Two-sided Orthogonal Reductions to Condensed Forms on Asymmetric Multicore Processors

Pedro Alonso[a], Sandra Catalán[b], José R. Herrero[c],
Enrique S. Quintana-Ortí[b], Rafael Rodríguez-Sánchez[b,*]

[a] *Dept. de Sistemas Informáticos y Computación, Univ. Politècnica de València, Spain*
[b] *Dept. Ingeniería y Ciencia de Computadores, Universidad Jaume I, Castellón, Spain*
[c] *Dept. d'Arquitectura de Computadors, Universitat Politècnica de Catalunya, Spain*

Abstract

We investigate how to leverage the heterogeneous resources of an Asymmetric Multicore Processor (AMP) in order to deliver high performance in the reduction to condensed forms for the solution of dense eigenvalue and singular-value problems. The routines that realize this type of two-sided orthogonal reductions (TSOR) in LAPACK are especially challenging, since a significant fraction of their floating-point operations are cast in terms of memory-bound kernels while the remaining part corresponds to efficient compute-bound kernels. To deal with this scenario: 1) we leverage implementations of memory-bound and compute-bound kernels specifically tuned for AMPs; 2) we select the algorithmic block size for the TSOR procedures via a practical model; and 3) we adjust the type and number of cores to use at each step of the reduction. Our experiments validate the model and assess the performance of our asymmetry-aware TSOR routines, using an ARMv7 big.LITTLE AMP, for three key operations: the reduction to tridiagonal form for symmetric eigenvalue problems, the reduction to Hessenberg form for general eigenvalue problems, and the reduction to bidiagonal form for singular-value problems.

Keywords: Dense linear algebra, condensed forms, eigenvalue problems, singular-value problems, asymmetric multicore processors, heterogeneous computing, multi-threading, workload balancing

*Corresponding author
Email addresses: palonso@upv.es (Pedro Alonso), catalans@uji.es (Sandra Catalán), josepr@ac.upc.edu (José R. Herrero), quintana@uji.es (Enrique S. Quintana-Ortí), rarodrig@uji.es (Rafael Rodríguez-Sánchez)

1. Introduction

We target the two-sided orthogonal reduction (TSOR) of a dense matrix to a condensed form (namely tridiagonal, Hessenberg or bidiagonal) as an initial step for the solution of dense eigenvalue or singular-value problems [1] on multi-threaded Asymmetric Multicore Processors (AMPs). Our interest in these architectures is motivated by the growing role of energy-efficient multicore systems-on-chip (SoC) on the road to exascale systems, and the challenge that represents how to exploit efficiently the heterogeneous types of cores in these architectures for dense linear algebra operations. An asymmetric big.LITTLE architecture presents the appealing property of being composed of two distinct types of cores, optimized for either raw performance and low power consumption. Energy simply reflects the consumption of power across a period of time. Therefore, a low-power symmetric multicore chip cannot always provide the most energy efficient solution. In particular, for some cases applications, a power-hungry but faster processor can provide a more energy-efficient solution while, for some others, it is more efficient from the point of view of energy to run the application at a lower pace (i.e., on a low-power processor). A symmetric multicore chip cannot provide an optimal configuration for both types of scenarios. Our motivation to target this type of heterogeneous architecture is that, in the dense linear algebra domain, the primary objective is performance and, therefore, it becomes crucial to exploit both types of cores present in the ARM big.LITTLE SoC.

Eigenvalue problems appear, among others, in computational quantum chemistry, finite element modeling, multivariate statistics, and density functional theory [2]. The computation of the singular values of a matrix is relevant, for example, in signal processing, big data, genomics, statistics, natural language text processing, etc. [1, 2].

Efficient and numerically reliable algorithms for the computation of the eigenvalues/singular values of a dense matrix consist of two stages [1]. The $m \times n$ input matrix A (with $m = n$ for eigenvalue problems) is first reduced to an $m \times n$ condensed matrix C via a sequence of orthogonal transformations applied from the left and right to A (two-sided transformations). This initial stage is then followed by the application of a specific solver to accurately compute the eigenvalues/singular values of C.

In this paper we describe several optimizations to the procedures that perform the TSOR stage of the matrix A to condensed form specifically designed for an ARM big.LITTLE AMP. The reason for addressing the first stage of the reduction only is that such transformations cost $O(n^3)$ floating-

point arithmetic operations (flops) while, in general, the second stage has a minor contribution to the total cost and performance of the solver. Furthermore, for simplicity, we assume that the eigenvectors/singular vectors of the problem are not requested. Otherwise, a few operations complete their computation as part of a third stage. This last stage is nonetheless straightforward to parallelize even on an AMP and, therefore, it is not discussed further.

LAPACK (*Linear Algebra PACKage*) [3] provides three main routines for TSOR to distinct condensed forms:

- SYTRD reduces a symmetric matrix to tridiagonal form via similarity (i.e., eigenvalue-preserving) transformations;
- GEHRD reduces a general matrix to Hessenberg form via similarity transformations; and
- GEBRD transforms a general matrix to bidiagonal form.

The former two routines are applied as an initial stage to compute the eigenvalues of a square matrix, while the last routine is the first step for the computation of the singular values. All three routines cast a significant part of its flops in terms of the Level-2 BLAS (*Basic Linear Algebra Subprograms*) [4] for the matrix-vector product, while the remaining flops are performed in terms of Level-3 BLAS [5].

In a recent work [6], we exposed the poor performance of SYTRD on an ARM big.LITTLE AMP, even if the Level-3 BLAS kernels are optimized to exploit the asymmetry of the architecture. The reason for this behavior is that the memory-bound nature of the Level-2 BLAS, combined with their limited scalability, turn these components into the factor that dominates the efficiency of SYTRD. In [7] we reported a remarkable acceleration for SYTRD achieved by using architecture-aware micro-kernels for the Level-2 BLAS and an asymmetry-aware dynamic schedule of these kernels. In the present paper, we extend that work making the following contributions:

- We discuss the generalization of our techniques developed in [7] for SYTRD to the two remaining TSOR procedures, demonstrating their applicability in the reduction to bidiagonal form implemented in LAPACK routine GEBRD as well as the reduction to Hessenberg form in LAPACK routine GEHRD.
- We propose a performance model that guides the selection of the optimal algorithmic block size and core configuration for the TSOR stage.
- We perform a detailed experimental analysis to illustrate the performance benefits of our architecture- and asymmetry-aware variants of SYTRD, GEBRD and GEHRD on an ARMv7 big.LITTLE architecture.

Overall, we believe that the approach applied to optimize the routines for the TSOR to condensed forms on the target ARMv7 SoC presented in this work carries over to other asymmetric and heterogeneous architectures, including hybrid CPU-GPU systems, as well as multisocket/multicore servers where distinct CPUs/cores operate at different frequencies.

The rest of the paper is organized as follows. In Section 2 we introduce the reduction to condensed forms for the solution of dense eigenvalue and singular-value problems. Section 3 presents the target AMP and the implementation of Level-2 and Level-3 BLAS on the AMP. Section 4 describes the keys towards performance optimization of TSOR routines. Section 5 presents a performance model that is leveraged to determine the optimal algorithmic block size. Finally, we present the experimental results in Section 6, and we draw some conclusions in Section 7.

2. Reduction to Condensed Forms

Given a square matrix A, of order n, the associated eigenvalue problem is formally defined by

$$AX = X\Lambda, \tag{1}$$

where the $n \times n$ diagonal matrix $\Lambda = \mathrm{diag}(\lambda_1, \lambda_2, \ldots, \lambda_n)$ contains the eigenvalues of A, and the columns of the $n \times n$ matrix X contain the corresponding eigenvectors [1]. The singular value decomposition (SVD) of an $m \times n$ matrix A is defined as

$$A = U\Sigma V^T, \tag{2}$$

where $\Sigma = \mathrm{diag}(\sigma_1, \sigma_2, \ldots, \sigma_r)$ is a square matrix of order $r = \min(m, n)$ that contains the singular values of A in decreasing order of magnitude (i.e., $\sigma_i \geq \sigma_{i+1}$); and U, V^T, of respective dimensions $m \times r$ and $r \times n$, are orthogonal and their columns comprise the left and right singular vectors of the matrix.

The routines in LAPACK for the solution of (symmetric and general) eigenproblems as well as the computation of the singular values tackle dense instances of these problems by first reducing A to a condensed matrix[1] C, of dimension $m \times n$ (with $m = n$ for eigenproblems), via a collection of Householder (orthogonal) reflectors [1]. For performance reasons, at each iteration

[1] There exists a multi-stage approach that performs the TSOR in two or more steps, by first reducing A to a band matrix and then successively refining this to the sought-after condensed form [8]. Nevertheless, we will not consider this alternative approach as it often requires a higher number of flops.

of these TSOR procedures, several orthogonal reflectors are aggregated into a single block reflector, which is then applied via calls to efficient Level-3 BLAS. We next describe this process in some detail.

Let us denote the algorithmic block size as b and, for simplicity, assume hereafter that m, n are both integer multiples of b. Consider that we have progressed up to an iteration $j \in \{1, 2, \ldots, \min(m,n)/b\}$, applying the necessary transformations (from the left and right) to the matrix in order to obtain:

$$A \quad \rightarrow \quad \left(\begin{array}{c|c|c} C_{00} & C_{01} & 0/C_{02} \\ \hline C_{10} & A_{11} & A_{12} \\ \hline 0 & A_{21} & A_{22} \end{array}\right),$$

where C_{00} is $(j-1)b \times (j-1)b$; A_{11} is $b \times b$; and the blocks C_{00}, C_{10}, C_{01} (and C_{02}), contain the corresponding entries of the sought-after condensed form C. The following operations are then computed during the current iteration of the TSOR routines SYTRD, GEHRD and GEBRD:

(a) PANEL FACTORIZATION (PF): The "current" column-panel $\left(\begin{array}{c} A_{11} \\ \hline A_{21} \end{array}\right)$ and row-panel $(A_{11} \mid A_{12})$ are reduced to the target condensed form using a sequence of orthogonal transformations. Simultaneously, these transformations are aggregated in the form of matrices V, X, both of dimension $m - jb \times b$, and U, Y, both of size $n - jb \times b$, such that the application of these transformations yields

$$\left(\begin{array}{c|c|c} C_{00} & C_{01} & 0/C_{02} \\ \hline C_{10} & C_{11} & C_{12} \\ \hline 0 & C_{21} & A_{22} - VY^T - XU^T \end{array}\right),$$

implying that, upon completion of this operation, the computation of the condensed form has progressed by b columns/rows.

(b) TRAILING UPDATE (TU): The submatrix A_{22} is updated as $A_{22} := A_{22} - VY^T - XU^T$.

This generic TSOR procedure implements a blocked algorithm that processes the $m \times n$ matrix A, from top-left to bottom-right, in blocks of b-column/row panels starting at columns/rows $\hat{\jmath} = (j-1)b = 0,\ b,\ 2b, \ldots$. The bulk of the computation in PF corresponds to the formation of matrices X, Y (U, V are obtained as part of the panel factorization). In particular, for each reduced column in the panel, this may require several matrix-vector multiplications.

This generic TSOR procedure requires some specialization depending on the type of condensed form to be computed:

- For SYTRD, $m = n$, A is symmetric, $X = Y$, $U = V$ and, in order to exploit the symmetry, only the lower (or upper) half of A_{22} is updated in TU. Taking into account these considerations, the computation in PF involves several small general matrix-vector products (GEMV) and a large symmetric matrix-vector product (SYMV). Furthermore, the update in TU can be performed via a single call to the Level-3 BLAS kernel for the symmetric rank-$2k$ update (SYR2K). The overall cost of routine SYTRD is $4n^3/3$ flops[2], with $2n^3/3$ flops performed via calls to SYR2K, and the rest corresponding to the Level-2 BLAS GEMV and SYMV.
- The reduction to bidiagonal form via GEBRD can be re-organized to reduce the computational cost in case $m \gg n$ (or vice-versa), but the previous procedure is the preferred choice if $m \approx n$. We will focus hereafter in the "squarish" case. For GEBRD, the effect of the Level-2 BLAS is more prominent. The total cost of this reduction, $8n^3/3$ flops, is split into $2n^3/3$ flops performed in terms of the Level-3 BLAS for the general matrix-matrix multiplication (GEMM) and $2n^3$ flops as calls to GEMV.
- The reduction to Hessenberg form via GEHRD slightly differs from the generic TSOR procedure in the specific blocks that are updated (and annihilated) at each iteration [9]. The cost of this reduction is $10n^3/3$ flops, with 20% cast as different types of Level-2 BLAS matrix-vector products and the remaining 80% in efficient Level-3 BLAS [9].

3. Asymmetry-Aware BLAS for AMPs

In this section, we first describe the target AMP, and then we briefly review the implementation of the Level-3 and Level-2 BLAS tuned for this type of architectures [6, 7].

3.1. Target architecture

All the experimentation was carried out using the heterogeneous multiprocessing device ODROID-XU4 furnished with a Samsung Exynos 5422 SoC. This AMP comprises an ARM Cortex-A15 quad-core processing cluster

[2] In general, we neglect the lower order terms in the cost expressions. Furthermore, we assume real arithmetic.

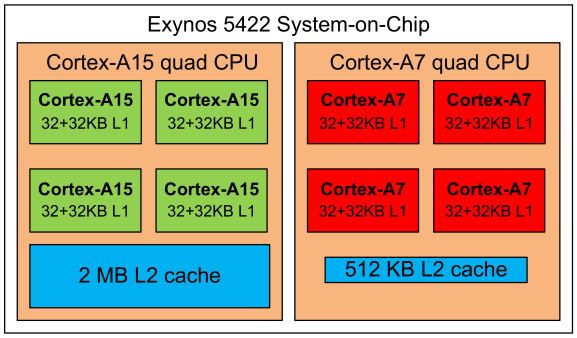

Figure 1: Exynos 5422 block diagram.

(big) plus an ARM Cortex-A7 quad-core processing cluster (LITTLE), both implementing the ARMv7 micro-architecture. Each Cortex core has its own private 32-Kbyte L1 (data) cache. The four ARM Cortex-A15 cores share a 2-Mbyte L2 cache, and the four ARM Cortex-A7 cores share a smaller 512-Kbyte L2 cache; see Figure 1. In addition, the two clusters access a common 2-Gbyte DDR3 RAM. The experiments were performed with the Cortex-A7 cores operating at 1.4 GHz and the Cortex-A15 at 1.5 GHz, using real single-precision IEEE arithmetic. The following analysis and results can be easily adapted to other AMPs, datatypes, and precision.

All our experiments employ the sequential Level-1 kernels from BLIS (version 0.1.8), in combination with the multi-threaded asymmetry-aware instances of the Level-3 and Level-2 kernels introduced in [6, 7].

3.2. Level-3 BLAS for AMPs

All Level-3 BLIS, including GEMM and SYR2K, follow the path pioneered by GotoBLAS to organize the routine as three nested loops around two packing routines and a macro-kernel; see Loops 1–3 in Figure 2, corresponding to the BLIS implementation of the GEMM $\hat{C} += \hat{A} \cdot \hat{B}$, with $\hat{C}$, $\hat{A}$ and $\hat{B}$ of dimensions $m \times n$, $m \times k$ and $k \times n$, respectively. BLIS internally decomposes the macro-kernel into two additional loops around a micro-kernel that, in turn, is implemented as a loop around a rank-1 update (Loops 4–5 in Figure 2). The micro-kernel is usually encoded in assembly, or in C enhanced with vector intrinsics, and is responsible for the actual computations. The packing routines orchestrate the data transfers between consecutive levels of the cache memory hierarchy. In most architectures, m_r, n_r are in the range 4–16; m_c, k_c are in the order of a few hundreds; and n_c can be up

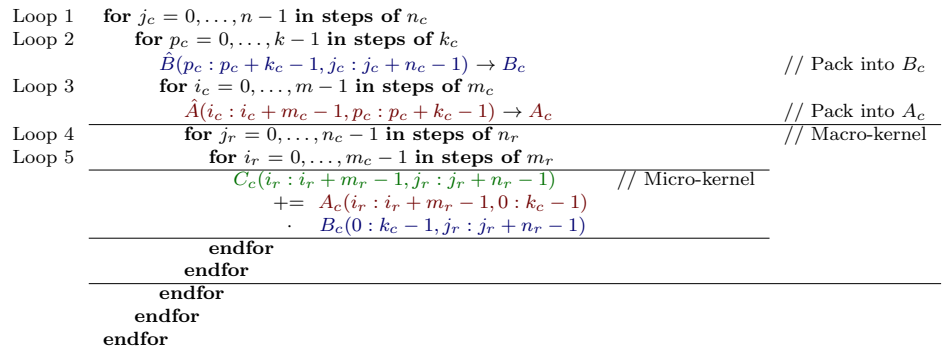

Figure 2: High performance implementation of GEMM in BLIS. In the code, $C_c \equiv \hat{C}(i_c : i_c + m_c - 1, j_c : j_c + n_c - 1)$ is just a notation artifact, introduced to ease the presentation of the algorithm, while A_c, B_c correspond to actual buffers that are involved in data copies.

	m_r	n_r	m_c	k_c	n_c
ARM Cortex-A15	4	4	400	368	4,096
ARM Cortex-A7	4	4	88	368	4,096

Table 1: Parameters for optimal performance of the Level-3 kernels in BLIS on the ARMv7 big.LITTLE embedded in the Exynos 5422 SoC using real single-precision IEEE arithmetic.

to a few thousands [10, 11]. The parameters that optimize performance for the ARM Cortex-A15 and Cortex-A7 are displayed in Table 1. These values were determined experimentally, as part of a separate study. A couple of observations are worth to be pointed out. First, the same value of k_c optimizes performance for both types of ARMv7 cores. Second, for this SoC with no L3 cache, close-to-optimal performance was attained using smaller values for n_c (in the range of 1,000-2,000).

An asymmetry-aware parallelization of the Level-3 BLAS was presented in [6]. That work leverages dynamic scheduling in order to distribute the iteration space of Loop 3 between the two types of clusters proportionally to their performance. Internally, a static schedule is applied to partition the iteration space of Loop 4 among the homogeneous cores of the same cluster; see [6] for details.

3.3. Level-2 BLAS for AMPs

The implementation of the Level-2 BLIS kernels follows a general structure that we illustrate in this subsection. For this purpose, we will leverage

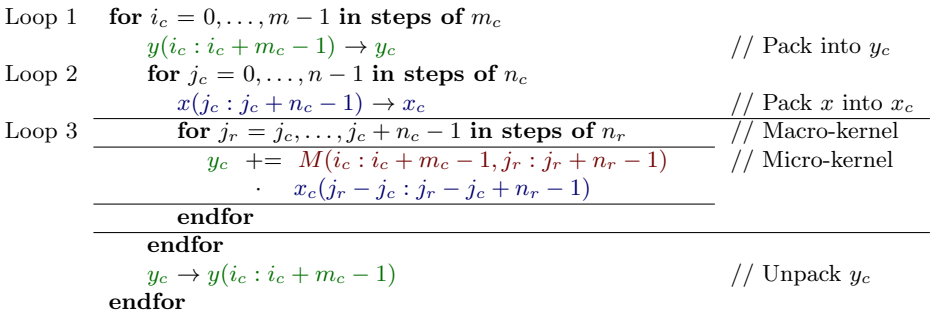

Figure 3: High performance implementation of the GEMV kernel in BLIS. In the code, x_c, y_c are buffers involved in data copies in case x, y are stored with a nonunit stride. Otherwise, they simply refer to the corresponding entries of the original vectors.

the general matrix-vector product GEMV $y += M \cdot x$, with the matrix M of dimension $m \times n$, and y, x vectors with m, n entries, respectively. This kernel is implemented in BLIS as two loops (see Loops 1 and 2 in Figure 3) around two packing routines and a macro-kernel. The GEMV macro-kernel contains an additional loop (Loop 3 in Figure 3) around a micro-kernel that casts each update as a fused vector-vector multiply-add [10]. The fusion factor is 4 and depends on the width of the SIMD NEON intrinsics. The packing routines in the GEMV kernel copy the contents of y, x into contiguous buffers y_c, x_c, and unpack y_c into the result vector y (if these vectors were stored with a nonunit stride). No packing is performed on M since there is no reuse in the BLIS implementation of the general matrix-vector product.

In [7] we developed micro-kernels for the ARM Cortex-A7 and Cortex-A15 cores that exploit the NEON units in these architectures, employing software prefetching and SIMD instructions. There, we also proposed a solution to parallelize both kernels, on the AMP targeted in our work, that extracts the concurrency from Loop 1 via an OpenMP construct that dynamically distributes its iteration space among the threads/cores.

One important theoretical advantage of selecting a dynamic schedule is that the workload is automatically adjusted to the performance capabilities of the two different types of cores in the ARM big.LITTLE AMP. Furthermore, by using distinct cache-aware values of m_c for the Cortex-A15 and the Cortex-A7, the kernels can take advantage of the cache memory hierarchy specific to each type of core; see [7] for details. The parameters that optimize performance for the ARM Cortex-A15 and Cortex-A7 are displayed in Table 2.

	n_r	m_c	n_c
ARM Cortex-A15	4	832	2,560
ARM Cortex-A7	4	144	2,560

Table 2: Parameters for optimal performance of the Level-2 kernels in BLIS on the ARMv7 big.LITTLE embedded in the Exynos 5422 SoC using real single-precision IEEE arithmetic.

An important issue that explains the poor scalability of the Level-2 BLAS is the low ratio between the number of flops and memory accesses, which turns this type of operations into memory-bound kernels that, on current architectures, proceed at the speed dictated by the bandwidth of the memory layer where M is stored. As a consequence, the performance that can be achieved by any Level-2 BLAS greatly depends on the memory bandwidth of the target platform. An important insight gained from the experimental evaluation in [7] is that the Level-2 BLAS kenels hardly scale when increasing the number of big cores, but they do scale for the LITTLE cores. The reason is that a single big core almost saturates the memory bandwidth of the Cortex-A15 cluster so that minor performance increments can be expected by adding more cores of this type. In contrast, the memory bandwidth for the LITTLE cluster is enough to feed all four LITTLE cores.

4. General Optimization of the TSOR Routines

There are four optimization keys that have to be addressed to ensure high performance for the execution of the TSOR routines on the target AMP:

- Development of tuned micro-kernels for the Level-2 and Level-3 BLAS and each type of core.
- Asymmetry-aware parallelization of the Level-2 and Level-3 BLAS.
- Selection of the algorithmic block size for the TSOR procedure.
- Configuration of the number and type of cores to utilize for each type of Level-2 and Level-3 BLAS kernel invoked from the TSOR procedures.

The first two factors were briefly discussed in Section 3, and in more detail in the references therein. This section offers a general evaluation of the impact of the last two on performance.

4.1. The practical role of the algorithmic block size

The algorithmic block size b selected for the TSOR routines has an important performance effect that has to be put into perspective. In order to

illustrate this, the next experiment shows the impact of the block size on the global performance, measured in GFLOPS (billions of flops per second). For simplicity, we run this experiment using a single Cortex-A15 core. Figure 4 reports a performance gap between the lowest and highest GFLOPS rates of about 0.5 GFLOPS for the smallest problem size on the three routines. As the problem dimension is increased, the fluctuation narrows and, for the largest problem, it is around 0.25 GFLOPS for SYTRD and 0.17 GFLOPS for GEBRD. However, on GEHRD the performance gap increases with the problem dimension for small block sizes. This is due to the higher percentage of Level-3 BLAS invoked from this routine which favors the use of larger block sizes. The main conclusion from this preliminary experiment is that the block size exerts a relevant and consistent impact on performance along the problem size range.

To better understand the role of the block size b, Figure 5 profiles the influence of this parameter on the distinct building blocks (i.e., BLAS kernels) appearing in the TSOR routines, exposing some important details:

- SYTRD: The symmetric matrix-vector product (SYMV, green lines in the figure) accounts for a major part of the global execution time, with this fraction of the practical cost growing with the problem size. This implies that an optimization of this particular kernel, via either an architecture-aware implementation or an asymmetry-aware parallelization, can be expected to yield important gains on the performance of the reduction routine. In addition, the execution time of SYMV is basically independent of the block size. Therefore, the optimization of this parameter for SYTRD can be pursued by taking into account only the other two components of the reduction, namely GEMV and SYR2K.

  The execution time of the general matrix-vector products (carried out via GEMV, dark blue lines) grows with the block size, while that of the symmetric rank-$2k$ update (SYR2K, red lines) has the opposite behavior. The reason for these opposite trends lies in that an increase of the block size shifts part of the computational cost of the reduction (in the order of n^2b flops) from the symmetric rank-$2k$ update to the general matrix-vector product. This has a minor impact on the theoretical cost/execution time of the SYR2K kernel, as the volume of computations performed in terms of this type of operations is $2n^3/3$ flops; indeed, the reduction in the execution time of this component is basically due to the use of a larger block size, which delivers a higher GFLOPS rate. However, increasing the amount of flops that are cast in terms of GEMV has a major effect on the practical cost of GEMV,

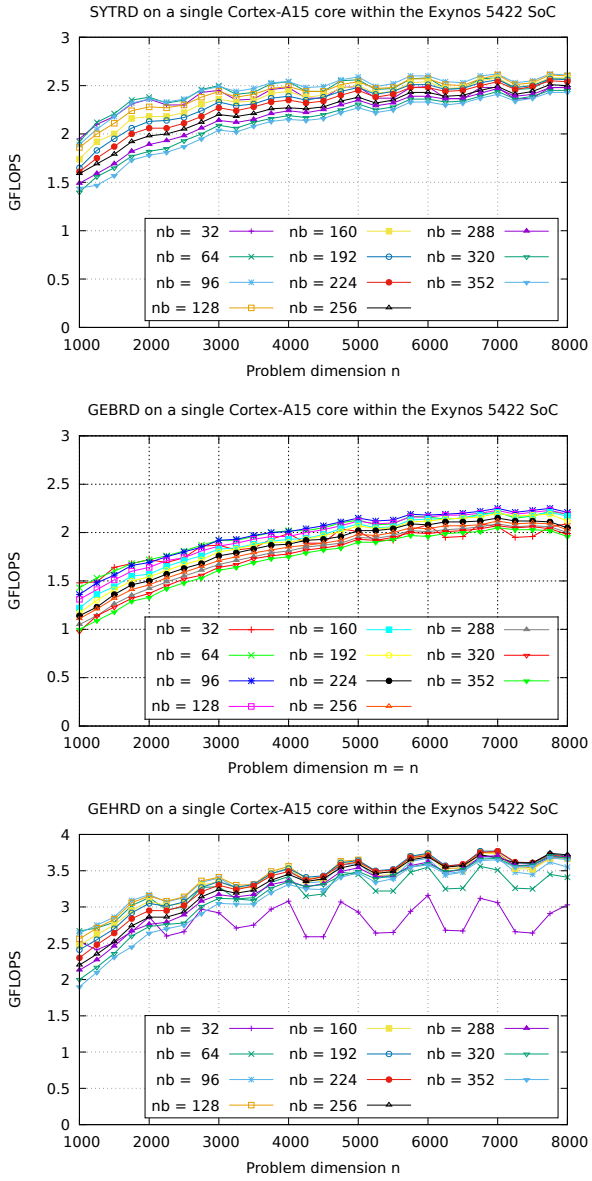

Figure 4: Performance of SYTRD (top), GEBRD (middle) and GEHRD (bottom) on a single Cortex-A15 core within the ARM big.LITTLE AMP embedded in the Exynos 5422 SoC using different algorithmic block sizes.

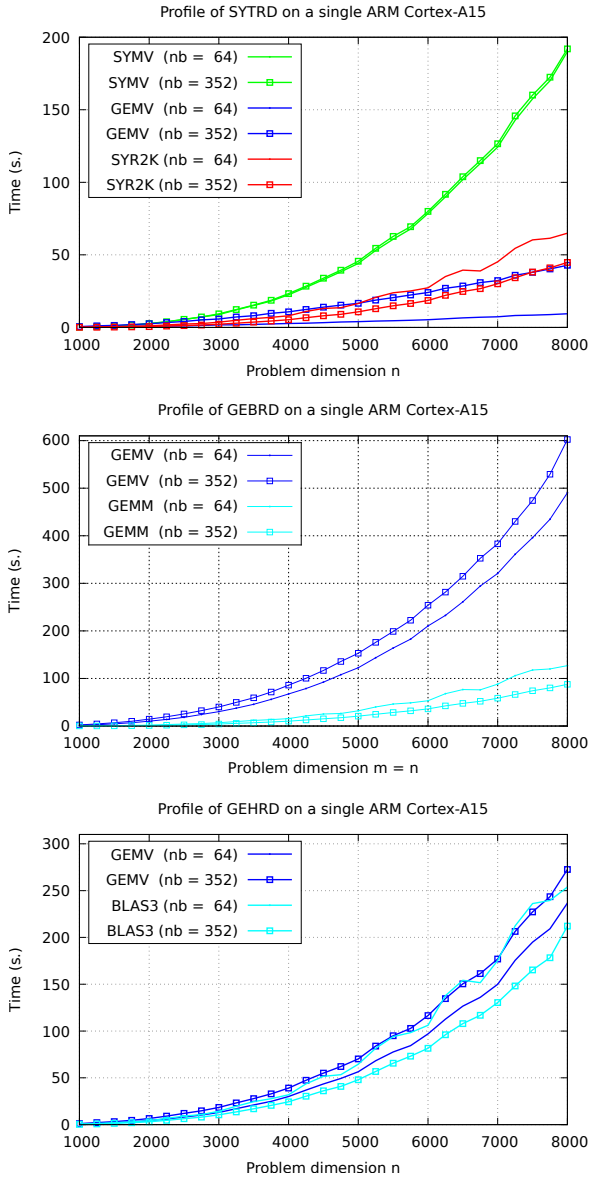

Figure 5: Profile of execution time spent by SYTRD (top), GEBRD (middle) and GEHRD (bottom) on a single ARM Cortex-A15 core embedded in the Exynos 5422 SoC. This experiment sets the block size to 64 and 352, respectively, for comparison.

as this is a memory-bound operation that proceeds at a much lower GFLOPS rate. In other words, although increasing b produces a small raise in the amount of flops that are cast in terms of GEMV (when compared to the total flops of the reduction routine) the practical cost (i.e., execution time) becomes much larger due to the low performance of this kernel.

- GEBRD: The execution time is clearly split into two components: the general matrix-vector products basically found in PF (GEMV, dark blue lines), and (two) general matrix-matrix multiplications for TU (GEMM, light blue lines). Increasing the block size here shifts part of the flops from TU to PF, with an effect on performance similar to that already discussed for SYTRD at the end of the previous item.

- GEHRD: Again, the execution time is mainly split into two components, namely that of GEMV and that of the Level-3 BLAS issued (GEMM and TRMM). The execution time of GEMV (dark blue lines) grows with the block size, while the Level-3 BLAS (light blue lines) has the opposite behaviour. However, since about 80% of the flops are executed in the Level-3 BLAS calls, for this TSOR procedure the execution time corresponding only to the GEMV amounts to about 50% of the total. This favors the use of larger block sizes as the loss in GEMV due to the adoption of a larger block size (dark blue lines) is outweighted by the gains obtained in the Level-3 BLAS when using a larger block size (light blue lines).

4.2. Selection of the core configuration

A complementary factor that dictates the performance of the TSOR procedures, when executed on an AMP, is the number/type of cores (configuration) that are employed for the execution of each building block.

The two plots in the top row of Figure 6 report the performance rate attained by GEMV, using matrix operands of two practical shapes encountered in the TSOR routines. These two graphs reveal that the threshold dimension from which it is more convenient to use a Cortex-A15 core plus the full Cortex-A7 cluster depends on the iteration step (m-dimension) and the algorithmic block size (n-dimension). For small block sizes, large values in the m-dimension favour the use of the Cortex-A15 core plus the full Cortex-A7 cluster. In contrast, for large block sizes, small values in the m-dimension are to be preferred. In addition, the block size dictates the highest sustainable performance observed for GEMV. For small block sizes,

this kernel attains 2.5 GFLOPS due to data re-use in the caches, but this value decreases to only 2 GFLOPS for large block sizes.

The four plots in the bottom two rows of Figure 6 show the results for an analogous experiment using the Level-3 BLAS routines and two matrix shapes that appear during the TSOR routines. The conclusions inferred from this analysis of the Level-3 BLAS is similar to that presented for GEMV. In summary, the point from which it is more beneficial to use the entire SoC or the Cortex-A15 cluster only depends on the iteration step and the block size. However, for the Level-3 BLAS, large block sizes tend to render higher performance, as they allow to select closer-to-optimal loop strides while extracting an ampler level of concurrency within the kernels [7].

To complete the analysis of the main building blocks present in the TSOR routines, Figure 7 shows the performance of the SYMV routine. In contrast with the previous kernels, an optimal configuration of this building block always exploits a Cortex-A15 core plus the full Cortex-A7 cluster.

In summary, the routines for the basic building blocks identify independent work units (blocks of loop iterations) that will be then scheduled to the distinct types of cores by the OpenMP runtime using a *dynamic scheduling strategy*. At this point we remark that *i)* a dynamic scheduling scheme introduces higher overhead than a static scheduling since the work units are generated at runtime and this overhead is more visible for small problem dimensions; and *ii)* dynamic scheduling requires a medium to large number of work items to deliver a fair workload balance especially when the work items are of different size and the cores present distinct computational performances. In addition, the algorithmic block size directly affects the shapes of the matrix operands passed to the BLAS kernels invoked from the TSOR procedures. Furthermore, the experiments in this section illustrate that the block size changes the threshold from which it is more beneficial to use a certain configuration for the execution of a certain building block (kernel). Therefore, the effect of the block size has to be analyzed simultaneously with the core configuration.

5. Modeling the Performance of the TSOR Routines

In [7] we selected the optimal block size for SYTRD and the core configuration for the three building blocks appearing in this reduction (i.e., SYMV, GEMV and SYR2K) by conducting an exhaustive experimental analysis of all possible combinations during the execution of SYTRD. While this is doable, we next propose a more methodical approach to model performance; see also [12, 13, 14, 15]. Here we select the optimal block size for the TSOR

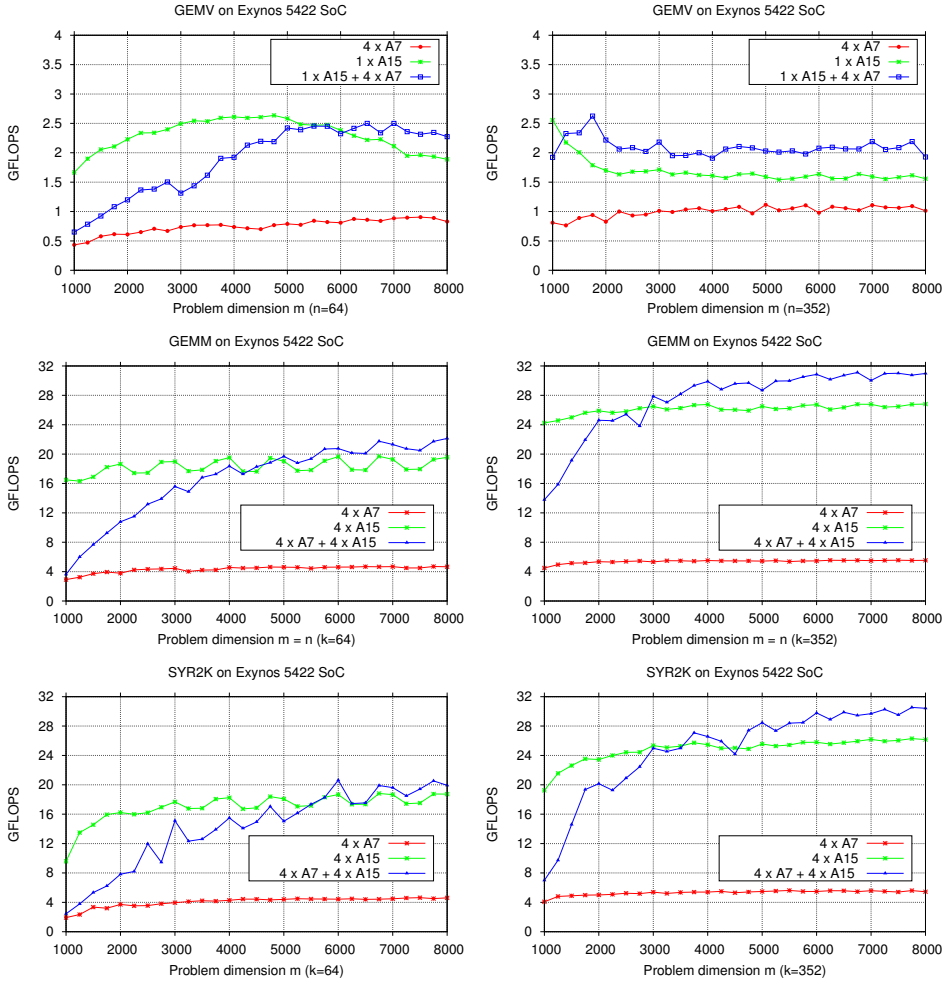

Figure 6: Performance of Level-2 and Level-3 BLAS on the ARMv7 big.LITTLE embedded in the Exynos 5422 SoC.

procedures, based on the experimental performance observed for their building blocks and the theoretical flop count for each type of building block. In order to illustrate this, we employ the specific case of the reduction to tridiagonal form of a symmetric $n \times n$ matrix A via routine SYTRD. The same method carries over to the remaining two TSOR procedures.

At this point, we remind some of the observations from [7], connecting them to the experiments in the previous section:

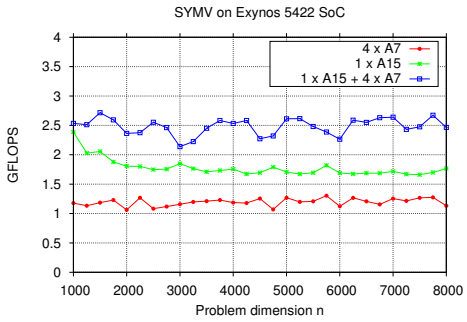

Figure 7: Performance of SYMV on the ARMv7 big.LITTLE embedded in the Exynos 5422 SoC.

- Consider, for simplicity, that n is an integer multiple of the block size: $n = r \cdot b$, for a given integer r. The blocked single-step reduction to tridiagonal form processes the $n \times n$ matrix A, from top-left to bottom-right, in a set of iterations $j \in \{1, 2, \ldots, n/b\}$, in blocks of b-column panels starting at rows/columns $\hat{\jmath} = (j-1)b = 0,\ b,\ 2b, \ldots, (r-1)b$. In general n may not be a multiple of the block size. Then, the last panel receives a special treatment using unblocked code.

- At iteration j, the assembly of V requires b symmetric matrix-vector multiplications, of decreasing dimensions $n-(\hat{\jmath}+1), n-(\hat{\jmath}+2), n-(\hat{\jmath}+3), \ldots, n-(\hat{\jmath}+b)$. Overall, the reduction performs $n-2$ calls to SYMV, involving matrices of dimensions $n-1, n-2, \ldots, 2$. Thus, the block size b has no effect on the number of flops nor the operands' shapes of the sequence of calls to SYMV. We can conclude, hence, that b should exert no impact on the performance of SYTRD. This observation is confirmed by the results in the top plot in Figure 5.

- The experimental analysis in [7] revealed that a close-to-optimal configuration for the parallel execution of SYMV, on the Exynos 522 SoC, employs a single Cortex-A15 core plus the full quad-core Cortex-A7 cluster. Slightly higher performance can be attained by activating a second Cortex-A15 core, but this will come at a non-negligible energy cost, which may be relevant for an energy-efficient architecture. In consequence, we prefer the configuration with a single Cortex-A15 core. We use hand-coded micro-kernels for both types of core architectures, and a dynamic distribution of the iteration space of Loop 1 among the system cores, with cache-aware granularity m_c that depends on the

core type (see Tables 1 and 2).

- The assembly of V at iteration j requires $6 \cdot b$ general matrix-vector multiplications of dimensions that depend on the algorithmic block size. More specifically, the dimensions of GEMV vary (linearly in both dimensions) from $n - (\hat{\jmath} + 1) \times 1$ to $n - (\hat{\jmath} + b) \times b$. As a consequence, the overall number of flops performed by GEMV directly depends on the algorithmic block size of SYTRD, and we can conclude that b plays some role on performance. This is confirmed by the experiment in the top plot in Figure 5.

- The experimental analysis in [7] hinted similar conclusions for GEMV to those exposed for SYMV in the sense that close-to-optimal performance is obtained by using a single Cortex-A15 core plus the full quad-core Cortex-A7 cluster. However, for small problem dimensions, it is more beneficial to use a single Cortex-A15 core.

- At the end of each iteration j, the SYTRD routine invokes the SYR2K kernel to update the trailing submatrix in A of order $n - (\hat{\jmath} + b) + 1$, using two panels of dimension $n - (\hat{\jmath} + b) + 1 \times b$ each ($A_{22} := A_{22} - UV^T - VU^T$). Therefore, increasing the block size b accelerates the decay of the trailing submatrix dimensions as the iteration progresses, but augments the number of columns in the panels. In conclusion, we can expect that b has a certain effect on the performance of the sequence of calls to SYR2K, because it affects the operands' dimensions and shapes. See again the results in the top plot in Figure 5.

- The algorithmic block size directly affects the matrix shapes involved in SYR2K and changes the threshold value for which it is more beneficial to use only the Cortex-A15 cluster or the full SoC. In conclusion, we should employ either the Cortex-A15 cluster or the full SoC when the dimension of A_{22} is smaller or larger than the threshold for a given algorithmic block size.

To sum up, at iteration $j \in \{1, 2, \ldots, n/b\}$, routine SYTRD invokes the following Level-2 and Level-3 BLAS routines:

1. $b-1$ calls to SYMV, each involving a square matrix of order $r-k$, with $r = n - (j-1)b$ and $k = 1, 2, \ldots, b-1$.
2. $6b$ calls to GEMV, each of the b involving a matrix of dimension $(r - k) \times k$, with $k = 1, 2, \ldots, b$.

3. A single call to SYR2K to perform two updates of the form $\hat{C} += \hat{A} \cdot \hat{A}^T$, on a triangular part of a square result matrix $\hat{C}$ of order $s = n - jb$ and $\hat{A}$ of dimension $s \times b$.

Therefore, the total cost of the routine, $4n^3/3$ flops, can be distributed among the three building blocks as follows:

1. SYMV: $\sum_{j=1}^{n/b} \sum_{k=1}^{b-1} 2(r-k)^2 = 2n^3/3$ flops.
2. GEMV: $\sum_{j=1}^{n/b} 12(rb^2/2 - b^3/3) = 3n^2 b$ flops.
3. SYR2K: $\sum_{j=1}^{n/b} 2s^2 b = 2n^3/3$ flops.

Note that the number of calls to SYMV and the dimension of the matrix operand for this kernel are independent of the algorithmic block size. Therefore, this type of kernel does not play a role in the optimization of b, and our target can be simplified to the minimization of the execution time for GEMV and SYR2K only. This can be formulated as:

$$\min_b \ \{T_{gemv} + T_{syr2k}\},$$

where the execution time due to the flops performed via GEMV and SYR2K are given by

$$\begin{aligned} T_{gemv} &= \sum_{j=1}^{n/b} \sum_{k=1}^{b} \frac{12(r-k)k}{G_{gemv}(r-k,k,\mathcal{C})} \quad \text{and} \\ T_{syr2k} &= \sum_{j=1}^{n/b} \frac{2s^2 b}{G_{syr2k}(s,b,\mathcal{C})}, \end{aligned}$$

respectively. In the last expressions, $G_{gemv}(p,q,\mathcal{C})$ and $G_{syr2k}(p,q,\mathcal{C})$ stand for the FLOPS (flops per second) rates delivered by the corresponding routines[3] when operating on a problem of dimension (p,q) using a core configuration $\mathcal{C}$. At this point, we remind that, for GEMV, the optimal configuration employs either a single Cortex-A15 core or 1 Cortex-A15 + 4 Cortex-A7 cores. In contrast, for SYR2K the optimization procedure has to select between 4 Cortex-A15 cores or the full Exynos 5422 SoC; see Figure 6.

This optimization model guides the search for the optimal block size and core configuration for SYTRD using the data for the experimental GFLOPS rates observed for SYR2K and GEMV. As we are only interested in a qualitative comparison of the execution time for different values of b and core configurations, we do not need to perform an exhaustive evaluation of the

[3]In our model we distinguish the FLOPS rates of GEMV for the transposed and non transposed case.

building blocks. Instead, we can select some representative values and interpolate the FLOPS for the missing performance rates. Moreover, we note that the building blocks GEMM and GEMV appear also in the remaining two TSOR procedures, GEHRD and GEBRD. Therefore, we can reuse most of the experimental evaluation of the building blocks to tune the block size and core configuration for all three TSOR routines.

Figure 8 shows the evaluation of the performance determined via the model in comparison with the practical results obtained from an exhaustive execution of SYTRD using different algorithmic block sizes. For each problem dimension, the top plot in that figure reports model-driven estimates of the time increment with respect to the execution time obtained when using the optimal block size for that problem size. Concretely, for the problem of dimension $n = 1,000$, the variation of time is normalized with respect to the execution time using an algorithmic block size $b = 32$ (which corresponds to the optimal value of b for that problem size); for n ranging from 1,250 to 2,500 the results are normalized with respect to the execution time using $b = 64$; and for $n > 2,500$, they are normalized with respect to the execution time using $b = 96$. In order to offer quantitative variations of the execution time, the model should have also taken into account the execution time of SYMV. However, as we are only interested in a qualitative detection of the optimal algorithmic block size, we can simplify the search by neglecting the impact of SYMV in the model. Overall, the model estimates that the optimal block size is either 64 or 96, with the differences between these two algorithmic block sizes being below 1%. In addition, the model exposes that the execution time grows with the algorithmic block size.

The model-driven search of the optimal algorithmic block size is validated with the exhaustive evaluation of the performance of SYTRD in the bottom plot in Figure 8. The practical results confirm that the actual algorithmic block sizes yielding the highest performance are also 64 and 96, with the performance declining when the algorithmic size exceeds the largest of these values.

The previous experiment shows that the model can be used to perform a search of the optimal algorithmic block size without testing the factorization itself. However, as the model predicts performance differences below 1% between the two close-to-optimal algorithmic block sizes, (though similar to those observed in practice,) this search methodology may introduce small deviations in the value selected for b. Table 3 quantifies the impact of a suboptimal choice of b, comparing the execution time for executions that employ the algorithmic block size predicted by the model against those with the optimal algorithmic block size obtained from the experimentation. The

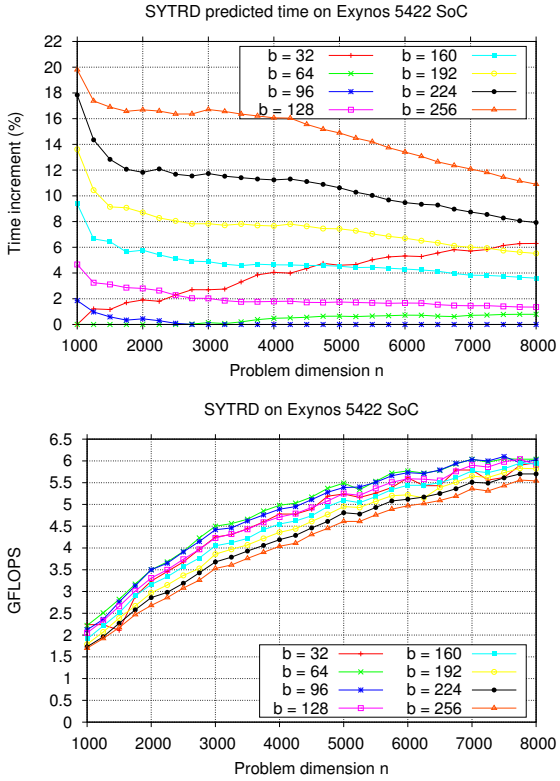

Figure 8: Model-driven estimation of the relative execution time (top) and actual performance (bottom).

results in the table reveal that the relative error is consistently below 2% (except in one case), being smaller than 1% for most problem dimensions.

6. Performance of the Tuned TSOR routines

This section demonstrates the performance benefits of a tuned selection of the block size and core configuration, (together with the integration of architecture-aware microkernels and asymmetry-aware parallel version of the building blocks) for the TSOR routines SYTRD, GEBRD and GEHRD.

During the execution of these routines, we dynamically adjust the number/type of cores independently for the main building blocks in order to tune the performance depending on the dimensions of the operands that are involved in each call to a building block. This dynamic optimzation was

Problem dimension	Optimal (b) Model	Real	Difference (%)	Problem dimension	Optimal (b) Model	Real	Difference (%)
1,000	32	32	–	4,750	96	64	1.66
1,250	64	64	–	5,000	96	64	1.82
1,500	64	64	–	5,250	96	96	–
1,750	64	64	–	5,500	96	64	0.36
2,000	64	64	–	5,750	96	64	1.05
2,250	64	64	–	6,000	96	64	0.69
2,500	64	64	–	6,250	96	64	0.17
2,750	96	64	1.89	6,500	96	96	–
3,000	96	64	1.78	6,750	96	96	–
3,250	96	64	2.19	7,000	96	64	0.16
3,500	96	64	0.86	7,250	96	96	–
3,750	96	64	1.85	7,500	96	96	–
4,000	96	64	1.81	7,750	96	64	1.32
4,250	96	64	1.59	8,000	96	64	0.33
4,500	96	64	1.16	Average			0.71

Table 3: Relative differences of time for SYTRD between executions using the optimal block size determined by the model and the real optimal value detected via exhaustive experimental tests.

applied to GEMV, SYMV and SYR2K for SYTRD; GEMV and GEMM for GEBRD; and GEMV, GEMM and TRMM for GEHRD.

Figure 9 illustrates the performance of the three TSOR routines, using the model-driven optimal algorithmic block size for SYTRD (top, $b = 64$), GEBRD (middle, $b = 96$) and GEHRD (bottom, $b = 128$). The architecture-aware microkernels for the Level-2 BLAS kernels and the asymmetry-aware parallelizations of the Level-2/3 kernels correspond to the implementations presented in [7] and [6], respectively. The plots include a configuration with no optimizations applied to the Level-2 BLAS (labeled as "Initial") as well as one where all optimizations are present (labeled as "Asymmetry-aware"). Additionally, for comparison purposes these plots include four additional reference configurations:

- 4 x A15: execution on the Cortex-A15 cluster only (4 threads).
- 4 x A7: execution on the Cortex-A7 cluster only (4 threads).
- Ideal - 4: theoretical performance rate obtained by adding the GFLOPS rates of the isolated Cortex-A15 cluster and the isolated Cortex-A7 cluster.
- Ideal - 1: theoretical performance rate obtained by adding the GFLOPS rate of a single Cortex-A15 core multiplied by 4 (number of cores in the cluster) plus that of a single Cortex-A7 multiplied by 4.

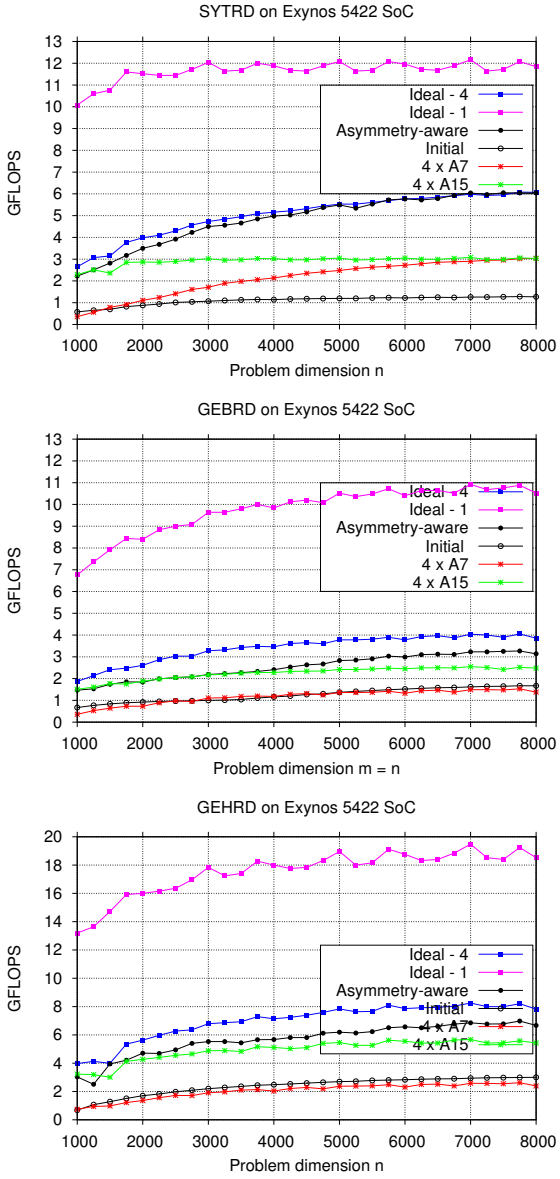

Figure 9: Performance of SYTRD, GEBRD and GEHRD on the ARMv7 big.LITTLE embedded in the Exynos 5422 SoC. This experiment sets the block size to $b = 64$ for SYTRD, $b = 96$ for GEBRD and $b = 128$ for GEHRD.

Let us discuss in detail the results for SYTRD (top plot in Figure 9). The use of the Cortex-A15 cluster only shows a performance rate that is almost flat, close to 3 GFLOPS. The reason is that the Level-2 BLAS, dominates the execution time of the routine, and adding more than one Cortex-A15 thread does not contribute any performance benefit. In contrast, the trend observed for the line that employs the Cortex-A7 cluster only shows an asymptotic performance that is close to that of the Cortex-A15 cluster, as the Level-2 BLAS do scale with the problem dimension for this type of cores. These results can be related back to the analysis of the building blocks SYMV, GEMV and SYR2K in Section 4.

The asymmetry-aware configuration shows a consistent performance advantage over its homogeneous (i.e., symmetric or single-cluster) counterparts as the former takes advantage of the computational power of the Cortex-A15 cores for the execution of the Level-3 BLAS SYR2K and the scalability of Level-2 BLAS SYMV/GEMV on the Cortex-A7 cluster. Concretely, for the largest problem size the speed-up of this solution grows to be above 2 with respect to the execution using any of the two clusters in isolation. In addition, the asymmetry-aware configuration benefits from the multi-threaded Level-2 BLAS and the optimized micro-kernels to deliver a performance rate that is up to 6× higher than the initial configuration. Focusing on the ideal (theoretical) configurations, our solution attains a performance rate that lies close to that of the Ideal-4 case, showing a fair distribution of the workload between the two clusters and no significant performance leaks. A less pleasant scenario appears in the comparison against the Ideal-1 case. This is explained by the actual lack of scalability of the Level-2 BLAS when executed on the Cortex-A15, in contrast with the unrealistic assumption of perfect scalability for the Ideal-1 line. In more detail, we note that the line labeled as Ideal-1 corresponds to an ideallistic (and quite irreal) performance rate that results from aggregating the practical performance of a single Cortex-A15 multiplied by 4 plus the practical performance of a single Cortex-A7 core multiplied by 4. Therefore, it is as if the 8 cores where operating in isolation, without memory access conflicts. The main reason that the practical performance of our parallel implementation the TSOR routines is far from the Ideal-1 curve is that the CPU-memory bandwidth rapidly saturates (in particular for the Cortex-A15 cores) due to the strong memory-bound nature of the level-2 BLAS kernels included in the TSOR routines.

The major cause for the acceleration of SYTRD, which allows to narrow the gap between the performances of the asymmetry-aware configuration and Ideal-4, is the large amount of symmetric matrix-vector products

(SYMV) which are, in turn, large and independent of the block size. Large matrix-vector products result in appealing opportunities to exploit a parallel configuration. Unfortunately, this is not the case for GEBRD. The middle plot in Figure 9 shows that, for this TSOR routine, the asymmetry-aware configuration steadily approaches but does not reach the performance of the Ideal-4 curve. Although there exists a large amount of calls to GEMV in GEBRD, and the aggregated time spent on this kernel is considerably large compared with the total execution time, only a small fraction of the GEMV kernels involve a matrix operand that is large enough to benefit from a parallel configuration. This is also the case even for large problem sizes, as the matrix operand passed to the matrix-vector multiplications decreases in size at each iteration step. Thus, using Ideal-4 as a theoretical reference function here is, to a certain extent, unrealistic. The bottom plot in Figure 9 presents the results obtained for GEHRD. Again, the asymmetry-aware configuration approaches but does not reach the performance of the Ideal-4 curve for similar reasons to those commented above for GEBRD.

To close this section, we point out that employing a parallel configuration for the execution of small-size matrix-vector products is counterproductive. To avoid this negative effect, we designed an adaptive strategy that, according to problem dimension and block size, selects the best core configuration runtime. This strategy ensures that the performance of the asymmetry-aware configuration matches that attained with the Cortex-A15 cluster for small- to medium-size problems (for example, $n \leq 3500$ for SYTRD). Compared with that, for larger problems, our asymmetry-aware algorithms add the Cortex-A7 cluster to the computation, raising the GFLOPS rate by a factor that is close to 30%.

7. Conclusions

We have presented architecture- and asymmetry-aware realizations of the TSOR procedures for the solution of general and symmetric dense eigenvalue problems as well as singular-value problems for ARM big.LITTLE multicore architectures. Our experiments with tuned versions of these routines, specifically optimized for the ARM Cortex-A15 and Cortex-A7 cores present in the ODROID-XU4, show a significant acceleration of the execution time compared with a simple execution of LAPACK's legacy codes for this purpose.

Our theoretical and practical analyses reveal the large impact of the Level-2 BLAS kernels on the performance of the TSOR procedures and the critical roles of the algorithmic block size and the core configuration.

Concretely, the block size has to be finely adjusted to distribute the workload between the Level-2 and Level-3 kernels, taking into account that the memory-bound nature of the former often places this type of operations on the critical path of the algorithm. In addition, an optimal execution also depends on the number and type of cores employed for each type and dimension of the building blocks, with these two parameters determining when it becomes convenient to add the LITTLE cores to the execution.

This research opens a number of interesting questions. As part of future work, we plan to investigate the trade-off between performance and energy consumption, possibly sacrifizing the former in favour of attaining a lower energy footprint. We recognize that this may be an appealing goal on an energy-efficient architecture such as an ARM big.LITTLE. In the future, we also plan to analyze in more detail the relationship between the performance parameters and the architecture (cache size/levels/organization, memory bandwidth, etc.) as well as explore the potential extension of this work to clusters (distributed-memory architectures) consisting of heterogenous nodes.

Acknowledgements

The researchers from Universidad Jaume I were supported by project TIN2014-53495-R of MINECO and FEDER, and the FPU program of MECD. The researcher from Universitat Politècnica de València was supported by the Generalitat Valenciana PROMETEOII/2014/003. The researcher from Universitat Politècnica de Catalunya was supported by projects TIN2015-65316-P from the Spanish Ministry of Education and 2014 SGR 1051 from the Generalitat de Catalunya, Dep. d'Innovació, Universitats i Empresa.

References

[1] G. H. Golub, C. F. V. Loan, Matrix Computations, 3rd Edition, The Johns Hopkins University Press, Baltimore, 1996.

[2] R. M. Martin, Electronic Structure: Basic Theory and Practical Methods, Cambridge University Press, Cambridge, UK, 2008.

[3] E. Anderson, Z. Bai, J. Demmel, J. E. Dongarra, J. DuCroz, A. Greenbaum, S. Hammarling, A. E. McKenney, S. Ostrouchov, D. Sorensen, LAPACK Users' Guide, SIAM, Philadelphia, 1992.

[4] J. J. Dongarra, J. Du Croz, S. Hammarling, R. J. Hanson, An extended set of FORTRAN basic linear algebra subprograms, ACM Trans. Math. Soft. 14 (1) (1988) 1–17.

[5] J. J. Dongarra, J. Du Croz, S. Hammarling, I. Duff, A set of level 3 basic linear algebra subprograms, ACM Trans. Math. Soft. 16 (1) (1990) 1–17.

[6] S. Catalán, J. R. Herrero, F. D. Igual, R. Rodríguez-Sánchez, E. S. Quintana-Ortí, C. Adeniyi-Jones, Multi-threaded dense linear algebra libraries for low-power asymmetric multi-core processors, Journal of Computational Science (2016) – doi:http://dx.doi.org/10.1016/j.jocs.2016.10.020.
URL http://www.sciencedirect.com/science/article/pii/S1877750316302812

[7] P. Alonso, S. Catalán, J. R. Herrero, E. S. Quintana-Ortí, R. Rodríguez-Sánchez, Reduction to tridiagonal form for symmetric eigenproblems on asymmetric multicore processors, in: Proc. 8th Int. Workshop on Programming Models and Applications for Multicores and Manycores, PMAM'17, ACM, New York, NY, USA, 2017, pp. 39–47. doi:10.1145/3026937.3026938.
URL http://doi.acm.org/10.1145/3026937.3026938

[8] C. H. Bischof, B. Lang, X. Sun, A framework for symmetric band reduction, ACM Trans. Math. Soft. 26 (4) (2000) 581–601.

[9] G. Quintana-Ortí, R. van de Geijn, Improving the performance of reduction to Hessenberg form, ACM Trans. Math. Softw. 32 (2) (2006) 180–194. doi:10.1145/1141885.1141887.
URL http://doi.acm.org/10.1145/1141885.1141887

[10] F. G. Van Zee, R. A. van de Geijn, BLIS: A framework for rapidly instantiating BLAS functionality, ACM Trans. Math. Softw. 41 (3) (2015) 14:1–14:33.

[11] F. G. V. Zee, T. M. Smith, B. Marker, T. M. Low, R. A. V. D. Geijn, F. D. Igual, M. Smelyanskiy, X. Zhang, M. Kistler, V. Austel, J. A. Gunnels, L. Killough, The BLIS framework: Experiments in portability, ACM Trans. Math. Soft. 42 (2) (2016) 12:1–12:19.

[12] E. Peise, P. Bientinesi, Performance modeling for dense linear algebra, in: 2012 SC Companion: High Performance Computing, Networking Storage and Analysis, 2012, pp. 406–416. doi:10.1109/SC.Companion.2012.60.

[13] P. Alonso, S. Catalán, F. D. Igual, R. Mayo, R. Rodríguez-Sánchez, E. S. Quintana-Ortí, Time and energy modeling of highperformance level-3 BLAS on x86 architectures, Simulation Modelling Practice and Theory 55 (2015) 77 – 94. doi:http://dx.doi.org/10.1016/j.simpat.2015.04.003.
URL http://www.sciencedirect.com/science/article/pii/S1569190X15000635

[14] E. Peise, P. Bientinesi, A Study on the Influence of Caching: Sequences of Dense Linear Algebra Kernels, Springer International Publishing, Cham, 2015, pp. 245–258. doi:10.1007/978-3-319-17353-5_21.
URL http://dx.doi.org/10.1007/978-3-319-17353-5_21

[15] E. Peise, D. Fabregat-Traver, P. Bientinesi, On the Performance Prediction of BLAS-based Tensor Contractions, Springer International Publishing, Cham, 2015, pp. 193–212. doi:10.1007/978-3-319-17248-4_10.
URL http://dx.doi.org/10.1007/978-3-319-17248-4_10