

Tuning the Blocksize for Dense Linear Algebra Factorization Routines with the Roofline Model

Peter Benner³, Pablo Ezzatti¹, Enrique S. Quintana-Ortí², Alfredo Remón^{3(✉)},
and Juan P. Silva¹

¹ Instituto de Computación, Universidad de la República,
11300 Montevideo, Uruguay
{pezzatti,jpsilva}@fing.edu.uy

² Dep. de Ingeniería y Ciencia de la Computación, Universidad Jaime I,
12701 Castellón, Spain
quintana@icc.uji.es

³ Max Planck Institute for Dynamics of Complex Technical Systems,
39106 Magdeburg, Germany
{benner, remon}@mpi-magdeburg.mpg.de

Abstract. The optimization of dense linear algebra operations is a fundamental task in the solution of many scientific computing applications. The *Roofline Model* is a tool that provides an estimation of the performance that a computational kernel can attain on a hardware platform. Therefore, the RM can be used to investigate whether a computational kernel can be further accelerated. We present an approach, based on the RM, to optimize the algorithmic parameters of dense linear algebra kernels. In particular, we perform a basic analysis to identify the optimal values for the kernel parameters. As a proof-of-concept, we apply this technique to optimize a blocked algorithm for matrix inversion via Gauss-Jordan elimination. In addition, we extend this technique to multi-block computational kernels. An experimental evaluation validates the method and shows its convenience. We remark that the results obtained can be extended to other computational kernels similar to Gauss-Jordan elimination such as, e.g., matrix factorizations and the solution of linear least squares problems.

Keywords: Roofline model · Dense linear algebra · Gauss-Jordan elimination

1 Introduction

Dense numerical linear algebra operations are crucial for the solution of a vast number of scientific computing applications. In response to this, highly tuned

All researchers acknowledge the support from the EHFARS project funded by the German Ministry of Education and Research BMBF.

E.S. Quintana-Ortí—Supported by the CICYT project TIN2014-53495-R of the *Ministerio de Economía y Competitividad* and FEDER.

basic numerical linear algebra subroutines (BLAS) [5], as well as more complex routines as those defined in LAPACK [1], have been developed and integrated into high performance libraries. There are several implementations of BLAS and/or LAPACK, usually specialized and maintained for different types of architectures by the processor manufacturer, such as IBM ESSL, Intel MKL or NVIDIA CUBLAS.

The *Roofline model* (RM) [16] is a graphical tool that can be leveraged to investigate the performance as well as identify the limiting factors of a computational kernel, including e.g. those in BLAS and LAPACK, executed in a given hardware architecture. Concretely, the RM consists of a two-dimensional chart that displays the (theoretical) peak memory bandwidth and performance of a platform, and relates these bounds to the *arithmetic intensity* (AI) of a computational kernel, defined as the ratio between floating-point arithmetic operations (flops) and memory accesses (memops) of the implementation.

In this paper we analyze the effect that AI exerts on the practical performance of blocked algorithms for dense matrix factorizations, such as those in LAPACK, making the following concrete contributions:

- We introduce a simple theoretical analysis to determine the *algorithmic blocksize* that reduces memops, optimizing AI and in general performance, of a blocked algorithm for matrix inversion via Gauss-Jordan elimination (GJE) [6].
- We extend this simple model to deal with more complex multi-block variants that improve AI for the inversion procedure.
- We provide a compact experimental analysis on a quadcore Intel processor to validate our findings.
- Finally, we remark that our study carries over, among others, to several other matrix factorization algorithms for the solution of linear systems and linear least squares problems [6].

The rest of the paper is structured as follows. In Sect. 2, we offer a brief review of the RM. Next, in Sect. 3 we revisit matrix inversion via GJE; and in Sect. 4 we introduce the analysis to compute the optimal algorithmic blocksize from the perspective of AI. Additionally, in that section we extend our study to a multi-block variant, which is a conventional technique to improve the performance of dense linear algebra factorization algorithms. In Sect. 5, we outline the experimental impact of the blocksizes previously, on a practical implementation. Finally, in Sect. 6 we summarize the results and emphasize a few concluding remarks derived from our work.

2 The Roofline Model

The RM separates the memory-bound and compute-bound “spaces” of an architecture as a function of AI. In particular, the model provides a two-dimensional easy-to-read chart that illustrates the crossover threshold between the peak

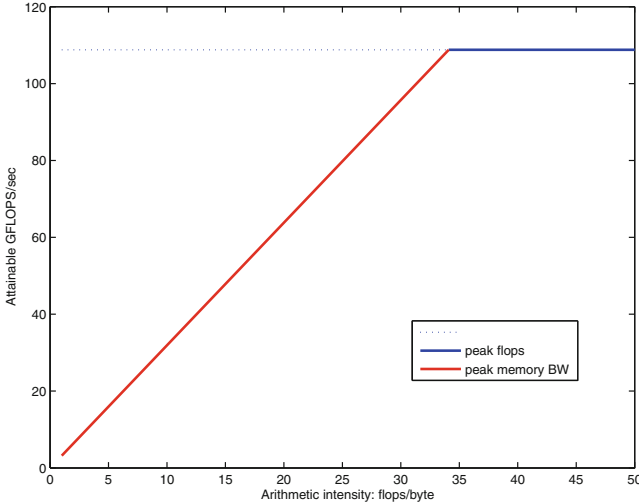


Fig. 1. The RM for an Intel Core i7-4770. The intersection between the red and blue lines identify the threshold between memory bandwidth (BW)-bound and compute-bound areas, as a function of the operation’s AI. (Color figure online)

memory bandwidth and performance (in terms of flops per second) of the hardware platform, showing the relation between the maximum performance attainable by the hardware and the AI of the computational kernel.

To create the model, the peak performance and memory bandwidth of the target system are needed. These figures are typically obtained from the hardware manufacturer, though it is also possible to use benchmarks to experimentally replace them with more realistic/practical values, see e.g. [14]. To illustrate this, Fig. 1 presents the RM for the hardware platform employed in the experimental evaluation in Sect. 5.

In order to position a computational kernel with respect to the bounds defined by the RM, it is necessary to determine the kernel’s AI. This can be computed from (estimations for) the total flops and memory accesses performed by the kernel. It should be noted that the RM is platform-specific, but can be re-used for any computational kernel executed in that system.

To summarize, the RM provides a helpful means to understand how the memory bandwidth constrains the performance, for memory-bounded algorithms, and/or identify how much an application can be accelerated (as the gap between the real and the attainable performance reported by the model). More details on the RM can be found in [8, 10, 15].

3 Matrix Inversion via GJE

Our general goal for this work is to exploit the principles underlying RM to improve the performance of dense linear algebra operations. As a proof-of-concept,

Algorithm: $[A] := \text{GJE_BLK}(A)$	
$A \rightarrow \left(\begin{array}{c c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right)$ where A_{TL} is 0×0 while $m(A_{TL}) < m(A)$ do Determine block size b $\left(\begin{array}{c c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right) \rightarrow \left(\begin{array}{c c c} A_{00} & A_{01} & A_{02} \\ \hline A_{10} & A_{11} & A_{12} \\ \hline A_{20} & A_{21} & A_{22} \end{array} \right)$ where A_{11} is $b \times b$	
$\begin{bmatrix} A_{01} \\ A_{11} \\ A_{21} \end{bmatrix} := \text{GJE_UNB} \left(\begin{bmatrix} A_{01} \\ A_{11} \\ A_{21} \end{bmatrix} \right)$ $A_{00} := A_{00} + A_{01}A_{10}$ $A_{20} := A_{20} + A_{21}A_{10}$ $A_{10} := A_{11}A_{10}$ $A_{02} := A_{02} + A_{01}A_{12}$ $A_{22} := A_{22} + A_{21}A_{12}$ $A_{12} := A_{11}A_{12}$	Unblocked Gauss-Jordan Matrix-matrix product Matrix-matrix product Matrix-matrix product Matrix-matrix product Matrix-matrix product Matrix-matrix product
$\left(\begin{array}{c c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right) \leftarrow \left(\begin{array}{c c c} A_{00} & A_{01} & A_{02} \\ \hline A_{10} & A_{11} & A_{12} \\ \hline A_{20} & A_{21} & A_{22} \end{array} \right)$ endwhile	

Fig. 2. Blocked algorithm for matrix inversion via GJE without pivoting.

we perform our study on the blocked algorithm for matrix inversion, based on GJE, described in this section.

GJE is an appealing method for matrix inversion, with a computational cost and numerical properties analogous to those of the conventional approach based on the LU factorization [9], but superior performance on a variety of architectures, from clusters [13] to general-purpose multicore processors and GPUs [3].

Figure 2 shows a blocked version of GJE for matrix inversion using the FLAME notation. There, $m(A)$ stands for the number of rows of the matrix A . More details on the notation can be found in [4, 7]. For a detailed description of the algorithm and the unblocked version of GJE, invoked from inside the blocked routine, see [2, 13]. For simplicity, we do not include the application of pivoting during the factorization, but details can be found there as well. Given a square (nonsingular) matrix of size $n = m(A)$, the cost of matrix inversion using this algorithm is $2n^3$ flops. Furthermore, the inversion is carried out in-place so that, upon completion, the entries of A are overwritten with those of its inverse.

At this point, we emphasize that the blocked algorithm in Fig. 2 casts most of its operations in terms of matrix-matrix products and other BLAS (inside the unblocked routine for GJE). Therefore, the conclusions from our intensity-performance analysis via RM in the next section can be also extended to several other dense linear algebra operations, such as the solution of linear systems via

the LU and Cholesky factorizations, as well as least-squares computations using the QR decomposition, among others [6].

4 Optimizing the Algorithmic Blocksize

4.1 General Discussion of High Performance for Dense Linear Algebra Routines

The usual approach to attain high performance for the execution of a dense linear algebra operation in a current architecture formulates the computation in the form of a blocked algorithm, where the bulk of the flops are computed via BLAS-3 operations such as, e.g., matrix-matrix products. This is motivated by the high performance offered by the BLAS-3 operations, due to their intrinsic parallelism and their convenient flops-to-memops ratio. Compared with this, the BLAS-1 and BLAS-2 kernels perform a number of flops of the same order as the volume of memory accesses, in general achieving a small fraction of the theoretical peak performance of a current general-purpose architecture. The performance attained by blocked algorithms strongly depends on the value of the algorithmic blocksize, b . This parameter determines how operations are distributed among the different kernels. Identify the best value for b is a complex task since it depends on the underlying hardware as well as on the computational kernel [11, 12].

4.2 Blocked Algorithm for GJE

In the particular case of the GJE, the use of a large algorithmic blocksize b concentrates most of the flops inside subroutine GJE_UNB, which is rich in BLAS-1 and BLAS-2 kernels. Consequently, the performance provided by the unblocked stages in GJE_UNB will dictate the performance of the whole algorithm. At the opposite extreme, the selection of a very small value for b transforms the BLAS-3 operations in GJE_BLK into quasi-BLAS-2 kernels (due to the reduced number of columns in the blocks of the form A_{x1}). Our aim is therefore to identify the value of b that maximizes the use of BLAS-3 operations, and thus minimizes the volume of memory accesses. Given an algorithm with a fixed computational cost, reducing the memops factor improves its AI (as the ratio between flops and memops), and generally the attained performance.

The main loop of GJE_BLK requires a total of n/b iterations (see Fig. 2), with each step requiring the computation of BLAS-3, BLAS-1/BLAS-2 kernels. Note that in general, BLAS-3 are compute-bound while BLAS-1/BLAS-2 are memory-bound. Concretely, the flops of each iteration are distributed as follows:

- BLAS-1 and BLAS-2: $2nb^2$ flops to factorize the panel, i.e. $[A_{01}; A_{11}; A_{21}]$.
- BLAS-3: $2n(n-b)b$ flops to update the rest of the matrix.

Now, assuming that BLAS-1 and BLAS-2 kernels perform $O(1)$ flops per memop while BLAS-3 kernels perform $O(b)$ flops per memop, the total number of memory accesses needed by GJE is approximately:

$$n/b(2n(n-b) + 2nb^2), \quad (1)$$

which can be simplified into:

$$2n^2 (n/b - 1 + b) \text{ memops.} \quad (2)$$

If we consider communication (memops) as overhead, finding the optimal blocksize b^{opt} is then equivalent to minimizing the number of memops given by (2). Therefore, we just need to find the root(s) of the derivative function of (2) with respect to b in order to obtain $b^{opt} = \sqrt{n}$.

Moreover, the arithmetic intensity of the GJE_BLK is then given by

$$\frac{2n^3}{2n^2 (n/b^{opt} - 1 + b^{opt})} \text{ flops-per-memop} \quad (3)$$

and, as $b^{opt} = \sqrt{n}$, the “best” arithmetic intensity we can attain with our blocked algorithm for matrix inversion via GJE_BLK is

$$\frac{n}{2\sqrt{n} - 1} \approx \frac{\sqrt{n}}{2} \text{ flops-per-memop.} \quad (4)$$

4.3 Multi-block Variant of GJE

In this section we describe a multi-block strategy to accelerate GJE, and how to extend the analysis based on the RM/AI in order to identify the optimal blocksizes for this variant.

The Multi-block GJE partly casts the operations involved by the panel factorization in terms of BLAS-3 kernels, in order to further increase the number of flops performed using this type of kernels in the algorithm. For this purpose, in the multi-block version of the algorithm, subroutine GJE_UNB is replaced by a slightly modified version of GJE_BLK that can operate with rectangular matrices. As a result, the multi-block variant of GJE is parametrized by two blocksizes: the outer blocksize b , applied during the execution of the blocked algorithm, and the inner blocksize c , employed during the factorization of the panel. For simplicity, hereafter we will assume that b is an integer multiple of c .

Leveraging RM to Select the Optimal Blocksizes. Using a similar strategy to that presented in Sect. 4.1, we can infer the optimal values for both blocksizes and use them to establish the best arithmetic intensity attainable by the multi-block variant of the GJE algorithm.

In this case, the flops performed during an iteration of the main loop can be decomposed into the following three terms:

- BLAS-1 and BLAS-2: $2nc^2$ flops to factorize the panel (note that this factorization itself requires b/c steps).
- BLAS-3: $2n(b-c)c$ flops to update the elements within the panel.
- BLAS-3: $2n(n-b)b$ flops to update the rest of the matrix.

Let us assume that c offers a rough measure of the relation between the flops and memory accesses for the BLAS-3 operations executed inside the panel, and let b be its counterpart for the BLAS-3 operations to update the elements placed out of the panel. Consider again that the BLAS-1 and BLAS-2 kernels perform $O(1)$ flops per memory access, while the ratio for the BLAS-3 is $O(b)$. Then, the outer loop is executed n/b times, while the inner loop b/c times per step of the outer loop; and the total number of memops of the multi-block variant is

$$n/b(b/c(2nc + 2n(b-c)) + 2n(n-b)), \quad (5)$$

which can be simplified to

$$2n^2(c + b/c + n/b - 2) \text{ memops.} \quad (6)$$

Differentiating the previous expression with respect to c , and finding the roots of the result, we obtain that the value of the inner blocksize that minimizes the number of memory accesses is $c^{opt} = \sqrt{b}$. This is natural as the computation performed by the inner loop is analogous to the application of a “rectangular” version of the blocked algorithm for GJE to a matrix of dimension $n \times b$.

Replacing c by its optimal value c^{opt} , in Eq. (6), we then obtain:

$$2n^2(n/b - 2 + 2\sqrt{b}) \text{ memops.} \quad (7)$$

Similarly, if we derive Eq. (7) with respect to b , and equate the result to zero, we obtain the value of b that minimizes the number of memory accesses as $b^{opt} = (\sqrt[3]{n})^2$.

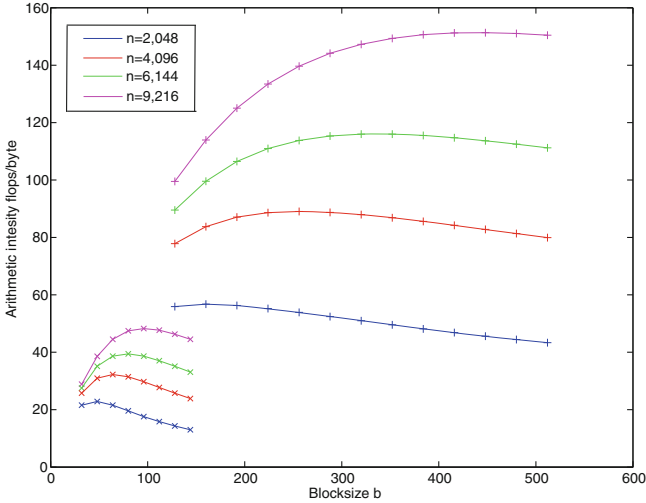


Fig. 3. Effect of the external blocksize b on AI. Lines with marks “ \times ” and “ $+$ ” represent the AI for the blocked and the multi-block algorithm respectively. For the multi-block algorithm, $c = c^{opt} = \sqrt{b}$.

Finally, the arithmetic intensity of the multi-block variant of GJE is given by

$$\frac{2n^3}{2n^2(n/b - 2 + b/c + c)} \text{ flops-per-memop}, \quad (8)$$

and, in the case of b^{opt} and c^{opt} ,

$$\frac{n}{n/b - 2 + 2\sqrt{b}} = \frac{n}{3\sqrt[3]{n} - 2} \approx \frac{(\sqrt[3]{n})^2}{3} \text{ flops-per-memop}. \quad (9)$$

To close this section, Fig. 3 illustrates the effect of the (external) blocksize b on the AI of the blocked GJE-algorithm for matrix inversion and its multi-block version, clearly identifying the existence of optimal values for both algorithms, and the much higher AI of the multi-level variant.

5 Experimental Evaluation

The experiments in this section were performed on an Intel-based server equipped with an Intel Core i7-4770 processor (4 cores operating at 3.40 GHz) using double precision (DP) floating-point arithmetic. The (theoretical) peak floating-point rate of this hardware platform is 108.8 DP GFLOPS (billions of flops/sec) and the (theoretical) peak bandwidth is 25.5 GB/s (i.e., 3.18 millions of DP numbers per second). All the implementations rely on the multi-threaded implementation of BLAS provided by Intel MKL 11.1, and the experiments are configured to exploit all 4 cores in the platform by spawning 4 threads during the execution of the BLAS.

We first carry out an experiment that aims to empirically assess the impact of the blocksize on the performance of the blocked implementation of the GJE method to invert matrices of four dimensions. To avoid variations due to cache dimensions and associativity, we select $n=2,048$, 4,096, 6,144 and 9,216. For brevity, and to better exploit the processor’s vector units, we only experiment with “spaced” values of b that are integer multiples of $s=32$ (except for the 2,048 case, where we use integer multiples of $s=16$). For each matrix dimension, the theoretical optimal blocksize b^{opt} is computed as described in Sect. 4.1. We then test three different values for b , corresponding to the two integer multiples of s closer to b^{opt} above it (i.e., $(\lfloor b^{opt}/s \rfloor + 1)s$ and $(\lfloor b^{opt}/s \rfloor + 2)s$) as well as the closest integer multiple below this value $(\lfloor b^{opt}/s \rfloor s)$.

Table 1 displays the results obtained for this initial study, showing the value of b^{opt} for each matrix dimension and the performance (in GFLOPS) attained using the three values selected for b . The best performance is always observed for the value of b closest to b^{opt} , validating our formula to determine the optimal blocksize setting.

In addition, the performance observed for the implementation of GJE_BLK grows with the dimension of the matrix, a result that is also aligned with the theoretical study, as the computational intensity is proportional to b and larger matrices demand larger blocksizes.

Table 1. Performance (in GFLOPS) of GJE_BLK to invert matrices of different dimensions using several blocksizes b .

Matrix dimension	b^{opt}	b	GFLOPS
2,048	45	32	38
		48	40
		64	40
4,096	64	32	45
		64	52
		96	51
6,144	78	64	62
		96	76
		128	75
9,216	96	64	78
		96	102
		128	98

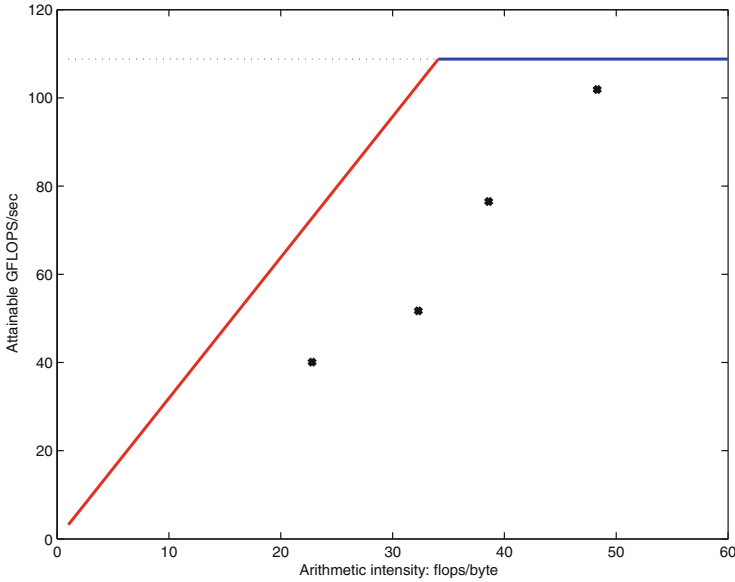


Fig. 4. RM applied to the inversion of matrices via GJE on an Intel Core i7-4770. (Color figure online)

Figure 4 relates performance/AI of the GJE kernel with the parameters of RM for the target architecture. The position in the x-axis is calculated using Eq. (3), and the black dots show the performance attained with the optimal value of b for each problem dimension.

Table 2. Performance (in GFLOPS) reported by the multi-block GJE variant.

Matrix dimension n	$b^{opt} - c^{opt}$	$b - c$	GFLOPS	Arithmetic intensity
2,048	161–12	160–16	62	56
4,096	256–16	256–16	69	89
6,144	335–18	320–16	82	115
9,216	406–20	384–16	94	149

Considering the results in the figure, we point out that the increment in AI is accompanied with improvements in the actual performance. However, the values calculated for AI seem to be overestimated, as the line connecting the black dots shows a gradient similar to that of the bandwidth limit (red line), but shifted in the x-axis.

We next evaluate the multi-block version of GJE, described in Sect. 4.3, to identify the optimal value for the two block sizes: b and c . Table 2 presents the theoretical optimal values for these parameters, the actual values tested for the block sizes, the performance attained (in GFLOPS), and the AI according to Eq. (8).

An inspection of the results in the table reveals that the use of a multi-block technique is especially effective for the inversion of matrices of moderate dimension. Concretely, the multi-block algorithm increases the performance by 50% for the smallest problem but it is slightly slower for the largest problem. This is because this technique aims to reduce the impact of the memory-bound operations, a hazard that has a stronger effect for small- to moderate-size problems. In particular, when $n = 9,216$, the computation is not memory-bound and, therefore, the multi-block technique does not yield any gain. Additionally, the blocked algorithm employs the optimal blocksize while suboptimal block sizes are employed by the multi-block algorithm (due to the multiple of 32 restriction).

In practice, even though the AI factors show that the performance should be limited by the peak performance of the system, in practice it is limited by the memory bandwidth. This is a sign that the theoretical model overestimates the actual AI.

6 Concluding Remarks and Future Work

The Roofline model offers a measure of the optimization potential of a computational routine, relating its AI to the theoretical peak memory bandwidth and peak performance of the target architecture. For dense linear algebra factorization methods, blocked algorithms aim to improve performance by casting a significant fraction of its computations in terms of efficient, compute-bound BLAS-3 kernels that are only constrained by the processor’s peak GFLOPS rate. A key parameter to optimize these algorithms is the blocksize, which determines the fraction of the flops that are computed as BLAS-3 vs. BLAS-1/2 kernels and, therefore, governs the performance of the global algorithm.

In this paper we have presented simple yet accurate models to determine the blocksize that optimizes AI for a blocked matrix inversion algorithm based on GJE. Furthermore, we have extended the formulation to a multi-level variant that delivers even higher rates of AI. Our experimental results in an Intel processor with four cores validates the approach, showing that the increases in AI actually result in a performance improvement for both the original blocked algorithm and its multi-level counterpart.

In the future we plan to apply the same techniques to other dense linear algebra algorithms and platforms. We also intend to obtain more precise formulas for the arithmetic intensity.

References

1. Anderson, E., Bai, Z., Demmel, J., Dongarra, J.E., DuCroz, J., Greenbaum, A., Hammarling, S., McKenney, A.E., Ostrouchov, S., Sorensen, D.: LAPACK Users' Guide. SIAM, Philadelphia (1992)
2. Benner, P., Ezzatti, P., Quintana-Ortí, E.S., Remón, A.: Unleashing CPU-GPU acceleration for control theory applications. In: Caragiannis, I., et al. (eds.) EuroPar 2012. LNCS, vol. 7640, pp. 102–111. Springer, Heidelberg (2013). doi:[10.1007/978-3-642-36949-0_13](https://doi.org/10.1007/978-3-642-36949-0_13)
3. Benner, P., Ezzatti, P., Quintana-Ortí, E.S., Remón, A.: Matrix inversion on CPU-GPU platforms with applications in control theory. *Concurrency Comput. Pract. Exp.* **25**(8), 1170–1182 (2013)
4. Bientinesi, P., Gunnels, J.A., Myers, M.E., Quintana-Ortí, E.S., van de Geijn, R.A.: The science of deriving dense linear algebra algorithms. *ACM Trans. Math. Softw.* **31**(1), 1–26 (2005)
5. Dongarra, J.J., Du Croz, J., Hammarling, S., Duff, I.: A set of level 3 basic linear algebra subprograms. *ACM Trans. Math. Softw.* **16**(1), 1–17 (1990)
6. Golub, G.H., Van Loan, C.F.: Matrix Computations, 3rd edn. The Johns Hopkins University Press, Baltimore (1996)
7. Gunnels, J.A., Gustavson, F.G., Henry, G.M., van de Geijn, R.A.: FLAME: formal linear algebra methods environment. *ACM Trans. Math. Softw.* **27**(4), 422–455 (2001)
8. Hennessy, J.L., Patterson, D.A.: Computer Architecture: A Quantitative Approach. Elsevier, London (2011)
9. Higham, N.J.: Accuracy and Stability of Numerical Algorithms, 2nd edn. Society for Industrial and Applied Mathematics, Philadelphia (2002)
10. Lo, Y.J., Williams, S., Straalen, B., Ligocki, T.J., Cordery, M.J., Wright, N.J., Hall, M.W., Olike, L.: Roofline model toolkit: a practical tool for architectural and program analysis. In: Jarvis, S.A., Wright, S.A., Hammond, S.D. (eds.) PMBS 2014. LNCS, vol. 8966, pp. 129–148. Springer, Heidelberg (2015). doi:[10.1007/978-3-319-17248-4_7](https://doi.org/10.1007/978-3-319-17248-4_7)
11. Mehta, S., Garg, R., Trivedi, N., Yew, P.: TurboTiling: leveraging prefetching to boost performance of tiled codes. In: Proceedings of the 2016 International Conference on Supercomputing, ICS 2016, New York, NY, USA, pp. 38:1–38:12. ACM (2016)
12. The ELAPS framework: <http://hpac.rwth-aachen.de/~peise/elaps>. High Performance and Automatic Computing group at RWTH-Aachen University

13. Quintana-Ortí, E.S., Quintana-Ortí, G., Sun, X., van de Geijn, R.A.: A note on parallel matrix inversion. *SIAM J. Sci. Comput.* **22**, 1762–1771 (2001)
14. Talagala, N., Arpaci-Dusseau, R.H., Patterson, D.A.: Micro-benchmark based extraction of local and global disk characteristics. *Citeseer* (1999)
15. Unat, D., Chan, C., Zhang, W., Williams, S., Bachan, J., Bell, J., Shalf, J.: ExaSAT: an exascale co-design tool for performance modeling. *Int. J. High Perform. Comput. Appl.* **29**(2), 209–232 (2015)
16. Williams, S., Waterman, A., Patterson, D.: Roofline: an insightful visual performance model for multicore architectures. *Commun. ACM* **52**(4), 65–76 (2009)

Algorithms and Architectures for Parallel Processing
ICA3PP 2016 Collocated Workshops: SCDT, TAPEMS,
BigTrust, UCER, DLMCS, Granada, Spain, December
14-16, 2016, Proceedings

Carretero, J.; Garcia-Blas, J.; Gergel, V.; Voevodin, V.;
Meyerov, I.; Rico-Gallego, J.A.; Díaz-Martín, J.C.; Alonso,
P.; Durillo, J.; Garcia Sánchez, J.D.; Lastovetsky, A.L.;
Marozzo, F.; Liu, Q.; Bhuiyan, M.Z.A.; Furlinger, K.;
Weidendorfer, J.; Gracia, J. (Eds.)

2016, XXV, 384 p. 126 illus., Softcover

ISBN: 978-3-319-49955-0