Parallel Computing: Technology Trends
I. Foster et al. (Eds.)

© 2020 The authors and IOS Press.
This article is published online with Open Access by IOS Press and distributed under the terms of the Creative Commons Attribution Non-Commercial License 4.0 (CC BY-NC 4.0).
doi:10.3233/APC200036

On the Autotuning of Task-Based Numerical Libraries for Heterogeneous Architectures

Emmanuel AGULLO [a], Jesús CÁMARA [b,1], Javier CUENCA [b] and Domingo GIMÉNEZ [c]

[a] *HiePACS Team, Inria Bordeaux Sud Ouest, France*
[b] *Department of Engineering and Technology of Computers, University of Murcia, Spain*
[c] *Department of Computing and Systems, University of Murcia, Spain*

Abstract. A roadmap for autotuning task-based numerical libraries is presented. Carefully chosen experiments are carried out when the numerical library is being installed to assess its performance. Real and simulated executions are considered to optimize the routine. The discussion is illustrated with a task-based tile Cholesky factorization, and the aim is to find the optimum tile size for any problem size, using the Chameleon numerical linear algebra package on top of the StarPU runtime system and also with the SimGrid simulator. The study shows that combining a smart exploration strategy of the search space with both real and simulated executions results in a fast, reliable autotuning process.

Keywords. autotuning, linear algebra, task-based programming, heterogeneous computing, simulation

1. Introduction

The complexity of modern computers makes the design of high performance numerical libraries extremely challenging. Task-based programming paradigms have been proved to alleviate the exercise, as part of the burden is delegated to a third party software, commonly referred to as a runtime system. Nonetheless, the resulting libraries are often left with one or more parameters to be carefully set up in order to achieve high performance. This work describes an approach on how to use autotuning techniques to select the best values for some algorithmic parameters of the linear algebra routines of these kinds of libraries.

The proposed approach is applied to routines of Chameleon [1]. The computational kernels of the library are used as building blocks for higher-level routines designed for heterogeneous platforms composed of multicore CPUs with one or more GPUs. This dense linear algebra library is derived from PLASMA [2] and internally uses StarPU [3], a runtime system which enables us to express parallelism through sequential-like code

[1]Corresponding Author: J. Cámara, Department of Engineering and Technology of Computers, University of Murcia, 30100 Espinardo, Murcia, Spain; E-mail: jcamara@um.es.

and which schedules the different tasks over the hybrid processing units. These tasks are executed by using optimized implementations of linear algebra libraries, such as Intel MKL [4] for multicore CPU and MAGMA [5] or cuBLAS [6] for GPU. In previous works, several frameworks have been developed which focus on how to optimize linear algebra kernels on heterogeneous platforms [7,8]. Other approaches are proposed to predict the performance of a dynamic task-based runtime system for heterogeneous multicore architectures [9]. In contrast with those previous works, we propose the application of tuning strategies to obtain the best value of the algorithmic parameters of the routines for an efficient use of the hybrid components in the computational node. The application of these strategies is illustrated with the Cholesky routine, a fundamental and representative linear algebra algorithm, with the focus on the selection of the value for the tile size.

The rest of the paper is organized as follows. Section 2 introduces the Cholesky routine of Chameleon and how it is executed by using the StarPU runtime system. Section 3 describes the training strategies proposed for selecting the best values for the tile size. Experimental results are shown in Section 4 for a heterogeneous platform. Possible extensions of the methodology are discussed in Section 5. Section 6 concludes the paper.

2. Cholesky Routine of Chameleon

The Cholesky factorization (or Cholesky decomposition) of an $n \times n$ real symmetric positive definite matrix A has the form $A = LL^T$, where L is an $n \times n$ real lower triangular matrix with positive diagonal elements. This factorization is mainly used as a first step for the numerical solution of linear equations $Ax = b$, where A is a symmetric, positive definite matrix.

The reference implementation of the Cholesky factorization for machines with hierarchical levels of memory is part of the LAPACK library [10]. It consists of a succession of panel (or block column) factorizations followed by updates of the trailing submatrix.

In the Chameleon library, the Cholesky routine follows a tile-based scheme in which the $n \times n$ matrix to be factorized is split in multiple submatrices, or tiles, of size $nb \times nb$ [1]. To enable the concurrent use of all the computational units on a heterogeneous platform, the Chameleon library splits the work into smaller tasks, which correspond to the computational kernels involved in performing the decomposition: `potrf`, `trsm`, `gemm` and `syrk`. The complexity of scheduling these tasks, solving data dependencies and of data consistency is delegated to StarPU [3]. By default, it uses the *lws* scheduler, because it provides correct load balancing and locality, and also takes into account priorities, although different scheduling policies can be selected, such as *eager, prio, ws,...* However, none of them considers the selection of the best value to use for the tile size, *nb*, in the Cholesky routine. Therefore, it is necessary to develop optimization strategies to suitably select the best value for *nb*.

Figure 1 shows the steps for executing a linear algebra routine of Chameleon (such as the Cholesky decomposition) using the StarPU runtime system. Each routine is computed following a tile-based algorithm. Then, a direct acyclic graph is created with the dependencies between tasks and, finally, these tasks are scheduled using the StarPU runtime system, which executes each of the tasks in the different computational units with the use of optimized implementations of the basic linear algebra routines.

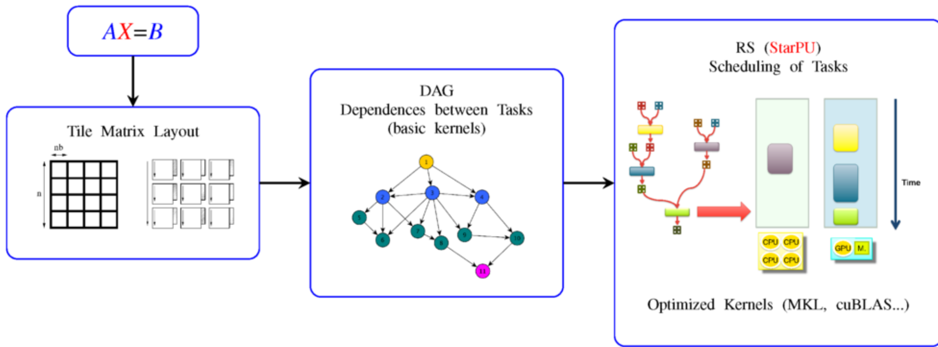

Figure 1. Execution of a linear algebra routine in Chameleon.

3. Training Strategies

Our study focuses initially on applying training strategies to select the value for the tile size, *nb*, of the Cholesky routine for a number of selected and representative problem sizes. Empirical and simulated approaches are combined with exhaustive and pruned searching methods. So, four resulting training strategies are considered:

S1: Empirical+Exhaustive

A naive approach for tuning the library would be to collect empirical data exhaustively from a large set of experiments for representative problem sizes. The routine is experimentally executed on the heterogeneous platform using a set of tile sizes for each selected problem size, n. As a result, the performance for each pair (n, nb) is obtained and stored for further use.

S2: Empirical+Pruned

Usually the time employed by *S1* approach is very high. So, to reduce the experimentation time, ensuring at the same time results close to the exhaustive ones, a pruned strategy of the search space can be used. It exploits the fact that the tile size *nb* trades off the performance of an individual task (the higher the *nb*, the higher the performance of the task) with the concurrency between tasks (the smaller the *nb*, the wider the DAG of tasks). We therefore consider a strategy similar to the one employed on multicore-only platforms [11]. In the proposed approach, the search starts with the lowest problem size (e.g, $n = 2000$) and seeks the optimum tile size *nb*. Once a given problem size n has been explored, the next problem size ($n = 4000, 8000, \ldots$, in that order) is investigated. The key idea is that the search continues with the next problem size using as its starting-point the best tile size selected for the previous problem size. For instance, if the optimum tile size for $n = 4000$ is $nb = 256$, the pruned strategy directly assesses $nb = 256$ (not evaluating the previous value for the tile size) for $n = 8000$ and, then, the next tile size (in increasing order) is considered until it reaches a tile size with which the performance is not improved. Then, the process continues with the next problem size using as starting-point the best *nb* obtained for the previous problem size, and so on.

S3: Simulated+Exhaustive

The *S1* and *S2* strategies require access to a heterogeneous platform for the experiments. Instead, we can use a simulator and apply an off-line training strategy on a separate laptop. For this purpose, we use the SimGrid simulator [9]. During an empirical phase, for each tile size and set of problem sizes, a very quick sampling of data is collected for each of the routine kernels and the generated information is stored in files called *codelets*. The information stored is based on performance models of the execution time, which can be history-based or regression-based, and is used by the simulator to estimate the duration of a task. After that, the simulator could be used over these *codelets* on a personal laptop to estimate the performance for each pair (n, nb), so reducing the experimentation time with respect to the empirical approaches.

S4: Simulated+Pruned

This strategy is applied in the same way as *S2*, but using the information collected after applying the *S3* strategy. The goal is to further reduce the search time required to obtain the tile size for each problem size while maintaining a good performance estimation.

4. Experimental Results

The experiments were carried out on a heterogeneous node with 12 CPU cores (2 hexa-core) and 6 NVIDIA GPUs (4 GeForce GTX590 and 2 Tesla K20c) using the set of problem sizes $\{2000, 4000, 8000, \ldots, 32000\}$ and a fixed set of tile sizes $\{208, 256, 288, 320, 384, 448, 512, 576\}$.

4.1. Searching the Tile Size

The results obtained for the Cholesky routine of Chameleon using the exhaustive strategies (*S1* and *S3*) are shown in Figure 2. Figure 2a shows the results obtained with the empirical *S1* strategy. The performance (y-axis) for each problem size significantly depends on the tile size nb, reaching the asymptotic value when using the highest tile sizes in larger problem sizes. Figure 2b, instead, shows the results when using the simulated *S3* strategy. The performance achieved for each problem size is very similar to that obtained with the empirical strategy, especially for large matrix and tile sizes.

If we consider the empirical and simulated approaches using the pruned strategy, satisfactory results are obtained. Figure 3 shows that the performances obtained with the *S1* and *S2* strategies perfectly overlap, but both approaches use the actual platform to perform the search for the tile-size values. The simulated *S3* and *S4* strategies, however, return very decent tuning without (almost) using the actual compute node during the training phase, achieving performance results similar to the empirical ones.

The results obtained with the four strategies are similar in terms of performance, but not in terms of the search time required. Table 1 compares the nb value selected for each problem size using each one of the strategies (values also shown in Figure 3) and the time employed in finding each value during the search process. In the empirical approaches (*S1* and *S2* strategies) the selected values for the tile size are identical, but the search time employed is lower when using the pruned strategy. The *S1* strategy uses 30 minutes of

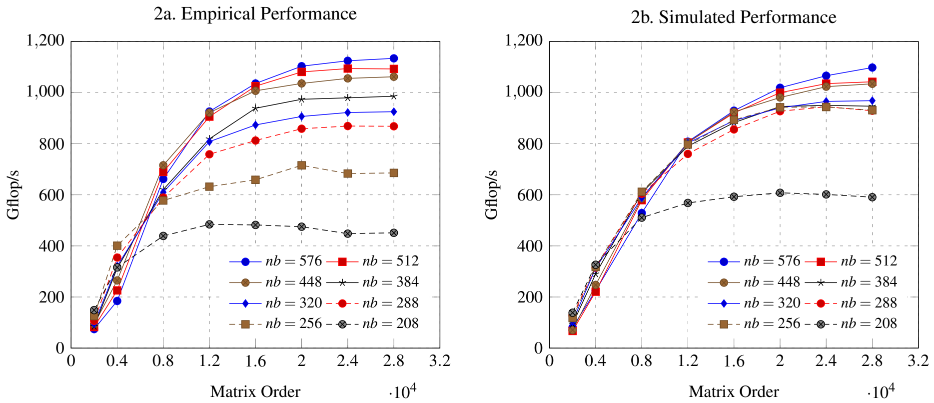

Figure 2. Empirical (2a) and simulated (2b) performance of the Cholesky routine of Chameleon using a fixed set of *nb* values for each problem size.

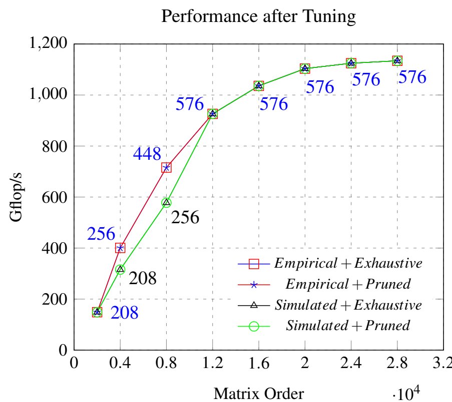

Figure 3. Performance of the Cholesky routine of Chameleon after applying each of the four considered strategies (the selected *nb* value by each strategy is also displayed).

platform time to find the *nb* values. However, by using the *S2* pruned strategy, the time is reduced to 154 seconds. With the simulated approaches (*S3* and *S4* strategies), the values obtained for the tile size are similar to those obtained with the empirical approaches, but with much less search time, and the search can be done on a separate laptop. For small and medium problem sizes, the *nb* values differ slightly from those obtained with the empirical approaches due to small variations in the performance estimated by the simulator, but the time employed in searching for the *nb* values is reduced. With the *S3* strategy, the simulation time is about 30 minutes and with the *S4* pruned strategy it is reduced to only 84 seconds. There are some entries in the table for the *S2* and *S4* pruning strategies for which the time is not displayed (represented as '-'). This is because when the search reaches the highest value considered for *nb*, this value is used for higher problem sizes, so no time is spent searching for them.

Table 1. Value for the tile size (*nb*) for each problem size using the different strategies and execution time (in seconds) employed during the search for each strategy and problem size.

	$S1$		$S2$		$S3$		$S4$	
n	*nb*	time	*nb*	time	*nb*	time	*nb*	time
2000	208	67	208	17	208	2	208	1
4000	256	71	256	25	208	5	208	3
8000	448	89	448	67	256	24	256	13
12000	576	125	576	45	576	67	576	67
16000	576	184	576	-	576	143	576	-
20000	576	273	576	-	576	321	576	-
24000	576	400	576	-	576	470	576	-
28000	576	581	576	-	576	753	576	-

4.2. Testing the Training Strategies

Once the routine has been trained for a set of problem sizes and different values for the tile size, we test the validity of the training with a set of experiments for an intermediate set of problem sizes. This testing process consists of applying an interpolation process to the information stored for the tile size during the search process performed by each of the training strategies. The goal of this tuning strategy is to analyze how far the results obtained (in terms of Gflops) with the tile size selected are with respect to the experimental optimum. Table 2 compares the results obtained. In general, the selected *nb* value (*nb* column) differs slightly from the optimum (*nb_opt* column). Furthermore, the deviation of the performance with the tuning strategy with respect to the experimental optimum (*dev* column) is quite small, mainly for large problem sizes (between 1% and 5%). Nevertheless, this interpolation process can be considered a valid tuning strategy because it allows fast prediction of a good value for *nb* for a given problem size.

Table 2. Comparison of the tile size (*nb*) selected by applying an interpolation process with respect to the experimental optimum. The *dev* column shows the deviation of the performance (in %) obtained with each tuning strategy with respect to the highest experimental performance.

		$S1$		$S2$		$S3$		$S4$	
n	*nb_opt*	*nb*	*dev*	*nb*	*dev*	*nb*	*dev*	*nb*	*dev*
3000	240	232	5	232	5	208	2	208	2
6000	288	352	12	352	12	232	15	232	15
10000	512	512	0	512	0	416	5	416	5
14000	672	576	3	576	3	576	3	576	3
18000	672	576	3	576	3	576	3	576	3
22000	896	576	1	576	1	576	1	576	1
26000	896	576	1	576	1	576	1	576	1

5. Extensions to the Experimental Study

So far, the experiments have been carried out considering a fixed set of values for the tile size and using all the computing units of the heterogeneous node. Results are satisfactory for the proposed training strategies, but when an intermediate set of problem sizes is used

(e.g, by a user), the decisions in the selection of the value for the algorithmic parameters are not always the best ones [12,13]. In this section we analyze how to improve the tuning process, either by adjusting the search for the tile size or by selecting the appropriate number of computing units to use.

5.1. Other Values for the Tile Size

Experimental results show that when an interpolation process is applied for some problem sizes, the deviation in Gflops with the selected *nb* is a little too far from the optimum. Rather than searching for the optimum value for *nb*, we can consider neighboring values and analyze the variability obtained in terms of performance in order to decide the best value for *nb*. Table 3 shows the deviation obtained for each one of the intermediate problem sizes when considering three values to the left and to the right of the interpolated one (*nb* column). The value used to obtain the next (or previous) neighbor is set according to the problem size. For $n \leq 5000$ a value of 8; for $5000 < n < 10000$ a value of 16 and for $n \geq 10000$ a value of 32. Therefore, the distance value used for *nb* could be automatized according to the range of problem sizes considered. A positive value in the deviation means an increase in performance over that achieved with the selected *nb* by the interpolation process, and a negative value means a decrease in the performance. In general, when $n \leq 10000$, the lowest deviation (or best improvement) is achieved with the immediately previous neighbor to *nb*. Instead, when $n > 10000$, the best neighbor is usually the third in increasing order with respect to *nb*. Therefore, the interpolation process could be slightly adjusted for a better selection of the tile size to use for a given problem size. For that, a search process could be applied, starting from the interpolated values for *nb* and considering both the distance value for *nb* in function of the problem size, and a percentage value for cases where an extreme value for *nb* is reached, in order to continue exploring in that direction until a new value decreases the performance.

Table 3. Comparison of the performance variability (in %) obtained with several neighbors with respect to the selected tile size (*nb*).

n	nb_1	dev	nb_2	dev	nb_3	dev	nb	nb_4	dev	nb_5	dev	nb_6	dev
3000	208	+4	216	+2	224	+3	232	240	+5	248	-10	256	-6
6000	304	+6	320	+7	336	+12	352	368	-4	384	+2	400	+1
10000	416	-5	448	-2	480	-1	512	544	-6	576	-3	608	-4
14000	480	0	512	+1	544	-2	576	608	-1	640	+1	672	+3
18000	480	-3	512	-4	544	-7	576	608	-3	640	-3	672	+3
22000	480	-6	512	-3	544	-8	576	608	-3	640	-2	672	-1
26000	480	-7	512	-3	544	-8	576	608	-3	640	-3	672	-1

5.2. Other Algorithmic Parameters

As mentioned, the StarPU runtime system is able to efficiently schedule the kernels among the available computational units of the system, but it tends to execute them using all the devices of the node. Our proposal is to analyze whether an appropriate selection of the number of computational units to use for each problem size allows better performances with an efficient use of the computational resources. We apply a selective search process which consists of successively adding computing units (CPU and each

GPU), following an increasingly powerful order. It is important to notice that the current version of StarPU does not support the data-transfer model between GPUs implemented on the latest NVIDIA devices. So, the process starts by searching for the best tile size for the current problem size and platform configuration (the initial device considered is the CPU). Then, the search continues by adding the most powerful GPU, and the best value for the tile size is searched for by applying a bi-directional guided search, using as starting-point the best value obtained for the previous platform configuration. When the process finishes, both the best platform configuration and tile size for each problem size are obtained. Table 4 shows the results of applying this tuning process for a set of problem sizes on the heterogeneous node considered (12 CPU cores and 6 NVIDIA GPUs: 4 GeForce GTX590, numbered 0, 2, 3 and 4, and 2 Tesla K20c, numbered 1 and 5). It is important to note that StarPU only uses physical cores of the CPU (without hyper-threading), therefore, the number of CPU cores is adjusted depending on the number of GPUs used, since one CPU core is intended to manage one GPU. For the set of problem sizes considered, the search process takes about 185 minutes, but each experiment is performed 10 times in order to obtain representative means for the Gflops. For small problem sizes, a subset of the computing units of the node is selected, but when the problem size increases it tends to use all the computing units. Column 'Tuned_Gflops' shows the performance obtained with the configuration selected by the tuning process, and 'Cham_Gflops' shows the performance of the routine when it is executed with the same tile size but using the default platform configuration. For all problem sizes, the best performance is obtained in the tuned case even when the whole platform is used, since by default StarPU schedules the tasks among workers (each of the GPUs) based on data dependencies, but does not take into account the computational power of the computing units. Therefore, despite the search time employed, this tuning process is a good strategy to consider if we want to efficiently use the computing units of the node.

Table 4. Performance obtained for each problem size with the best configuration selected ($Tuned_Gflops$) and using the default platform configuration ($Cham_Gflops$).

		Computing Units			
n	nb	CPU_Cores	GPU_IDs	$Tuned_Gflops$	$Cham_Gflops$
1000	112	12	$\{-\}$	76	46
2000	192	9	$\{1,5,0\}$	164	117
3000	192	8	$\{1,5,0,2\}$	285	196
4000	240	7	$\{1,5,0,2,3\}$	412	352
5000	256	6	$\{1,5,0,2,3,4\}$	545	465
6000	256	6	$\{1,5,0,2,3,4\}$	626	554
7000	320	6	$\{1,5,0,2,3,4\}$	687	608
8000	320	6	$\{1,5,0,2,3,4\}$	753	682
9000	304	6	$\{1,5,0,2,3,4\}$	791	713
10000	304	6	$\{1,5,0,2,3,4\}$	834	758
11000	512	6	$\{1,5,0,2,3,4\}$	880	803
12000	576	6	$\{1,5,0,2,3,4\}$	909	896

6. Conclusions

Task-based libraries allow us to efficiently schedule and execute linear algebra kernels on heterogeneous platforms, but they are not able to decide the best values for some algorithmic parameters of the routines, such as the tile size *nb*. In this work we propose some tuning strategies for selecting satisfactory values for the tile size on tile-based routines. We also analyze the best number of computing units to use for each problem size on a heterogeneous platform. The Cholesky routine is considered as proof of concept, using highly optimized implementations of the Cholesky factorization both for multicore and GPU. We focus on the tile size as the algorithmic parameter to optimize because this routine is executed in the Chameleon library by following a tile-based algorithm. The experimental results obtained are satisfactory, showing that the pruning strategies (both with empirical and simulated approaches) are good options to select the value for the tile size for each problem size in a short time, allowing us to obtain performances close to the experimental optima. Also, we show that a good selection of the computing units of the node for each problem size (mainly for medium problem sizes) is paramount if we want to efficiently use the computational resources with a better exploitation of the system. Our aim is to apply the proposed methodology to other linear algebra routines (such as LU or QR factorization) and to integrate the tuning process inside the Chameleon library, extending the study of selecting which computing units to use to bigger heterogeneous platforms (with a large number of computational resources).

Acknowledgment

This work was supported by the Spanish MCIU and AEI, as well as European Commission FEDER funds, under grant RTI2018-098156-B-C53.

References

[1] Emmanuel Agullo, Cédric Augonnet, Jack Dongarra, Hatem Ltaief, Raymond Namyst, Samuel Thibault, and Stanimire Tomov. Faster, Cheaper, Better – a Hybridization Methodology to Develop Linear Algebra Software for GPUs. In *GPU Computing Gems*, volume 2. Morgan Kaufmann, September 2010.

[2] Alfredo Buttari, Julien Langou, Jakub Kurzak, and Jack Dongarra. A class of parallel tiled linear algebra algorithms for multicore architectures. *Parallel Computing*, 35(1):38 – 53, 2009.

[3] Cédric Augonnet, Samuel Thibault, Raymond Namyst, and Pierre-André Wacrenier. StarPU: a unified platform for task scheduling on heterogeneous multicore architectures. *Concurrency and Computation: Practice and Experience*, 23(2):187–198, 2011.

[4] Intel MKL Web Page. `https://software.intel.com/en-us/mkl`. [Online; accessed 29.07.2019].

[5] Emmanuel Agullo, Jim Demmel, Jack Dongarra, Bilel Hadri, Jakub Kurzak, Julien Langou, Hatem Ltaief, Piotr Luszczek, and Stanimire Tomov. Numerical linear algebra on emerging architectures: The PLASMA and MAGMA projects. *Journal of Physics: Conference Series*, 180:012037, jul 2009.

[6] CUDA Toolkit Documentation. `https://docs.nvidia.com/cuda/cublas/index.html`. [Online; accessed 29.07.2019].

[7] Hartwig Anzt, Blake Haugen, Jakub Kurzak, Piotr Luszczek, and Jack J. Dongarra. Experiences in autotuning matrix multiplication for energy minimization on GPUs. *Concurrency and Computation: Practice and Experience*, 27(17):5096–5113, 2015.

[8] Brice Videau, Kevin Pouget, Luigi Genovese, Thierry Deutsch, Dimitri Komatitsch, Frdric Desprez, and Jean-Franois Mhaut. BOAST: A Metaprogramming Framework to Produce Portable and Efficient Computing Kernels for HPC Applications. *The International Journal of High Performance Computing Applications*, 32(1):28–44, 2018.

[9] Luka Stanisic, Samuel Thibault, Arnaud Legrand, Brice Videau, and Jean-François Méhaut. Faithful Performance Prediction of a Dynamic Task-Based Runtime System for Heterogeneous Multi-Core Architectures. *Concurrency and Computation: Practice and Experience*, 27(16):4075–4090, 2015.

[10] Ed Anderson, Zhaojun Bai, Christian H. Bischof, L. Susan Blackford, James Demmel, Jack J. Dongarra, Jeremy Du Croz, Anne Greenbaum, Sven Hammarling, A. McKenney, and Danny C. Sorensen. *LAPACK Users' Guide, Third Edition*. Software, Environments and Tools. SIAM, 1999.

[11] Emmanuel Agullo, Jack Dongarra, Rajib Nath, and Stanimire Tomov. A Fully Empirical Autotuned Dense QR Factorization for Multicore Architectures. In *Euro-Par 2011 Parallel Processing*, volume 6853 of *Lecture Notes in Computer Science*, pages 194–205, Aug 2011.

[12] Jesús Cámara, Javier Cuenca, Domingo Giménez, Luis Pedro García, and Antonio M. Vidal. Empirical installation of linear algebra shared-memory subroutines for auto-tuning. *International Journal of Parallel Programming*, 42(3):408–434, Jun 2014.

[13] Gregorio Bernabé, Javier Cuenca, Luis-Pedro García, and Domingo Giménez. Auto-tuning techniques for linear algebra routines on hybrid platforms. *Journal of Computational Science*, 10:299 – 310, 2015.