 8). Hence, it is more complex to adjust the finalization of the threads in this operation. Moreover, the execution of the leaf and intermediate nodes can be interleaved during the execution of the PCG, yielding more complex scenarios. In any case, the DVFS approach is also useful in UpTrSv, reducing the waiting time related to the implicit barriers (red area in the traces). Figure 8 shows how the trace changes when the DVFS approach is applied to PCG.

### 3.3 Memory-bound aware

The energy-aware implementations differ for CPU-bound and memory-bound operations. For the first ones, the best option is usually to execute the code using the highest CPU frequency. However, the strategy may be suboptimal for the second ones because the execution is limited by the
memory transfer rate. In this case, the selection of an appropriate CPU frequency allows to adjust memory and CPU rates, trading off the execution time for power consumption, such that the energy efficiency is properly improved.

Many of the computations in the PCG are BLAS-1 and BLAS-2 operations, which are memory-bound operations. As in the previous section, we only consider the most expensive operations (the leaf nodes of SpMV, LwTrSv, and UpTrSv) that include different operations applied on sparse matrices, with a low memory transfer rate. Therefore, the best energy-aware implementation consists in determining the best P-state related to the corresponding tasks, instead of the highest frequency. However, the best frequency depends on the data (including its partition and its mapping) and on the machine architecture; therefore, it is known prior to the solution of the linear system.

A procedure similar to that described in Figure 7 can be defined to find the optimal frequency (see Figure 9), but, in this case, the execution time metric is changed to energy consumption. Moreover, this procedure always considers the whole operation, avoiding individual threads, and consequently, the changes on the P-states affect all individual tasks. Thus, the variable EnCon refers to the energy consumption of the analyzed operation. The main objective of the procedure is to increase the P-state of the threads as long as the last two energy-consumption readings satisfy a condition. Otherwise the P-state is decreased and the procedure finalizes. This condition incorporates the parameter ErrAllowed, which controls the relationship between the two readings. In this way, a value of the parameter greater than one allows to change the P-state of the threads although it produces a small loss of performance.

3.4 Combining energy-aware methodologies

The techniques described in the previous sections are orthogonal, and therefore, they can be combined. In theory, the maximum benefit should be achieved when the three methodologies are applied, indeed this assertion will be confirmed by means of the experimentation included in the next section. Note that when the DVFS approach and the MBA strategy are combined, the first one has to be changed because the application of DVFS...
has to be made when the MBA strategy has finalized. In this way, the first line of the algorithm in Figure 7 has to be removed because the initial frequency of the threads has been previously fixed using the algorithm of Figure 9.

4 | EXPERIMENTAL RESULTS

In this section, we present the impact of the energy-aware techniques on ILUPACK. With this aim, we compare the performance and energy efficiency of the implementation without any energy-aware technique, which is referred to as Performance-Oriented (PO), and the implementations on which some of the energy-aware techniques have been included.

4.1 | Hardware and software setup

For the experiments in this section, we employ a server equipped with two 8-core Intel Xeon(R) E5-2630 processors (Haswell-EP), running at 2.4 GHz, with 20 MB of L3 on-chip cache (LLC or last level of cache) each and with 64 GB of DDR3 RAM. Hyperthreading is not active. The operating system running in the server is Linux version 2.6.32-642.4.2.el6.centos.x86_64, the compiler is gcc 4.4.7 using the -O3 compiler optimization, and ILUPACK’s version is 2.4.5 The userspace Linux governor allows the processor cores to operate at 13 possible frequencies ranging from 1.2 GHz to 2.4 GHz, with a stride of 0.1 GHz. Each core can be set to a different frequency.

For the generation of the first test matrix, $A_{200}$, a large-scale linear system for the Laplacian equation $-\Delta u = f$ in a 3D unit cube $\Omega = [0,1]^3$ with Dirichlet boundary conditions, $u = g$ on $\partial \Omega$, was generated, whose discretization is a sparse symmetric positive system. The other two matrices in the experimentation (audikw_1 and ldoor) correspond to examples from the SuiteSparse Matrix Collection, with close to 1,000,000 rows/columns and different sparsity patterns. Table 1 shows some relevant features of these matrices.

Energy was measured using Intel’s RAPL (Running Average Power Limit) interface, reflecting the estimated consumption of the core-uncore (package), DRAM, and the total (core, uncore and DRAM) system. For the Haswell-EP the isolated on-core consumption is not provided by RAPL. The idle energy was obtained by executing the Linux sleep command during 60 seconds in all cores. This value was then subtracted to the total energy in order to obtain the net energy. The experiments were executed after a warm up period of 120 seconds using a busy-wait loop, and each experiment was repeated 10 times, showing the average values. Moreover, we use the cpufrequtils package to vary the CPU performance state. The experiments analyze the performance and energy efficiency of the different implementations when they are executed on a single socket. Therefore, we only report the results of the corresponding 8-core processor.

| Matrix          | n       | nnz_L | nnz_L/n | condest | nIter | nnz_M | $\bar{t}_{PO}$ |
|-----------------|---------|-------|---------|---------|-------|-------|----------------|
| $A_{200}$       | 8,000,000 | 31,880,000 | 3.99    | 2.73E+04 | 107   | 14.87  | 0.47           |
| audikw_1        | 943,695  | 39,297,771 | 41.64   | 6.95E+10 | 815   | 95.76  | 0.22           |
| ldoor           | 952,203  | 23,737,339 | 24.93   | 2.46E+08 | 485   | 38.08  | 0.09           |
4.2 | Experimental setup

The right-hand side vector \( b \) in the iterative solvers was always initialized to the product \( A(1, 1, \ldots, 1)^T \), and the PCG iteration was started with the initial guess \( x_0 = 0 \). The parameter that controls the convergence of the iterative process in ILUPACK, \( \text{restol} \), was set to \( 10^{-6} \), whereas the drop tolerance and the bound to the condition number of the inverse factors, which control ILUPACK’s multilevel incomplete factorization process, were set to 0.01 and 5, respectively. The approximate number of PCG iterations required to solve the corresponding linear system using these parameters, for a 32-leaf nodes TDG and employing IEEE754 real double-precision arithmetic, is also included in Table 1.

For each implementation, we have considered two different scenarios: balanced and unbalanced mappings. The first one is the default use of the library, in which the leaf nodes are sorted in the shared task queue regarding the number of nonzero elements, so that, during the preconditioner computation, the costlier nodes are factorized before. On the contrary, the second one has been manually built to generate an unbalanced execution, so that it is possible to verify how the energy-aware strategies are adapted to these situations. In each one of these scenarios, the same mapping of tasks to cores has been used for all the energy-aware implementations. Under these conditions, the differences are only related to the saving features of the corresponding techniques.

We have applied an incremental methodology in order to analyze the impact of each technique in the energy-consumption improvement. Thus, each new tested implementation always incorporates the previously techniques. Concretely, the DVFS implementations also include the RTH strategy, and the MBA implementations also use the two previous strategies.

The DVFS approach and the MBA strategy require to measure the energy consumption of each core when the operations are executed. To avoid the impact of errors in the measurement, both strategies accumulate the results of several PCG iterations. In theory, evaluating a larger number of iterations should improve the correctness of the measures, but the adjustment period and its overhead are extended. In our experimentation, 2 or 4 iterations have been tried, and similar improvements were obtained, although the overhead was smaller in the first case. Therefore, only the results for 2 iterations are shown next.

The parameterization of the MBA strategies requires to fix the parameter \( E\text{rAllowed} \), which was set to 1.000 or 1.005, and to determine on which type of tasks the P-state is changed. Our experiments show that it is more efficient to modify only the frequency of leaf nodes than to change the frequency for all the tasks.

4.3 | Analysis of the results

The performance and energy efficiency results are displayed by means of the tables, in which we expose the relative improvement of the corresponding variant with respect to the PO implementation. This computation is calculated as follows:

\[
\text{res}(\text{val_{IMP}}) = \frac{\text{ref}_{\text{PO}}}{\text{val}_{\text{IMP}}} - 1,
\]

where \( \text{ref}_{\text{PO}} \) is the value of the PO implementation related to the matrix that is used in the studied implementation (IMP). Note that negative values for this expression reflect a decrease of the performance, whereas positive values reveal improvements. In the tables, we multiply this result by 100, in order to show the corresponding percentage.

Tables 2 to 4 show the performance and energy efficiency of the different strategies for the matrices in Table 1. In general, the first analysis of these tables can conclude that the energy improvements for \textit{audikw.1} are always greater than those for A200 because the nonzero pattern of the first one is irregular, which yields a more difficult scenario to define a perfectly balanced mapping. In this way, the execution of \textit{audikw.1} includes more idle periods, and therefore, it offers more options to apply energy-aware techniques. The energy improvements of \textit{ldoor} are the worst in all the strategies because the execution time per PCG iteration of this matrix is smaller. The reason is that this matrix and its factor are sparser, and therefore, there exist less chances to reduce wait-time drains. This is especially clear when the MBA strategy improvements for \textit{ldoor} and the other two matrices are compared. The comparison of the two mappings for the three matrices renders the same conclusion: these strategies are more profitable for unbalanced mapping because there are more idle periods.

**Race-to-halt strategy.** The first remark after applying the RTH methodology is that, for the three matrices, types of mappings, and stages (PrCo and PCG), the impact of this strategy on the execution time is practically negligible because it is always less that 0.06%. In several cases, some minor improvements are achieved, arriving to 0.44%.

The same conclusion can be obtained for the DRAM values, since the use of the memory in this implementation has not changed with respect to the PO case. Therefore, the DRAM changes are directly related to the changes in the time.

The analysis of the total and net values of energy reveals the main advantage of this implementation, that is, the reduction of the package values and, consequently, the global ones. This improvement is more visible for the net energy and net power values.

For both matrix mappings, the previous assertions are fulfilled: this strategy reduces the energy-consumption and maintains the execution time. The relative variations of the net package energy for the PrCo are remarkable because they are higher than 5%, 14%, and 9%, respectively, for the unbalanced mapping of the three matrices. Furthermore, the increment of the net package energy for the PCG stage is interesting. It is higher than 3.3%, 5.3%, and 5.3%, respectively, when we use the unbalanced mapping of the matrices. Moreover, the improvements are also appreciable for the balanced mappings.
TABLE 2  Relative variation (in %) of the energy-aware variants with respect to PO, considering the balanced and unbalanced mappings of the A200 matrix when a 32-leaf TDG is processed by 8 cores. For DVFS approach, the values in Add Steps column refer to individual frequency changes, whereas for MBA strategy the bold values in this column represent changes in the 32-leaf nodes of the TDG

|                | Total Energy | Balanced Mapping | Unbalanced Mapping |                |                |                |                |                |                |
|----------------|--------------|------------------|--------------------|----------------|----------------|----------------|----------------|----------------|----------------|
|                | Pack         | Dram             | Global             | Pack           | Dram           | Global          | Pack           | Dram           | Global          |
| RTH(PcCo)      | 4.26         | 0.06             | 3.89               | 6.74           | 0.02           | 6.20            | 0.11           | 6.62           | 0.09            |
| RTH(PCG)       | 1.77         | -0.03            | 1.58               | 2.59           | -0.06          | 2.30            | 0.05           | 2.54           | -0.12           |
| DVFS_1         | 1.91         | -0.20            | 1.68               | 2.91           | -0.21          | 2.57            | -0.18          | 3.10           | -0.03           |
| DVFS_2         | 2.25         | -0.16            | 1.99               | 3.37           | -0.19          | 2.98            | -0.09          | 3.46           | -0.10           |
| DVFS_3         | 2.42         | -0.63            | 2.09               | 4.03           | -0.52          | 3.53            | -0.89          | 4.96           | 0.38            |
| PCK_0.0        | 12.66        | -5.76            | 10.40              | 24.80          | -5.23          | 20.74           | -6.93          | 34.09          | 1.82            |
| PCK_0.5        | 12.63        | -7.45            | 10.12              | 26.82          | -6.58          | 22.18           | -9.33          | 39.86          | 3.03            |
| GBL_0.0        | 12.36        | -5.43            | 10.18              | 24.15          | -4.81          | 20.27           | -6.80          | 33.20          | 2.13            |
| GBL_0.5        | 12.53        | -7.10            | 10.09              | 26.45          | -6.18          | 21.95           | -9.11          | 39.12          | 3.23            |

Dynamic VFS-based approach. Following the incremental methodology mentioned earlier, we applied several DVFS implementations on the RTH case. Here, we consider three implementations.

- **DVFS_1**: The DVFS approach is only applied on SpMV.
- **DVFS_2**: The DVFS approach is applied on SpMV and LwTrSv.
- **DVFS_3**: The DVFS approach is applied on SpMV, LwTrSv, and UpTrSv.

A new column (Add Steps) is included in Tables 2 and 3, reporting the number of P-state changes (see Figure 7) with respect to the previous row in the table. Therefore, all the values in the DVFS_1 row correspond to the application of the DVFS approach on SpMV, the DVFS_2 row shows the impact of the strategy on LwTrSv, whereas the DVFS_3 row is focused on UpTrSv.

In this approach, the impact on the execution time is still really small, since the reduction is always below 0.9%. There appear some small improvements, close to 0.6%, revealing one of the strengths of the strategy. The growth of the execution time is basically due to the overhead of the code in Figure 7, and therefore, it grows with the number of P-state changes.

Unlike the RTH strategy, now there is no direct relationship between DRAM values and execution time. One might expect that the memory would consume more energy because the threads spend a longer period in operation, executing and demanding data from memory, but the tables expose that this assumption does not hold in many cases. If the increment of the execution time is mainly related to the overhead of this strategy, it does not have any influence on the memory. Probably, the values of the tables show that the reduction of idle periods also reduces the overhead related to C-states transitions, and therefore, the threads are less time executing tasks, diminishing the memory working-time. This conclusion holds except for the case with more P-state changes (unbalanced mapping for audikw_1), presumably because, in that case, the additional active time of the threads is higher than the C-state savings.

Again, the improvements of the total and net energy values are the main benefits of this strategy. The tables exhibit the positive impact on the energy and power consumption of changing P-states, even though the transitions introduce additional overheads. Overall, the net package energy savings for the unbalanced mapping are close to 5%, 14.5%, and 14%, respectively, for the three matrices, whereas they are still visible for the balanced mapping for matrices A200 and audikw_1, improving 4.03% and 5.81%, respectively.

The analysis of the tables allows to conclude that the number of P-state changes directly influences the energy improvements. The experiments with the unbalanced mapping in the three matrices produce an augmentation of both number of P-state changes and energy improvements. The comparison between the different operations/stages is not clear, being dependent of the matrix and the mapping. The number of P-state changes is especially relevant on the LwTrSv and UpTrSv for the unbalanced mapping of the audikw_1, with values of 140 and 129, respectively.
### Memory-bound aware strategy.

The main objective of this strategy is to minimize the energy consumption of the implementation. Here, it is possible to use several methodologies to achieve this objective. We have considered the following four different variants.

- **PCK_0.0**: On DVFS_3, the package energy is minimized, allowing no error.
- **PCK_0.5**: On DVFS_3, the package energy is minimized, allowing a 0.5% of error.
- **GBL_0.0**: On DVFS_3, the global energy is minimized, allowing no error.
- **GBL_0.5**: On DVFS_3, the global energy is minimized, allowing a 0.5% of error.

In this strategy, the meaning of the last column in the tables (Add Steps) is different from that in the DVFS approach because, now, the values in this column do not represent the number of P-state changes applied on single tasks, but they refer to changes on all the leaf node tasks. Therefore, to obtain the real number of P-state, the bold values in the tables should be multiplied by 32.

The increase of the execution time of the MBA strategy is directly related to the error threshold, such that this time is augmented when the error is increased. In any case, the maximum decrease is around 10%. Moreover, the improvement of the total energy is always greater than the reduction of the performance. Furthermore, the comparison of balanced and unbalanced mapping allows to conclude that its behavior is due to the error rather than to the mapping. Thus, for A200, the error = 0.0 cases are close to 6.85% whereas the error = 0.5 cases are close to 9%.

It is also remarkable that the bold values for the balanced and unbalanced cases are really similar, but they are matrix-dependent. For A200 and audi2k_1, those values are, respectively, in the intervals (21, 27) and (21, 26), whereas the interval for the 1door is (7, 12). The small values of the last interval are due to the corresponding reduced execution time. Therefore, it is not possible to adjust the CPU frequency and the memory transfer rate. On the other hand, the similarity of the two previous intervals fixes the bold values in which the CPU and the memory rates are perfectly adjusted.

The increment of the execution time has a direct relationship with the DRAM consumption, although its benefits are more moderate. The reason is that this strategy reduces the DRAM net power. In this way, the pair (net energy, net power) for the unbalanced mapping of the three matrices are, respectively, close to (−6%, 3%), (−7%, 3%), and (−5%, −2%) when the error is allowed and (−5%, 2%), (−5%, 0%), and (−4%, −3%) if not.

The reduction of the energy consumption is relevant for the package and global values. Minimizing global energy produces improvements on the package energy, occasionally, even more, that if only package energy is optimized. Moreover, allowing some errors improves the energy consumption in many cases. The savings are huge, with the improvements of the package net power being greater than 39%, 53%, and 25%, respectively, for the unbalanced mapping of the three matrices; and close to 39%, 45%, and 15% for the balanced case. The similarity between the balanced and

---

### Table 3: Relative variation (in %) of the energy-aware variants with respect to PO, considering the balanced and unbalanced mappings of the audi2k_1 matrix when a 32-leaf TDG is processed by 8 cores. For DFVS approach, the values in Add Steps column refer to individual frequency changes, whereas for MBA strategy, the bold values in this column represent changes in the 32-leaf nodes of the TDG

|                  | Balanced Mapping | Unbalanced Mapping |
|------------------|------------------|--------------------|
|                  | Pack Dram Global | Pack Dram Global   |
|                  | Total Energy     | Net Energy         |
|                  |                  | Time               |
|                  |                  | Net Power          |
|                  |                  | Add Steps          |
|                  |                  |                    |
| RTH(PrCo)        | 8.65 0.36 8.12   | 13.77 0.22 13.26   |
|                  |                  | 0.44 0.12 0.06 0.27 |
|                  |                  | 12.76 -0.22        |
| RTH(PCG)         | 1.96 0.10 1.77   | 2.85 0.12 2.57     |
|                  |                  | 0.06 0.07 0.07 0.07 |
|                  |                  | 0.25              |
| DVFS_1           | 2.18 0.38 2.00   | 3.24 0.59 2.97     |
|                  |                  | -0.08 3.32 0.67    |
|                  |                  | 3.05 41            |
| DVFS_2           | 3.25 0.41 2.96   | 4.70 0.51 4.27     |
|                  |                  | 0.18 4.51 0.33     |
|                  |                  | 4.08 47            |
| DVFS_3           | 3.87 0.21 3.50   | 5.81 0.38 5.25     |
|                  |                  | -0.16 5.98 0.55    |
|                  |                  | 5.42 28            |
| PCK_0.0          | 16.09 -5.26 13.55| 30.28 -4.68 25.76  |
|                  |                  | -6.46 39.26 1.90   |
|                  |                  | 34.43 22            |
| PCK_0.5          | 16.41 -6.70 13.62| 32.87 -5.75 27.75  |
|                  |                  | -8.63 45.41 3.15   |
|                  |                  | 39.81 25            |
| GBL_0.0          | 15.87 -4.82 13.42| 29.34 -4.33 25.03  |
|                  |                  | -5.82 37.32 1.58   |
|                  |                  | 32.75 21            |
| GBL_0.5          | 16.19 -6.68 13.43| 32.71 -5.58 27.65  |
|                  |                  | -8.90 45.64 3.63   |
|                  |                  | 40.08 26            |
unbalanced cases for A200 and audikw_1 confirms that the adjustment of the CPU frequency and memory transfer rate are achieved when the bold values are between 21 and 27. Finally, although it could be expected, the improvement obtained with an error of 0.5% should enhance that computed without any error; this is not always true.

5 | CONCLUSIONS

Several energy-aware strategies have been introduced in this paper to improve the energy efficiency of the task-parallel version of ILUPACK, focusing the study in the iterative solve of SPD sparse linear systems. The RTH strategy and the DVFS strategy manage, respectively, the C-states and the P-states of the cores to reduce the energy consumption with a negligible impact on the execution time. Combining these two strategies, the MBA strategy allows to achieve higher energy savings at the cost of longer execution time. In this case, the net global energy figures are, respectively, close to 4%, 13%, and 12% for the tested three matrices. Additionally, the inclusion of the DFVS approach the values in Add Steps column refer to individual frequency changes, whereas for MBA strategy the bold values in this column represent changes in the 32-leaf nodes of the TDG.

In future work, we will consider to extend these techniques for other iterative solvers, on different multicore architectures and on clusters of multicores. Nowadays, almost all architectures implement energy-aware techniques that manage C-states, and therefore, the RTH strategy can be directly applied. However, only some of them allow to change the core frequency separately. If the P-state change affects to all the cores of the processor, only the MBA strategy can be used. Otherwise, both MBA and DVFS can be leveraged. Moreover, MBA strategy requires RAPL interface to obtain the energy consumption. On clusters, the implementation of the RTH strategy will depend on the MPI implementation, which it should allow to use "idle-wait" instead of "busy-wait" in the communications. Moreover, MBA will always adjust the CPU frequency and the memory transfer rate, whereas, if DFVS can be applied, it will allow to reduce the desynchronization of the task execution both among the processor cores and in different nodes of the cluster.

For new CPU architectures on which the Energy-Aware Race to Halt (EARtH) algorithm adapts the CPU frequency to the workload (Intels Speed Shift technology\textsuperscript{27}), the strategies will have to be changed. Thus, the MBA strategy will be substituted by the EARtH algorithm to determine the best frequency to balance the CPU frequency and the memory transfer rate. Later, the DFVS will be applied to fix the best frequency for each task taking advantage of the knowledge of the PCG operations, such that maximum savings are getting.

**TABLE 4** Relative variation (in %) of the energy-aware variants with respect to PO, considering the balanced and unbalanced mappings of the \texttt{idooor} matrix when a 32-leaf TDG is processed by 8 cores. For DFVS approach the values in Add Steps column refer to individual frequency changes, whereas for MBA strategy the bold values in this column represent changes in the 32-leaf nodes of the TDG.
ACKNOWLEDGMENTS

This work was supported by the CICYT projects TIN2014-53495-R and TIN2017-82972-R of the MINECO and FEDER, by the H2020 EU FETHPC Project 671602 "INTERTWinE", and by the project P1-1B2015-26 of the Universitat Jaume I.

ORCID

María Barreda  http://orcid.org/0000-0002-8526-1140

REFERENCES

1. Duranton M, Yehia S, De Sutter B et al. The HiPEAC vision. http://www.hipeac.net/roadmap. Retrieved July 2017.
2. Fuller SH, Millett LI. The Future of Computing Performance: Game Over or Next Level? Washington, DC: National Academies Press; 2011.
3. Esmaeilzadeh H, Blem E, Amant RS, Sankaralingam K, Burger D. Dark silicon and the end of multicore scaling. Paper presented at: 38th Annual International Symposium on Computer architecture (ISCA’11); 2011; San Jose, CA.
4. Borkar S, Chien AA. The future of microprocessors. Commun ACM. 2011;54(5):67-77.
5. Bollhöfer M, Saad Y, Schenk O. ILUPACK - preconditioning software package. Release V2.4 June 2011. Available online at http://ilupack.tu-bs.de
6. Aliaga JI, Badia RM, Barreda M, et al. Exploiting task and data parallelism in ILUPACK’s preconditioned CG solver on NUMA architectures and many-core accelerators. Parallel Comput. 2016;54:97-107. https://doi.org/10.1016/j.parco.2015.12.004
7. Aliaga JI, Barreda M, Bollhöfer M, Quintana-Ortí ES. Exploiting task-parallelism in message-passing sparse linear system solvers using OmpSs. In: Euro-Par 2016: Parallel Processing Workshop. Cham, Switzerland: Springer; 2016.
8. Aliaga JI, Barreda M, Dolz MF, Martín AF, Mayo R, Quintana-Ortí ES. Assessing the impact of the CPU power-saving modes on the task-parallel solution of sparse linear systems. Clust Comput. 2014;17(2):1335-1348.
9. HP Corp, Intel Corp, Microsoft Corp, Phoenix Tech, Ltd, Toshiba Corp. Advanced configuration and power interface specification. Revision 6.1. 2016.
10. Aliaga JI, Barreda M, Castaño A. Harvesting energy in ILUPACK via slack elimination. Zenodo. 2017. https://doi.org/10.5281/zenodo.814115
11. Elnozaily EN, Kistler M, Rajamony R. Energy-efficient server clusters. In: Power-Aware Computer Systems Second International Workshop, PACS 2002. Lecture Notes in Computer Science; vol. 2325. Cambridge, MA: Springer-Verlag; 2003:179-197.
12. Wang L, Khan SU, Chen D, et al. Energy-aware parallel task scheduling in a cluster. Futur Gener Comput Syst. 2013;29(7):1661-1670.
13. Alonso P, Dolz MF, Mayo R, Quintana-Ortí ES. Energy-efficient execution of dense linear algebra algorithms on multi-core processors. Clust Comput. 2013;16(3):497-509.
14. Kolodziej J, Khan SU, Wang L, Byrski A, Min-Allah N, Madani SA. Hierarchical genetic-based grid scheduling with energy optimization. Clust Comput. 2013;16(3):591-609.
15. Bollhöfer M, Saad Y. Multilevel preconditioners constructed from inverse–based ILUs. SIAM J Sci Comput. 2006;27(5):1627-1650. Special issue on the 8th Copper Mountain Conference on Iterative Methods.
16. Aliaga JI, Bollhöfer M, Dufrechou E, Ezzatti P, Quintana-Ortí ES. Leveraging data-parallelism in ILUPACK using graphics processors. Paper presented at: IEEE 13th International Symposium on Parallel and Distributed Computing (ISPDC 2014); 2014; Marseilles, France.
17. Bollhöfer M, Aliaga JI, Martin AF, Quintana-Ortí ES. ILUPACK. In: Encyclopedia of Parallel Computing. Boston, MA: Springer; 2011:917-926. https://doi.org/10.1007/978-0-387-09766-4_513
18. METIS official home page. http://glaros.dtc.umn.edu/gkhome/metis/metis/overview. Accessed September 30, 2017.
19. Karypis G, Kumar V. A fast and high quality multilevel scheme for partitioning irregular graphs. SIAM J Sci Comput. 1999;20(1):359-392.
20. Saad Y. Iterative Methods for Sparse Linear Systems. Philadelphia, PA: SIAM; 2003.
21. The OpenMP API specification for parallel programming. http://www.openmp.org/specifications/. Accessed September 30, 2017.
22. Aliaga JI, Bollhöfer M, Martin AF, Quintana-Ortí ES. Exploiting thread-level parallelism in the iterative solution of sparse linear systems. Parallel Comput. 2011;37(3):183-202.
23. BSC. Extrae: User guide manual for version 3.5.2. https://tools.bsc.es/extrae. Accessed September 30, 2017.
24. Davis TA, Hu Y. The university of Florida sparse matrix collection. ACM Trans Math Soft. 2011;38(1):1-25. https://doi.org/10.1145/2049662.2049663
25. Intel Corp. Intel 64 and IA-32 architectures software developer manual. Volume 3B: System programming guide, Part 2; 2015.
26. ThinkWiki. How to use cpufrequtils. http://www.thinkwiki.org/wiki/How_to_use_cpufrequtils
27. Doweck J, Kao WF, Lu AK-Y, et al. Inside 6th-generation intel core: new microarchitecture code-named skylake. IEEE Micro. 2017;37(2):52-62. https://doi.org/10.1109/MM.2017.38

How to cite this article: Aliaga JI, Barreda M, Castaño A. Energy-aware strategies for task-parallel sparse linear system solvers. Concurrency Computat Pract Exper. 2019;31:e4633. https://doi.org/10.1002/cpe.4633