Available online at www.sciencedirect.com

SciVerse ScienceDirect

Procedia Computer Science 18 (2013) 110 – 119

Procedia Computer Science

International Conference on Computational Science, ICCS 2013

Empirical Modelling of Linear Algebra Shared-Memory Routines

Jesús Cámara[a], Javier Cuenca[b], Luis-Pedro García[c,*], Domingo Giménez[a]

[a]*Departamento de Informática y Sistemas, University of Murcia, Murcia 30071, Spain*
[b]*Departamento de Ingeniería y Tecnología de Computadores, University of Murcia, Murcia 30071, Spain*
[c]*Servicio de Apoyo a la Investigación Tecnológica, Polytechnic University of Cartagena, Cartagena 30202, Spain*

Abstract

In this work the behavior of the multithreaded implementation of some LAPACK routines on PLASMA and Intel MKL is analyzed. The main goal is to develop a methodology for the installation and modelling of shared-memory linear algebra routines so that some decisions to reduce the execution time can be taken at running time. Typical decisions are: the number of threads to use, the block or tile size in algorithms by blocks or tiles, and the routine to use when there are several algorithms or implementations to solve the problem available. Experiments carried out with PLASMA and Intel MKL show that decisions can be taken automatically and satisfactory execution times are obtained.

Keywords: Performance modelling; Linear algebra; Shared-memory; PLASMA; LAPACK

1. Introduction

Multicore processors and cc-NUMA systems can offer performance improvements, but often demanding new programming methods and algorithms to utilize efficiently the complex architecture involved. In dense linear algebra software, PLASMA [1] and FLAME [2] are examples of libraries that have been designed to achieve high performance on multicore architectures. Software optimization techniques are necessary to obtain low execution times and benefit fully from the potential of the hardware. Decisions are taken at running time as a result of the work performed at installation time, by modelling the execution time of the routines or by applying some empirical approach to study the behavior of the routines. There have been studies on automatically tuning libraries. PHiPAC [3] and ATLAS [4] tune matrix multiplication codes automatically on a large range of CPU platforms. FFTW [5] is a self-tuning library designed to generate high performance code for discrete Fourier transform. OSKI [6] combines install-time evaluations with run-time models to tune sparse-matrix vector multiplication. SPIRAL [7] is a high-performance code generation system for digital signal processing transforms. ABCLib_DRSSED [8] is a parallel eigensolver with an auto-tuning facility. Depending on the type of the computational system used, the decisions to take may differ. For instance: selecting the appropriate number of threads to use at each level of parallelism, how to assign processes to processors or select the correct combination of algorithmic parameters (block size in algorithms by blocks, tile size in algorithms by tiles, etc.).

In [9], auto-tuning is carried out by applying installation techniques to a multithread version of the BLAS-3 matrix multiplication routine (`dgemm`), which constitutes the basic subroutine for many other higher-level linear

[*]Corresponding author.
E-mail address: luis.garcia@sait.upct.es.

algebra packages (LAPACK, ScaLAPACK, PLAPACK, HeteroScaLAPACK, etc.). Since in PLASMA parallelism is not hidden inside Basic Linear Algebra Subprograms (BLAS) [10], in our paper previous ideas for installing multithreaded basic linear algebra routines are extended to higher-level routines. We propose a new empirical modelling method, and the results obtained when applying our auto-tuning methodology to PLASMA are compared with the highly tuned implementations supplied by vendors such as Intel MKL [11].

The rest of the paper is organized as follows. Section 2 presents the auto-tuning methodology for linear algebra routines in shared-memory systems and describes the empirical modelling method proposed. This method obtains a theoretical model of the execution time with experimental estimation of coefficients. The application of empirical modelling to PLASMA routines is described in section 3. In section 4 we evaluate our auto-tuning methodology experimentally in different kinds of shared-memory systems. Finally, in section 5 the conclusions are summarized and some possible extensions of the work are considered.

2. The auto-tuning methodology

In this section we describe our auto-tuning methodology for linear algebra routines. To improve the scalability of shared-memory linear algebra routines, the auto-tuning methodology explained in [9] for the `gemm` routine can be extended to higher-level routines. The goal of this methodology is to find the most appropriate number of threads to use, together with the values of other algorithmic parameters. The methodology is divided in three phases:

- When a new routine is designed, or its code is known, a model of the execution time can be developed, which is used in the subsequent phases; in other cases, the model is empirically estimated. This approach is used here to auto-tune PLASMA routines. Auto-tuning of linear algebra routines in large cc-NUMA based on theoretical models is analyzed in [12] and can be combined with the empirical approach studied here.
- When a model is not available, some experiments are conducted in the installation of the routine, to analyze the behavior of the routine in the system for some significant values (*Installation_Set*). For example, experiments are conducted for different problem sizes, numbers of threads, and block sizes in routines working by blocks. In large NUMA systems, there is a shared RAM memory space but with non uniform data access time, making it difficult to develop an accurate model. In this case, the empirical representation of the behaviour of the routine is not easy, and extensive experimentation may be necessary. The installation process is performed once for a given routine on a given platform. The information generated in the installation is stored for use when the routine is executed. This information can be included in the routine together with a decision engine to obtain an auto-tuning version of the routine.
- When a problem is being solved, the problem size and the maximum number of cores indicated by the user are used to decide the number of threads for the solution of the problem. In routines working by blocks the block size should also be selected. The selection of those parameters is done in the auto-tuning routine prior to the call to the basic routine with the values selected for the parameters. The different possible values for the algorithm parameters are substituted in the empirically estimated model, and those values which provide the lowest theoretical execution time are used in the solution of the problem.

So, an empirically estimated model of the execution time can be used to determine the most appropriate values for the algorithm parameters (number of threads, block sizes, etc.) as well as the routine to use if several are available for the problem in question. The sequential execution time of all the routines considered has a cost of order $O(n^3)$, with n being the size of the matrix, and so in the theoretical model there will be terms in n^3, n^2 and n. For the parallel version where the number of threads (t) is the only algorithm parameter, the highest cost (n^3) should be divided by t, and other terms should be multiplied by t. Thus, we could consider all possible combinations $\{n^3, n^2, n, 1\} \times \{t, 1, \frac{1}{t}\}$, but for n^3 we consider only $\frac{n^3}{t}$, and the lowest order terms ($\frac{n}{t}$, t, 1 and $\frac{1}{t}$) are not included in the model. The execution time is modelled by:

$$T(n,t) = k_1 \frac{n^3}{t} + k_2 n^2 t + k_3 n^2 + k_4 \frac{n^2}{t} + k_5 nt + k_6 n \tag{1}$$

and a least squares adjustment can be used to estimate the values of the coefficients k_i for a particular routine in a particular system.

Some experiments are performed for different problem sizes and number of threads, and the function representing the square difference of the experimental and theoretical times is:

$$F(k_1, k_2, k_3, k_4, k_5, k_6) = \sum_{i=1}^{r} (T_i - T(n_i, t_i))^2 \tag{2}$$

where r is the number of experiments, T_i the execution time of experiment i, and n_i and t_i the matrix size and the number of threads in the experiment i. On doing the partial derivate of F with respect to each k_i and making it equal to zero, we obtain a basic equation that can be solved with a QR decomposition (least squares), or solved subject to the added constraints that the solution contains no negative elements (non-negative least squares) [13]. With non-negative least squares (NNLS) the values obtained for the coefficients will be positive or zero, and with least squares (LS) all coefficients have non zero values (positive or negative).

3. Empirically modelling of PLASMA routines

To achieve high performance on multicore architectures, PLASMA library relies on tile algorithms, which provide fine granularity parallelism. PLASMA performance strongly depends on tunable execution parameters trading off utilization of different system resources. The outer block size trades off parallelization granularity and scheduling flexibility with single core utilization, while the inner block size trades off memory load with extra-flops [14]. Only the QR and LU factorizations use inner blocking. Tuning PLASMA consists of finding the outer and inner block size pairs that maximize the performance. As shown in Figure 1, for the PLASMA Cholesky (`dpotrf`) routine on different shared-memory systems, the optimum tile size depends on the number of threads and the matrix size and is also system-dependent.

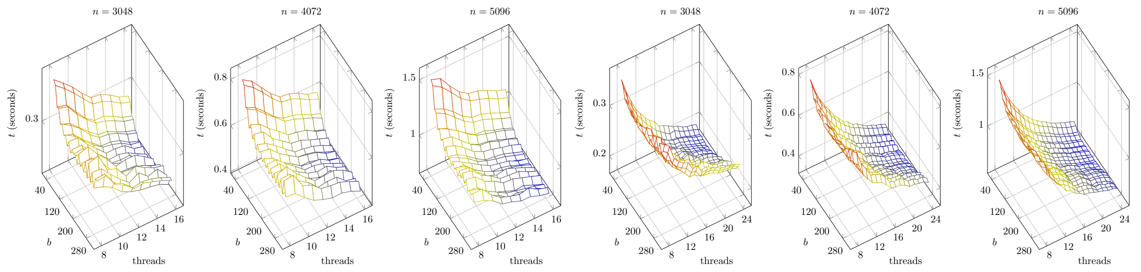

Fig. 1. PLASMA Cholesky (`dpotrf`) routine. Effect of outer block size (b) and number of threads in the execution time for different matrix sizes (n) on shared-memory machines with 16 cores (left) and 24 cores (right).

In this section, the methodology is applied to the PLASMA library to select the configuration of parameters, the number of threads, outer block size and inner block size for a particular problem and in a particular system. For the PLASMA routines, the model includes different combinations of n (matrix size), t (number of threads), b (outer block size) and l (inner block size). Thus, we could consider all the combinations $\{n^3, n^2, n, 1\} \times \{t, 1, \frac{1}{t}\} \times \{b^2, b, 1, \frac{1}{b}\} \times \{l^2, l, 1, \frac{1}{l}\}$, but the lowest order terms are not included in the model, and for n^3 we consider only $\frac{n^3}{t}$. So, the execution time for a PLASMA routine can be modelled by:

$$\begin{aligned} T(n,t,b,l) = {} & k_1\frac{n^3}{t} + k_2\frac{n^3}{bl} + k_3\frac{n^3}{l} + k_4\frac{n^3}{b} + k_5 n^2 tb + k_6 n^2 t + k_7\frac{n^2 t}{bl} + k_8\frac{n^2 t}{l} + k_9\frac{n^2 t}{b} + k_{10} n^2 tl + k_{11} n^2 b + \\ & k_{12} n^2 l + k_{13}\frac{n^2}{bl} + k_{14}\frac{n^2}{l} + k_{15}\frac{n^2}{b} + k_{16} n^2 + k_{17}\frac{n^2}{t} + k_{18}\frac{n^2 b}{t} + k_{19}\frac{n^2 l}{t} + k_{20} ntb^2 + k_{21} ntbl + \\ & k_{22} ntb + k_{23} ntl^2 + k_{24}\frac{nt}{l} + k_{25}\frac{nt}{b} + k_{26} ntl + k_{27} nt + k_{28} nb^2 + k_{29} nbl + k_{30} nb + k_{31} nl^2 + k_{32} nl + k_{33} n \end{aligned} \tag{3}$$

and, as described previously, a regression with a least squares adjustment (LS) or with a non-negative least squares (NNLS) can be performed to estimate the values of the coefficients k_i. When the PLASMA routine does not use inner blocking, for example a Cholesky factorization, equation 3 does not include terms in l. So, the execution time for a PLASMA routine that depends only on two parameters (outer block size and number of threads) can be modelled by:

$$\begin{aligned} T(n,t,b) = &k_1\frac{n^3}{t} + k_2\frac{n^3}{b} + k_3n^2tb + k_4n^2t + k_5\frac{n^2t}{b} + k_6n^2b + k_7\frac{n^2}{b} + k_8\frac{n^2}{t} + k_9n^2 + k_{10}\frac{n^2b}{t} + k_{11}ntb^2 + \\ &k_{12}ntb + k_{13}\frac{nt}{b} + k_{14}nt + k_{15}nb^2 + k_{16}nb + k_{17}n \end{aligned} \tag{4}$$

The installation of the routines in the system is made by executing the routine for each matrix size specified in an *Installation_Set* (set of some problem sizes used to install the routine in the system) by varying the number of threads, the outer block size and the inner block size to some possible preselected values. The experimental estimation of the coefficients is made with LS or NNLS and the model (`Mod-LS` or `Mod-NNLS`) for the execution time of the routine is obtained. Once the routine has been installed, the model and the different possible values for the algorithm parameters are stored. At execution time, the number of threads and the tile sizes with which the lowest time is obtained for each problem size are selected by using the information provided by the model empirically obtained at installation time.

4. Experimental results

This section shows the efficiency of our automatic tuning method for PLASMA routines on NUMA platforms, and the results obtained are compared with those with the Intel MKL implementation of the LAPACK library.

To validate this methodology experiments have been carried out in different shared-memory systems:

- **Hipatia**: a cluster with 14 nodes with 2 Intel Xeon Quad-Core, 2.80 GHz, and 2 nodes with 4 Intel Xeon Quad-Core, 2.93 GHz. The nodes with 16 cores are used in the experiments. The operating system is Linux 2.6.18 and it provides Intel `icc` (v12.0.0) and Intel MKL (v10.3.2).
- **Saturno**: a NUMA system with 24 cores, 4 Intel Xeon X7542 (hexa-core) processors, 1.87 GHz, 32 GB of shared-memory. The machine run Linux 2.6.35, Intel `icc` compiler (v12.0.2) and Intel MKL (v10.3.2).
- **Joule**: a NUMA system with 64 cores, 4 AMD Opteron 6276 (16 cores) processors, 2.3 GHz, 64 GB of shared-memory. Linux 2.6.32, Intel `icc` compiler (v12.1.3) and Intel MKL (v10.3.9) are used.

To obtain the values of the coefficients in the `Mod-NNLS` model, we used a freely available C interface to the Lawson-Hanson implementation of an algorithm for non-negative least squares [15]. To achieve accurate timing, we called the PLASMA routines six times for small matrix sizes, the average value of these executions was used with elimination of the non normal ones. The executions are performed with `numactl --interleave` to force that matrices to be distributed among the memory cores.

As an example we analyze how the empirically modelling works for the Cholesky (`dpotrf`) and QR (`dgeqrf`) factorizations. The Cholesky routine depends on two parameters (outer block size and number of threads), while the QR depends on the three parameters (outer block size, inner block size and number of threads). Experiments were conducted for some significant values (*Installation_Set*) of the problem size, number of threads that corresponds to the number of cores available per processor or socket, and outer-inner block size combinations that include default values used in PLASMA routines.

The *Installation_Set* used in Hipatia, Saturno and Joule for the PLASMA Cholesky factorization was {500, 1500, 2500, 3500, 4500, 5500, 6500, 7500, 8500, 9500}; with b ranging from 40 to 300, $b\_inc = 40$, and t ranging from 4 to the number of available cores in Hipatia, with $t\_inc = 4$, from 6 to the maximum number of available cores in Saturno, with $t\_inc = 6$, and from 16 to the maximum number of available cores in Joule, with $t\_inc = 16$.

With this *Installation_Set* the coefficients with non zero values obtained with NNLS for equation 4 on each platform were: k_1 and k_2 in Hipatia, k_1, k_2, k_3 and k_{11} in Saturno and k_1, k_2, k_3, k_6 and k_{11} in Joule. Therefore, the model (`Mod-NNLS`) for the estimated execution time would be:

$$T(n,t,b) = k_1\frac{n^3}{t} + k_2\frac{n^3}{b} \tag{5}$$

for the Hipatia platform.

$$T(n,t,b) = k_1\frac{n^3}{t} + k_2\frac{n^3}{b} + k_3n^2tb + k_{11}ntb^2 \tag{6}$$

for the Saturno platform, and finally:

$$T(n,t,b) = k_1\frac{n^3}{t} + k_2\frac{n^3}{b} + k_3n^2tb + k_6n^2b + k_{11}ntb^2 \tag{7}$$

for the Joule platform.

It is observed that with the `Mod-NNLS` method the initial theoretical model (equation 4) changes with the execution platform. In this way, the experimental estimation of the coefficients with the `Mod-NNLS` method provides an insight about the contribution of the value of the algorithmic parameters to the execution time of the routine.

Tables 1 (Hipatia), 2 (Saturno) and 3 (Joule) summarize the experimental results with the PLASMA Cholesky factorization. The optimum configuration (number of threads, outer block size) and the selection provided by the empirical modelling techniques `Mod-LS` and `Mod-NNLS` are shown for different problem sizes. The columns "deviation" show the deviation obtained with each method with respect to the optimum execution time for the values in a *Validation_Set* (set of some problem sizes used to validate the models in each system). The column "Default" shows the execution time for PLASMA Cholesky with no selection of outer block size parameter, and with as many threads as cores available in the platform.

Table 1. Comparison of the number of threads and outer block size configuration selected with the empirical modelling techniques `Mod-LS` and `Mod-NNLS` for the PLASMA Cholesky routine, and the deviation with each method, in Hipatia.

size	Optimum	Default	`Mod-LS`	deviation	`Mod-NNLS`	deviation
2000	0.127 (12,200)	0.142	0.140 (8,280)	9.85 %	0.132 (16,280)	4.17 %
3000	0.266 (16,280)	0.323	0.270 (12,280)	1.91 %	0.266 (16,280)	0.00 %
4000	0.479 (16,280)	0.625	0.479 (16,280)	0.00 %	0.479 (16,280)	0.00 %
5000	0.788 (16,280)	1.047	0.788 (16,280)	0.00 %	0.788 (16,280)	0.00 %
6000	1.206 (16,280)	1.663	1.206 (16,280)	0.00 %	1.206 (16,280)	0.00 %
7000	1.746 (16,280)	2.579	1.746 (16,280)	0.00 %	1.746 (16,280)	0.00 %
8000	2.442 (16,280)	3.686	2.442 (16,280)	0.00 %	2.442 (16,280)	0.00 %
9000	3.351 (16,280)	5.140	3.351 (16,280)	0.00 %	3.351 (16,280)	0.00 %
10000	4.380 (16,280)	6.935	4.380 (16,280)	0.00 %	4.380 (16,280)	0.00 %

Table 2. Comparison of the number of threads and outer block size configuration selected with de empirical modelling techniques `Mod-LS` and `Mod-NNLS` for the PLASMA Cholesky routine, and the deviation with each method, in Saturno.

size	Optimum	Default	`Mod-LS`	deviation	`Mod-NNLS`	deviation
2000	0.078 (24,80)	0.081	0.092 (12,80)	16.90 %	0.078 (24,80)	0.00 %
3000	0.181 (24,80)	0.182	0.204 (18,80)	12.64 %	0.181 (24,80)	0.00 %
4000	0.347 (24,120)	0.347	0.347 (24,120)	0.00 %	0.349 (24,80)	0.45 %
5000	0.587 (24,120)	0.587	0.591 (24,160)	0.57 %	0.587 (24,120)	0.00 %
6000	0.915 (24,120)	0.915	0.930 (24,160)	1.72 %	0.915 (24,120)	0.00 %
7000	1.333 (24,160)	1.352	1.345 (24,200)	0.88 %	1.352 (24,120)	1.37 %
8000	1.883 (24,160)	1.900	1.905 (24,240)	1.13 %	1.883 (24,160)	0.00 %
9000	2.557 (24,200)	2.598	2.580 (24,240)	0.91 %	2.587 (24,160)	1.20 %
10000	3.366 (24,200)	3.447	3.402 (24,280)	1.08 %	3.384 (24,160)	0.54 %

The differences in the execution time between the optimum and the installation techniques are bigger for small problems, for which the models of the execution time are not so satisfactory as for large problems. The `Mod-LS` technique gives a mean deviation of the optimum of approximately 1% in Hipatia, 4% in Saturno and 5% in Joule. The `Mod-NNLS` technique gives a mean deviation of the optimum of approximately 0.5% in Hipatia, 0.4% in Saturno and 7% in Joule. Thus, the preferred technique is `Mod-NNLS` for the PLASMA Cholesky routine in the three systems analyzed. The mean improvement (mean of the deviation between "Optimum" and "Default" columns) that we could obtain with respect to the "Default" execution is approximately 27% in Hipatia, 1% in Saturno and 13% in Joule.

Table 3. Comparison of the number of threads and outer block size configuration selected with de empirical modelling techniques `Mod-LS` and `Mod-NNLS` for the PLASMA Cholesky routine, and the deviation with each method, in Joule.

size	Optimum	Default	`Mod-LS`	deviation	`Mod-NNLS`	deviation
2000	0.124 (32,40)	0.185	0.159 (16,200)	27.33 %	0.147 (64,40)	18.42 %
3000	0.216 (64,60)	0.285	0.232 (48,40)	7.38 %	0.216(64,60)	0.00 %
4000	0.377 (64,60)	0.446	0.377 (64,60)	0.00 %	0.377 (64,60)	0.00 %
5000	0.594 (64,140)	0.676	0.642 (64,60)	8.05 %	0.642 (64,60)	8.05 %
6000	0.865 (64,140)	0.987	0.886 (64,60)	2.38 %	1.054 (64,80)	21.83 %
7000	1.235 (64,140)	1.432	1.255 (64,60)	1.65 %	1.357 (64,80)	9.91 %
8000	1.940 (64,200)	2.005	1.940 (64,200)	0.00 %	1.965 (64,100)	1.29 %
9000	2.578 (64,200)	2.668	2.578 (64,200)	0.00 %	2.629 (64,100)	1.76 %
10000	3.370 (64,200)	3.512	3.370 (64,200)	0.00 %	3.427 (64,100)	1.70 %

For the PLASMA QR factorization, the *Installation_Set* used in Hipatia, Saturno and Joule was {512, 1024, 1536, 2048, 2560, 3072, 3584, 4096, 4608, 5120}; with b ranging from 24 to 304 and $b\_inc = 40$, l from 28 to 208 with $l\_inc = 20$ and t with the same values as used in PLASMA Cholesky factorization.

With this *Installation_Set* on each platform, the model (`Mod-NNLS`) for the estimated execution time would be:

$$T(n,t,b,l) = k_1\frac{n^3}{t} + k_2\frac{n^3}{bl} + k_{19}\frac{n^2 l}{t} \tag{8}$$

for the Hipatia platform.

$$T(n,t,b,l) = k_1\frac{n^3}{t} + k_4\frac{n^3}{b} + k_{10}n^2tl + k_{20}ntb^2 + k_{23}ntl^2 + k_{24}\frac{nt}{l} \tag{9}$$

for the Saturno platform, and finally:

$$T(n,t,b,l) = k_1\frac{n^3}{t} + k_4\frac{n^3}{b} + k_{12}n^2l + k_{13}\frac{n^2}{bl} + k_{17}\frac{n^2}{t} + k_{19}\frac{n^2 l}{t} + k_{20}ntb^2 + k_{21}ntl^2 + k_{28}nb^2 \tag{10}$$

for the Joule platform.

Tables 4 (Hipatia), 5 (Saturno) and 6 (Joule) show, for different matrix sizes, the execution time (in seconds) obtained for the PLASMA QR factorization with optimum configuration (number of threads, outer block size, inner block size) and the selection provided by the empirical modelling techniques `Mod-LS` and `Mod-NNLS`. The columns "deviation" show the deviation obtained with each method with respect to the optimum execution time. The column "Default" shows the execution time for PLASMA QR with no selection of outer block size and inner block size parameters, and with as many threads as cores available in the system. For the PLASMA QR routine, the `Mod-LS` technique gives a mean deviation of the optimum of approximately 15% in Hipatia, 1% in Saturno and 8% in Joule, while the `Mod-NNLS` technique gives a mean deviation of the optimum of approximately 6% in Hipatia, 2% in Saturno and 4% in Joule. Again, the preferred modelling technique is `Mod-NNLS`. The mean improvement that we could obtain with respect to the "Default" execution is approximately 34% in Hipatia, 2% in Saturno and 7% in Joule.

The use of the models normally produces a reduction of the execution time with respect to the execution time without selection of the parameters, but this reduction is lower than the optimum obtained experimentally, because the model does not always select the best parameters combination. The selection of the number of threads, outer block size and inner block size allows a reduction of the execution time, which is greater for larger matrices. The values of the algorithmic parameters vary for different systems, problem sizes and routines, but with the models the auto-tuned PLASMA version can make a satisfactory selection of these parameters at execution time.

Table 4. Comparison of the number of threads, outer block size and inner block size configuration selected with the empirical modelling techniques `Mod-LS` and `Mod-NNLS` for the PLASMA QR routine, and the deviation with each method, in Hipatia.

size	Optimum	Default	`Mod-LS`	deviation	`Mod-NNLS`	deviation
1512	0.194 (16,224,88)	0.228	0.252 (12,304,188)	29.63 %	0.219 (16,304,68)	12.60 %
2024	0.318 (16,264,88)	0.464	0.377 (16,304,188)	18.48 %	0.348 (16,304,68)	9.12 %
2536	0.498 (16,264,88)	0.743	0.576 (16,304,188)	15.47 %	0.551 (16,304,68)	10.54 %
3048	0.715 (16,264,88)	1.209	0.836 (16,304,168)	16.84 %	0.803 (16,304,88)	12.32 %
3560	1.105 (16,264,88)	1.695	1.179 (16,304,168)	6.63 %	1.131 (16,304,88)	2.36 %
4072	1.528 (16,304,108)	2.489	1.500 (16,304,168)	4.66 %	1.536 (16,304,88)	0.52 %
4584	2.069 (16,264,88)	3.256	2.425 (16,304,28)	17.15 %	2.124 (16,304,108)	2.62 %
5096	2.683 (16,304,88)	4.414	2.990 (16,304,28)	11.44 %	2.690 (16,304,108)	0.24 %
5608	3.445 (16,304,108)	5.464	3.950 (16,304,28)	14.68 %	3.445 (16,304,108)	0.00 %

Table 5. Comparison of the number of threads, outer block size and inner block size configuration selected with the empirical modelling techniques `Mod-LS` and `Mod-NNLS` for the PLASMA QR routine, and the deviation with each method, in Saturno.

size	Optimum	Default	`Mod-LS`	deviation	`Mod-NNLS`	deviation
1512	0.097 (24,104,28)	0.106	0.097 (24,104,28)	0.00 %	0.105 (24,104,68)	8.39 %
2024	0.178 (24,104,28)	0.192	0.178 (24,104,28)	0.00 %	0.190 (24,104,68)	6.42 %
2536	0.306 (24,144,48)	0.306	0.318 (24,144,28)	3.76 %	0.306 (24,144,48)	0.00 %
3048	0.485 (24,144,48)	0.485	0.485 (24,144,48)	0.00 %	0.485 (24,144,48)	0.00 %
3560	0.695 (24,144,48)	0.695	0.695 (24,144,48)	0.00 %	0.695 (24,144,48)	0.00 %
4072	1.013 (24,144,48)	1.013	1.024 (24,184,48)	1.06 %	1.024 (24,184,48)	1.06 %
4584	1.333 (24,184,48)	1.336	1.333 (24,184,48)	0.00 %	1.333 (24,184,48)	0.00 %
5096	1.795 (24,184,48)	1.822	1.812 (24,224,48)	0.94 %	1.812 (24,224,48)	0.94 %
5608	2.315 (24,144,48)	2.315	2.390 (24,224,48)	3.21 %	2.389 (24,224,48)	3.21 %

Table 6. Comparison of the number of threads, outer block size and inner block size configuration selected with the empirical modelling techniques `Mod-LS` and `Mod-NNLS` for the PLASMA QR routine, and the deviation with each method, in Joule.

size	Optimum	Default	`Mod-LS`	deviation	`Mod-NNLS`	deviation
1512	0.123 (64,64,28)	0.181	0.141 (32,104,28)	14.15 %	0.127 (64,64,48)	3.38 %
2024	0.216 (48,64,28)	0.270	0.253 (64,104,48)	17.23 %	0.238 (64,64,48)	10.39 %
2536	0.334 (64,64,28)	0.363	0.341 (64,104,28)	2.12 %	0.340 (64,104,48)	7.65 %
3048	0.488 (64,64,28)	0.505	0.514 (64,104,28)	5.31 %	0.515 (64,104,48)	5.60 %
3560	0.700 (64,64,28)	0.704	0.703 (64,104,28)	0.45 %	0.705 (64,104,48)	0.78 %
4072	0.924 (64,144,48)	0.924	0.965 (64,104,28)	4.38 %	0.977 (64,104,48)	5.74 %
4584	1.221 (64,144,48)	1.221	1.308 (64,104,28)	7.05 %	1.242 (64,144,28)	1.68 %
5096	1.584 (64,144,48)	1.584	1.692 (64,104,28)	6.87 %	1.608 (64,144,28)	1.56 %
5608	2.006 (64,144,48)	2.006	2.229 (64,104,28)	11.10 %	2.039 (64,144,28)	1.62 %

4.1. *Comparison with Intel MKL LAPACK*

In this section we provide a comparison of PLASMA's behaviour against equivalent commercial software such as Intel MKL. The comparison of the behaviour of some routines was conducted on the three different multicore architectures, based on Intel Xeon (Hipatia and Saturno) and AMD (Joule). The results with Hipatia (figure 2) have been obtained in a node with 16 cores using 16 threads. The results with Saturno (figure 3) have been obtained using 24 threads, and in Joule (figure 4) 64 threads have been used.

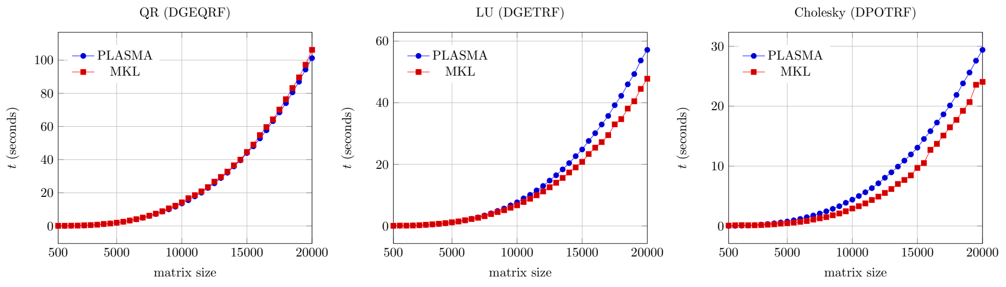

Fig. 2. Comparison of the execution times of the routines QR, LU and Cholesky, in Hipatia with 16 threads.

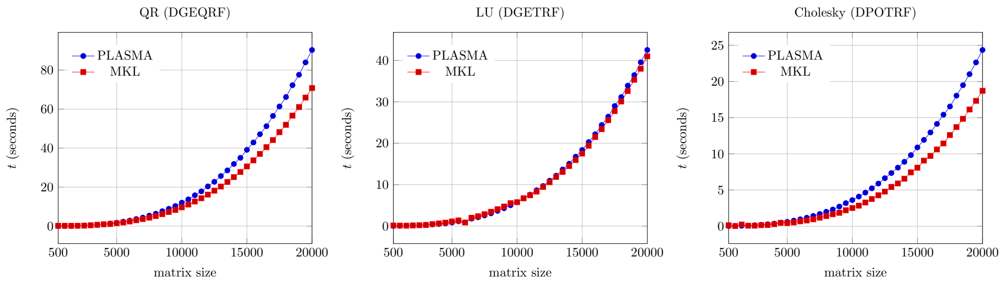

Fig. 3. Comparison of the execution times of the routines QR, LU and Cholesky, in Saturno with 24 threads.

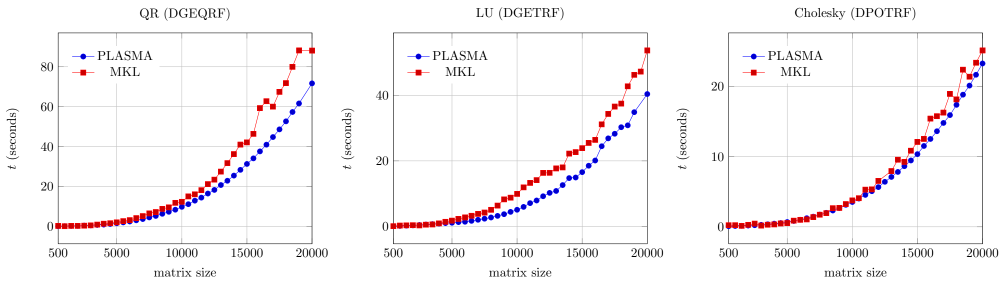

Fig. 4. Comparison of the execution times of the routines QR, LU and Cholesky, in Joule with 64 threads.

In the experiments, default values were used for the tile sizes: outer block size equal to 120 for the Cholesky factorization; outer block size equal to 200, and inner block size equal to 20 for the LU factorization, and outer block of size 144 and inner block of size 48 for the QR factorization. The figure illustrate that, in these systems, the Intel MKL outperform PLASMA for large matrices (except for the QR routine), and only for some small and moderate problem sizes are better results obtained with PLASMA. The MKL routines yields different improvement percentages with respect to the PLASMA in the three systems, which makes it impossible to draw general conclusions about the advantage of using MKL. The differences in the execution time between the libraries is not very large; therefore PLASMA can compete against MKL with a correct selection of the tile sizes together with the number of threads for a particular problem size and in a particular system.

In this case, the auto-tuning methodology can be used to decide which implementation to use and the values of the parameters. The models of these routines should provide information of the preferred routine and of the number of threads to use when solving a particular problem, depending on the problem size and on the size of the computational system (the number of cores reserved to solve the problem).

Table 7. Comparison of the time obtained for the PLASMA LU routine with the number of threads, outer block size and inner block size configuration (in brackets) selected with the methods `Mod-LS` and `Mod-NNLS`, the lowest time obtained with MKL LU and the lowest time obtained with PLASMA LU (Default). In Hipatia, times in seconds.

size	`Mod-LS`	`Mod-NNLS`	Default	MKL
2000	0.176 (16,120,20)	0.239 (16,280,40)	0.190	**0.147**
3000	0.433 (16,120,20)	0.447 (16,280,60)	**0.377**	0.378
4000	0.726 (16,280,100)	0.729 (16,280,60)	**0.669**	0.695
5000	1.124 (16,280,120)	**1.121** (16,280,80)	1.127	1.196
6000	**1.745** (16,280,240)	1.777 (16,280,80)	1.820	1.853
7000	2.621 (16,280,20)	**2.619** (16,280,80)	2.779	2.659
8000	**3.797** (16,280,20)	3.821 (16,280,40)	4.088	3.863
Total	**10.622**	10.753	11.05	10.791

Table 8. Comparison of the time obtained for the PLASMA LU routine with the number of threads, outer block size and inner block size configuration (in brackets) selected with the methods `Mod-LS` and `Mod-NNLS`, the lowest time obtained with MKL LU and the lowest time obtained with PLASMA LU (Default). In Saturno, times in seconds.

size	`Mod-LS`	`Mod-NNLS`	Default	MKL
3000	**0.214** (24,120,20)	0.464 (24,280,60)	0.324	0.279
4000	**0.383** (24,160,60)	0.647 (24,280,60)	0.504	0.653
5000	**0.692** (24,160,60)	0.859 (24,280,60)	0.865	1.121
6000	**1.144** (24,160,60)	1.184 (24,280,80)	1.156	1.853
7000	1.777 (24,160,120)	**1.770** (24,280,80)	2.131	2.359
8000	2.508 (24,200,20)	**2.498** (24,280,80)	2.508	3.452
9000	3.521 (24,200,20)	**3.464** (24,280,100)	3.521	4.727
Total	**10.239**	10.886	11.09	14.444

Table 9. Comparison of the time obtained for the PLASMA LU routine with the number of threads, outer block size and inner block size configuration (in brackets) selected with the methods `Mod-LS` and `Mod-NNLS`, the lowest time obtained with MKL LU and the lowest time obtained with PLASMA LU (Default). In Joule, times in seconds.

size	`Mod-LS`	`Mod-NNLS`	Default	MKL
3000	0.750 (64,120,20)	0.828 (64,280,80)	0.596	**0.536**
4000	**0.822** (64,160,160)	1.164 (64,280,60)	0.848	0.938
5000	**1.137** (64,200,200)	1.672 (64,280,60)	1.168	1.775
6000	**1.458** (64,200,200)	2.122 (64,280,60)	1.493	2.736
7000	**1.939** (64,200,200)	2.817 (64,280,80)	2.029	3.826
8000	**2.730** (64,200,20)	2.758 (64,280,80)	**2.730**	5.066
9000	**3.738** (64,200,20)	3.983 (64,280,80)	**3.738**	8.259
Total	**12.574**	15.344	12.602	23.136

Table 7 (Hipatia) shows the execution time (in seconds) for different matrix sizes, obtained for the PLASMA LU factorization when the parameters are selected with the two methods considered (`Mod-LS` and `Mod-NNLS`), the lowest execution time obtained for the MKL LU factorization, and the lowest time obtained with PLASMA LU factorization without parameter tuning. The improvements obtained with the two techniques are close, with a difference in the total time of approximately 4% in favor of `Mod-LS` with respect to the optimum execution time. Thus, the preferred installation method is `Mod-LS`. Similar results were obtained in the other two systems. In Saturno (table 8) the improvement in the total time is about 7% in favor of `Mod-LS`. In Joule (table 9) the auto-tuning selects satisfactory values of the parameters, but in this case the results are similar to those obtained without

parameter tuning. In this system, the difference in the total time with respect to the optimum is approximately 0.22% in favor of `Mod-LS`.

5. Conclusions

This paper describes an empirical auto-tuning approach for PLASMA implementation of LAPACK routines in NUMA systems. The auto-tuning approach involves providing a set of empirically obtained models that permits a satisfactory selection of the algorithmic parameters. The methodology has been tested in different systems. We focus on the Cholesky, QR and LU factorizations, but the work is representative of the process to be done for auto-tuning the whole library. The methodology has been shown to be useful for the design of routines which can obtain execution time close to the lowest achievable without the need for user expertise. The results have also shown that PLASMA can, with the application of our methodology, be very competitive compared to the Intel MKL library.

We are working on the application of similar methodologies to other types of parallelism (message-passing and GPUs), and to routines of different types or to particular scientific codes.

Acknowledgements

Partially supported by Fundación Séneca, Consejería de Educación de la Región de Murcia, 08763/PI/08, and the Spanish MINECO under grant TIN2012-38341-C04-03. The authors would like to thank Erique S. Quintana-Ortí from the High Performance Computing & Architectures group at the University Jaume I for granting us access to their Joule computing platform.

References

[1] Parallel Linear Algebra for Scalable Multi-core Architectures (PLASMA) project. `http://icl.cs.utk.edu/plasma`.
[2] G. Quintana-Ortí, E. Quintana-Ortí, R. V. D. Geijn, F. V. Zee, E. Chan, Programming matrix algorithms-by-blocks for thread-level parallelism, ACM Trans. Math. Softw. 36 (3) (2009) 14:1–14:26.
[3] J. Bilmes, K. Asanovic, C.-W. Chin, J. Demmel, Optimizing matrix multiply using PHiPAC: a portable, high-performance, ANSI C coding methodology, in: Proceedings of the 11th international conference on Supercomputing, ICS '97, ACM, 1997, pp. 340–347.
[4] R. C. Whaley, A. Petitet, J. Dongarra, Automated empirical optimization of software and the ATLAS project, Parallel Computing 27 (1-2) (2001) 3–35.
[5] M. Frigo, Steven, G. Johnson, The design and implementation of FFTW3, in: Proceedings of the IEEE, 2005, pp. 216–231.
[6] R. Vuduc, J. W. Demmel, K. A. Yelick, OSKI: A library of automatically tuned sparse matrix kernels, in: Proc. SciDAC, J. Physics: Conf. Ser., Vol. 16, 2005, pp. 521–530.
[7] M. Püschel, J. M. F. Moura, J. Johnson, D. Padua, M. Veloso, B. Singer, J. Xiong, F. Franchetti, A. Gacic, Y. Voronenko, K. Chen, R. W. Johnson, N. Rizzolo, SPIRAL: Code generation for DSP transforms, Proceedings of the IEEE, special issue on "Program Generation, Optimization, and Adaptation" 93 (2) (2005) 232– 275.
[8] T. Katagiri, K. Kise, H. Honda, T. Yuba, ABCLib_DRSSED: A parallel eigensolver with an auto-tuning facility., Parallel Computing 32 (3) (2006) 231–250.
[9] J. Cámara, J. Cuenca, D. Giménez, A. M. Vidal, Empirical autotuning of two-level parallel linear algebra routines on large cc-NUMA systems, in: IEEE 10th International Symposium on Parallel and Distributed Processing with Applications (ISPA), 2012, pp. 843–844.
[10] J. J. Dongarra, J. D. Croz, S. Hammarling, R. J. Hanson, An extended set of fortran basic linear algebra subprograms, ACM Transactions on Mathematical Software 14 (1988) 1–17.
[11] Intel Math Kernel Library (MKL). `http://www.intel.com/software/products/mkl`.
[12] J. Cámara, J. Cuenca, L.-P. García, D. Giménez, Auto-tuned nested parallelism: a way to reduce the execution time of scientific software in NUMA systems, in: 7th International Workshop on Parallel Matrix Algorithms and Applications (PMAA), 2012.
[13] C. L. Lawson, R. J. Hanson, Solving Least Squares Problems, Prentice Hall, 1974.
[14] A. Buttari, J. Langou, J. Kurzak, J. Dongarra, A class of parallel tiled linear algebra algorithms for multicore architectures, Parallel Computing 35 (1) (2009) 38–53.
[15] C interface to Algorithm NNLS. `https://software.sandia.gov/appspack/version3/nnls_8c-source.html`.