

Adapting Concurrency Throttling and Voltage-Frequency Scaling for Dense Eigensolvers

José I. Aliaga · María Barreda ·
M. Asunción Castaño · Manuel F. Dolz ·
Enrique S. Quintana-Ortí

Received: date / Accepted: date

Abstract In this paper we analyze the sources of power dissipation and energy consumption during the execution of high performance dense linear algebra (DLA) kernels on multicore processors. On top of this analysis, we propose and evaluate several strategies to adapt the concurrency throttling (CT) and the voltage-frequency setting (VFS) to obtain an energy-efficient execution of the DLA routine `dsytrd`. To design the strategies we take into account the differences between the memory-bound and CPU-bound kernels that govern this routine, and whether problem data fits into the processor's last level cache. Specifically, we experiment with these kernels to decide the optimal values of CT and VFS for an energy-aware execution of the `dsytrd` routine, and we also analyze the cost of changing CT and VFS.

Keywords Dense Linear Algebra · Eigenvalue Problems · Dynamic Concurrency Throttling (DCT) · Dynamic Voltage-Frequency Scaling (DVFS) · Energy Efficiency · Multithreaded BLAS · Multicore Processors

1 Introduction

The crucial role that a small collection of dense linear algebra (DLA) operations play in many scientific and engineering applications [?] has motivated, over the past decades, the development of highly tuned implementations of BLAS (*Basic Linear Algebra Subprograms*) [?] and LAPACK (*Linear Algebra PACKage*) [?] as, e.g., those included in Intel's MKL, AMD's ACML and IBM's ESSL. Unfortunately, in an era where power has become the key factor that constrains both the design and performance of current computer architectures [?,?], the kernels and routines in these libraries are largely optimized for raw performance (i.e., reduce execution

José I. Aliaga, María Barreda, M. Asunción Castaño, Enrique S. Quintana-Ortí
Depto. de Ingeniería y Ciencia de Computadores, Universidad Jaime I, 12.071–Castellón, Spain
E-mail: {aliaga,mvaya,castano,quintana}@uji.es

Manuel F. Dolz
Dept. de Informática, Universidad Carlos III de Madrid, 28911–Leganés (Madrid), Spain
E-mail: mdolz@inf.uc3m.es

time), either being completely oblivious of the energy they consume or operating under the assumption that tuning for performance is equivalent to optimizing energy.

Two crucial factors that control power dissipation and, in consequence, energy consumption of a multithreaded application running on a multicore processor, are the level of thread parallelism (concurrency throttling) and the core voltage-frequency setting [?, ?, ?]. However, the standard practice with DLA libraries simply dictates that the user sets the number of threads for the complete execution of the routines. In addition, these libraries mostly rely on the Linux governor modes to tune frequency and voltage (dynamic voltage-frequency scaling or DVFS [?]) during their execution.

In past work [?], we analyzed the *potential energy savings* that could be obtained via dynamic concurrency throttling (DCT) combined with DVFS for a key computational routine to tackle eigenproblems in LAPACK, namely the symmetric reduction to tridiagonal form (`dsytrd`), on an Intel Xeon E5-2620 (6-core) processor. In this paper, we extend that work to yield an *actual energy-efficient execution*, making the following new contributions in the process:

- We provide a complete experimental analysis, from the points of view of performance and performance-per-Joule (i.e., energy efficiency), of the two building blocks that govern the performance of `dsytrd`.
- In addition, we leverage the tool introduced in [?] to evaluate the transition cost of DVFS on the Intel Xeon E5-2620, exposing the potential negative impact (overhead) of this mechanism on a DLA operation.
- Combining the insights learnt from the studies in the two previous items, we adapt DCT and DVFS to deliver an energy-efficient multithreaded execution of `dsytrd`.
- We demonstrate the practical energy advantages and minimal impact on execution time of an adaptive control of DCT and DVFS for `dsytrd` on the Intel Xeon E5-2620, in particular compared with executions that rely on the standard Linux governor modes.

The rest of the paper is organized as follows. In Section 2 we briefly review the target DLA routine for our study. In Section 3 we offer a brief experimental evaluation of the performance-power-energy consumption of the building blocks for `dsytrd`. In Section ?? we evaluate the costs of DVFS on an Intel Xeon E5 processor, and then configure as well as analyze the adaptive mechanism to tune DCT and DVFS in routine `dsytrd`. Finally, the paper is closed with a few concluding remarks in Section ??.

2 Two-Sided Factorizations for Eigenvalue Problems

2.1 Overview

BLAS is organized into three groups or *levels*, known as BLAS-1, BLAS-2 and BLAS-3, with the kernels in the latter two respectively conducting a quadratic and a cubic number of flops (floating-point arithmetic operations) on a quadratic amount of data elements. On current cache-based architectures, tuned implementations of BLAS-3 generally deliver a high GFLOPS (billions of flops/sec.) rate,

close to the processor's peak, as they present a ratio of flops to memory operations that grows linearly with the problem size. On the other hand, the kernels in BLAS-2 cannot hide today's high memory latency, due to their low number of flops per memory access, and, in consequence, often deliver a low energy efficiency in terms of GFLOPS per watt (GFLOPS/W) as well; see, e.g., [?].

A significant fraction of LAPACK relies on tuned (possibly multi-threaded) implementations of BLAS to deliver portable performance over a variety of (multi-core) architectures. Among the contents of LAPACK, a few crucial two-sided factorization routines for the solution of standard/generalized symmetric/unsymmetric eigenproblems¹, which are responsible for most of the operations to tackle these complex problems [?], cast a large fraction of their flops in terms of BLAS-2 and BLAS-3 kernels. This is the case of the reduction to tridiagonal form (`dsytrd` operation), which is the first step towards the solution of the symmetric eigenvalue problem. In the remainder of this paper, we will address this particular operation to illustrate our energy-saving strategy, but the same ideas apply also to the reduction Hessenberg form for the unsymmetric eigenvalue problem (routine `dgehrd`).

2.2 Reduction to tridiagonal form via similarity transforms

Given a symmetric matrix $A \in \mathbb{R}^{n \times n}$, the standard routine in LAPACK for the reduction to tridiagonal form, `dsytrd`, yields the decomposition $A = Q^T T Q$, where $Q \in \mathbb{R}^{n \times n}$ is orthogonal and $T \in \mathbb{R}^{n \times n}$ is tridiagonal. In case only T is explicitly built, this routine requires $4n^3/3$ flops², roughly performing half of its flops in terms of BLAS-2 (mainly, via kernel `dsymv` for the symmetric matrix-vector product), while the other half is cast as BLAS-3 operations (concretely, via kernel `dsyr2k` for the symmetric rank- $2k$ update). The routine consists of a main loop that processes the input matrix A by blocks (panels) of b columns/rows per iteration. As the factorization proceeds, the size of the symmetric rank- $2k$ updates that are performed in the loop body decrease in the number of columns/rows, from $n - b$ towards 1 in steps of $k = b$ (algorithmic block size) per iteration. For `dsymv` the progression is from $n - 1$ to 1 in unit steps. The fragment of M-script code in Figure 1 illustrates this process.

In principle, the two BLAS kernels involved in `dsytrd` operation feature different properties:

- `dsymv` computes the matrix-vector product

$$y := \beta y + \alpha \hat{A}x, \quad (1)$$

where \hat{A} ($\equiv A(j+i : n, j+i : n)$) is a symmetric matrix of order $p = n - j - i + 1$; x, y are vectors with p entries; and α ($= 1$), β ($= 0$) are scalars. This operation conducts $2p^2$ flops on $p^2/2$ elements, as only the lower (or upper) triangular part of \hat{A} is accessed during the computation. Thus, there is a fixed ratio of 4 flops per matrix element read, which characterizes `dsymv` as a strongly memory-bound operation that naturally belongs to BLAS-2.

¹ The actual number of routines is larger, as there exist real/complex and single-/double-precision versions of these solvers that, in the symmetric cases, operate with the lower/upper triangular part of the matrix. Here we only focus on the real, double precision code, operating with the lower triangle.

² For simplicity, hereafter we neglect lower order terms in the arithmetic and storage costs.

```

1 for j = 1:b:n
2 %
3 % Reduce columns j:j+b-1 of A to tridiagonal form and form the
4 % matrix W which is needed to update the unreduced part of the matrix
5 %
6 for i = 1:b
7 %
8 % dsymv: Access only lower triangle of A( j+i:n, j+i:n )
9 %   W( j+i:n, i ) = A( j+i:n, j+i:n ) * A( j+i:n, j );
10 %   Computation of V( j+i:n, i )
11 %   ...
12 end
13 % ...
14 % dsyr2k: Update only lower triangle of A( j+b:n, j+b:n )
15 %   A( j+b:n, j+b:n ) = A( j+b:n, j+b:n ) - V( j+b:n, 1:b ) * W( j+b:n, 1:b )'
16 %                                     - W( j+b:n, 1:b ) * V( j+b:n, 1:b )';
17 % ...
18 end

```

Figure 1 Pseudo-code for the M-script implementation of `dsytrd` (n is multiple of b).

– `dsyr2k` computes the matrix-matrix products

$$\tilde{A} := \beta \tilde{A} + \alpha \tilde{V} \tilde{W}^T + \alpha \tilde{W} \tilde{V}^T, \quad (2)$$

updating only the lower (or upper) triangular part of the symmetric matrix \tilde{A} ($\equiv A(j+b:n, j+b:n)$), of order $m = n - j - b + 1$, with the products involving the two $m \times b$ panels \tilde{V} ($\equiv V(j+b:n, 1:b)$), \tilde{W} ($\equiv W(j+b:n, 1:b)$) and the scalars α ($= -1$), β ($= 1$). This kernel belongs to BLAS-3 and performs $2m^2b$ flops on $m^2/2 + 2mb$ elements. Thus, while `dsyr2k` is in principle a CPU-bound operation (assuming $m \approx b$, the ratio of flops to data accesses is $O(b)$), the operation becomes memory-bound when b is small.

3 Experimental Analysis of Building Blocks

In this section we illustrate the performance and energy efficiency of the building blocks `dsymv` and `dsyr2k`, under different configurations of DVFS–DCT, taking into account also a third parameter that plays a crucial role on these metrics, namely, whether the problem data fits into the processor’s last level cache (LLC).

For the experiments in the rest of the paper, we employ a server equipped with a 6-core Intel Xeon E5-2620 processor (2.0 GHz), with 16 MBytes of L3 on-chip cache (LLC or last level of cache), and 16 GBytes of RAM off-chip. Energy was measured using Intel’s RAPL (Running Average Power Limit) interface [?], capturing the consumption of the CPU and memory chips.

3.1 BLAS-2 kernel `dsymv`

Figure 2 reports the performance and energy efficiency attained by the kernel for the symmetric matrix-vector product (1) and two matrix sizes: $p = 1,000$ and 4,000. For the small problem dimension, the matrix occupies a bit more than

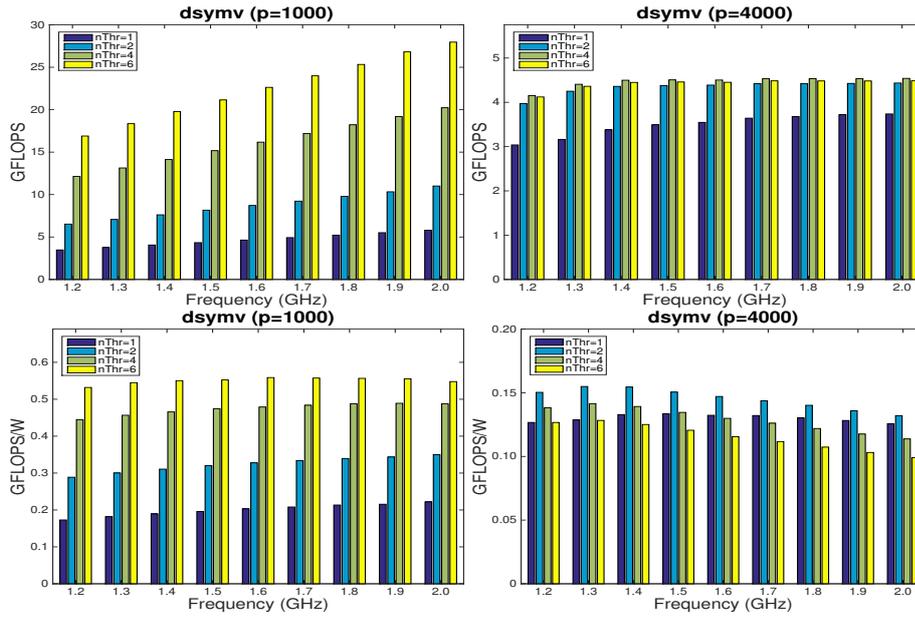


Figure 2 GFLOPS and GFLOPS/W (top and bottom, respectively) attained with `dsymv` kernel for the small and large problems (left and right, respectively), using different number of cores and voltage-frequency settings.

7 MBytes and perfectly fits into the LLC of the Intel Xeon E5-2620. In the second case, $p = 4,000$, the problem is well beyond the capacity of the LLC for this processor. Table ?? quantifies the speed-ups derived from these experiments taking as the reference the case of a serial execution (i.e., single core) at 1.2 GHz. These results expose a distinct behavior depending on the problem size. Let us consider first the scenario where the problem fits on-chip (i.e., the small problem). In this case, the GFLOPS rate perfectly scales with the frequency. For example, when a single core is employed and the frequency is raised from 1.2 to 2.0 GHz (i.e., by 66.7%), the performance also increases in the same proportion (speed-up of 1.66, see Table ??). A similar behavior is observed for any other number of cores; for example, with 6 cores, the observed speed-up is $8.03/4.85 = 1.65$. The scalability with the number of cores is not perfect, but still high: for example, the speed-up is 4.85 with 6 cores at 1.2 GHz, and $8.03/1.66 = 4.83$ when the cores operate at 2.0 GHz. The analysis of the energy efficiency for the small problem is slightly more complex, because of the combined effects of power/time on this metric and the multiple components of power (static vs dynamic and core vs uncore). In general, we observe an increase in the GFLOPS/W rate as the frequency is raised, with a stagnation of benefits at 1.9 GHz for 4 cores, and a negative impact when the frequency exceeds 1.6 GHz for 6 cores.

The behavior of `dsymv` when the problem lies off-chip is totally different. From the perspective of performance, there is only a minor increase when the number of cores is small and the source/target frequencies are low (i.e., the top left corner of the table for the large case). In addition, there is a clear negative impact for