Performance estimation of linear algebra numerical libraries

Alessandro de Rosis

Alessandro De Rosis*

Performance estimation of linear algebra numerical libraries

Abstract: In this work, numerical algebraic operations are performed by using several libraries whose algorithm are optimized to drain resources from hardware architecture. In particular, dot product of two vectors and the matrix-matrix product of two dense matrices are computed. In addition, the Cholesky decomposition on a real, symmetric, and positive definite matrix is performed through routines for band and sparse matrix storage. The involved CPU time is used as an indicator of the performance of the employed numerical tool. Results are compared to naive implementations of the same numerical algorithm, highlighting the speed-up due to the usage of optimized routines.

Keywords: Numerical linear algebra, finite element method, lattice Boltzmann method, high performance computing.

MSC 2010: 15A23, 74S05

DOI: 10.1515/jnma-2015-0002
Received October 14, 2012; accepted August 13, 2013

1 Introduction

The typical problems of structural mechanics are governed by differential equations. In this framework, the finite element method (FEM) [5] arose as a technique which allows to transform such kind of problems in ones governed by algebraic equations, whose solution can be addressed to a computer. The more sophisticated the analysis, the higher the computational cost is. Therefore, the usage of a computer becomes an essential part of the analysis process. Herein, several computational tools are be tested in order to assess the ability to improve the performance of a numerical FEM software. Such tools are known as linear algebra libraries, i.e. a set of routines performing matrix-vector computations, the solution of linear systems and eigenvalue problems, which are optimized to fully exploit hardware resources, that is expected to play a crucial role in the overall performance of a numerical software. In this context, the continuous differential problem becomes a discretized algebraic one described by matrices and vectors, simple to handle numerically by using a class of libraries able to solve matrix-matrix operations, linear systems, and eigenvalue problems; thus, the development of sophisticated linear algebra routines plays a crucial role.

One supposes to perform the matrix-matrix product of two square matrices of size n: the complexity of the problem, in terms of the number of elementary operations to be computed, amounts to $2n^3$. One would think that on the same machine this operation, carried out by tools using the identical algorithm, will last the same duration, but this is not true. In contrast to a naive implementation of an algorithm, these libraries are able to exploit the technology of the computer and then draw resources on the particular computer architecture, thus saving the duration of the analysis.

On the other hand, the developments in the technology results in the creation of microprocessors faster than the main memory; thus, the access to the memory represents a real bottleneck in the overall performance of an application. In 1970, Moore [25] suggested that the CPU speed would grow much more, while the RAM speed would be almost constant in the next thirty years. Consequently, an application can waste a lot of time waiting for data, involving a negative impact on the overall performance, and in addition preventing the exploitation of the high speed of the CPU.

***Corresponding Author: Alessandro De Rosis:** Laboratoire de Mécanique des Fluides et d'Acoustique (LMFA), École Centrale de Lyon, 36 avenue Guy de Collongue, 69134 Écully cedex, France. Email: alessandro.de-rosis@ec-lyon.fr

The purpose of this work is to demonstrate how routines that can use efficiently hardware technology will show better performances, even if they employ the same algorithm of a naive implementation, [1, 7, 16, 18]. For the numerical test campaign, a machine equipped with Intel Core i5-2400 3.40 GHz, L2 6 MB cache, 4 GB RAM, and the OS Ubuntu 12.04, has been used.

This paper is organized as follows. In Section 2, the theoretical/computational background is discussed. In Section 3, performances of numerical libraries are presented. Finally, Section 4 provides concluding remarks.

2 Numerical methods

In this section, the basic idea concerning the matrix-matrix product and the Cholesky decomposition are discussed, thus giving an idea about the number of involved floating point operations that a numerical algorithm performs. In addition, the methodology used to depict a performance profile is presented.

2.0.1 The matrix-matrix product

Let **A** and **B** be two matrices to be multiplied. Such matrices are defined as follows

$$A = \begin{bmatrix} A_{11} & A_{12} & \cdots & A_{1m} \\ A_{21} & A_{22} & \cdots & A_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ A_{n1} & A_{n2} & \cdots & A_{nm} \end{bmatrix}, \qquad B = \begin{bmatrix} B_{11} & B_{12} & \cdots & B_{1p} \\ B_{21} & B_{22} & \cdots & B_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ B_{m1} & B_{m2} & \cdots & B_{mp} \end{bmatrix} \tag{2.1}$$

where **A** and **B** possess $n \times m$ and $m \times p$ dimensions, respectively. The product matrix **C** of dimensions $n \times p$ is computed as

$$\mathbf{C}_{ij} = \sum_{k=1}^{m} \mathbf{A}_{ik}\mathbf{B}_{kj}. \tag{2.2}$$

If **A** and **B** are both square matrices of order n, $2n^3$ operations are required to compute the product matrix **C**. Notice that the matrix-vector product between the matrix **A** and the vector **b** is a specific case of the matrix-matrix one, where **A** is a $m \times n$ matrix, **b** is a column vector **c** of length n (or a matrix of dimension $n \times 1$) and the output is a vector of length n. If **A** is a $n \times n$ matrix, the number of involved operations amounts to $2n^2$. In addition, if **a** and **b** are two vectors of length n, the dot product of **a** and **b** is a scalar quantity c obtained by multiplying corresponding entries and then summing those products:

$$c = \sum_{i=1}^{n} \mathbf{a}_i\mathbf{b}_i. \tag{2.3}$$

A number of elementary operations equal to $2n$ is performed in order to compute the dot product of two vectors.

2.1 The Cholesky decomposition

The Cholesky decomposition is the factorization of a Hermitian, positive-definite matrix **A** into the product of a lower triangular matrix **L** and its conjugate transpose $\mathbf{L}^+$:

$$\mathbf{A} = \mathbf{L}\mathbf{L}^+, \qquad \mathbf{A} \in \mathbb{K}^{m\times m}. \tag{2.4}$$

If **A** is real and symmetric, the conjugate transpose is equal to the transpose $\mathbf{L}^T$:

$$\mathbf{A} = \mathbf{L}\mathbf{L}^T, \qquad \mathbf{A} \in \mathbb{R}^{n\times n}. \tag{2.5}$$

This algorithm starts with

$$\mathbf{A}^{(1)} = \mathbf{A} \tag{2.6}$$

$$\mathbf{A}^{(i)} = \begin{pmatrix} A_{i,i} & \mathbf{b}_i^\star \\ \mathbf{b}_i & \mathbf{B}^{(i)} \end{pmatrix} \tag{2.7}$$

$$\mathbf{L}_i = \begin{pmatrix} \frac{1}{\sqrt{A_{i,i}}} & 0 \\ -\frac{1}{A_{i,i}}\mathbf{b}_i & \mathbf{I} \end{pmatrix} \tag{2.8}$$

$$\mathbf{A}^{(i)} = \mathbf{L}_i^{-1} \begin{pmatrix} 1 & 0 \\ 0 & \mathbf{B}^{(i)} - \frac{1}{A_{i,i}}\mathbf{b}_i\mathbf{b}_i^\star \end{pmatrix} (\mathbf{L}_i^{-1})^\star. \tag{2.9}$$

Then, in the next steps:

$$\mathbf{A}^{(i+1)} = \mathbf{B}^{(i)} - \frac{1}{A_{i,i}}\mathbf{b}_i\mathbf{b}_i^\star \tag{2.10}$$

$$\mathbf{A}^{(i)} = \mathbf{L}_i^{-1} \begin{pmatrix} 1 & 0 \\ 0 & \mathbf{A}^{(i+1)} \end{pmatrix} (\mathbf{L}_i^{-1})^\star. \tag{2.11}$$

Iterations end after n steps when $\mathbf{A}(n) = 1$. The lower triangular matrix $\mathbf{L}$ is calculated as

$$\mathbf{L} = \mathbf{L}_1\mathbf{L}_2 \ldots \mathbf{L}_n. \tag{2.12}$$

The Cholesky–Crout algorithm starts from the upper left corner of the matrix $\mathbf{L}$ and proceeds to calculate the matrix column by column:

$$L_{i,i} = \sqrt{A_{i,i} - \sum_{k=1}^{i-1} L_{i,k}^2}, \qquad i = 1, \ldots, m \tag{2.13}$$

$$L_{j,i} = \frac{1}{L_{i,i}} \left(A_{j,i} - \sum_{\iota=1}^{i-1} L_{j,\iota}L_{i,\iota} \right), \qquad j = i+1, \ldots, m. \tag{2.14}$$

2.2 Evaluation of the performance profile

Let $\mathbf{S}$ be a set of solvers whose performances have to be estimated on a set of problem $\mathbf{T}$ by monitoring one or more informations. A parameter $s_{ij} \geqslant 0$ is related to the solver $i \in \mathbf{S}$ when it is applied to the problem $j \in \mathbf{T}$. The lower is s_{ij}, the better we can consider the solver i. For all the set $\mathbf{T}$, the performance of the solver i is compared to the best solver in $\mathbf{S}$. Let $\hat{s}_j = \min\{s_{ij}; i \in \mathbf{S}\}$. Then, for $\alpha \geqslant 1$ and for each $i \in \mathbf{S}$ it is possible to define

$$k(s_{ij}, \hat{s}_j, \alpha) = \begin{cases} 1, & s_{ij} \leq \alpha\hat{s}_j \\ 0, & \text{otherwise.} \end{cases} \tag{2.15}$$

The performance profile of the solver i is given by the expression

$$p_i(\alpha) = \frac{\sum_{j\in\mathbf{T}} k(s_{ij}, \hat{s}_j, \alpha)}{|\mathbf{T}|}, \quad \alpha \geqslant 1. \tag{2.16}$$

The quantity $p_i(1)$ gives the fraction of examples for which the solver i is more efficient, in terms of the s_{ij} parameter. Such parameter is identified in the involved CPU time.

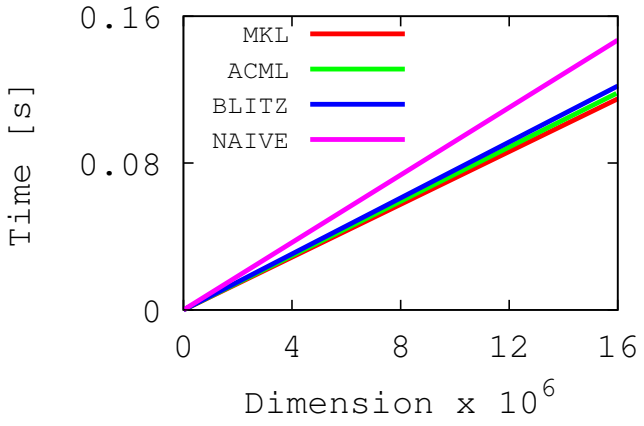

Figure 1. Dot product: CPU time [s] vs. dimension.

3 Results

In this section, a set of computational tools is tested in order to assess their performances on solving dot product and matrix-matrix product. Matrices characterized by different shapes and dimensions are used and the employed routines are designed to compute operations between generic dense matrices. Results are given in terms of involved CPU time and gigaflops. Moreover, the Cholesky decomposition, that is a very frequent in computational mechanics, is computed. Such operation usually requires a large amount of time in a FEM analysis. In this case, the employed routines are ad-hoc designed to handle real, symmetric and positive definite matrices. Banded and sparse storage techniques are used. Finally, the performance profile is estimated by using the procedure in [13].

3.1 The dot product

Here, the dot product of two vectors is computed. Since $2n$ operations are involved, it is expected a linear relation between the CPU time and the size of the vectors, being n the size of the vector. This is assessed in Fig. 1, where the CPU time in seconds is plotted against the dimension. Different libraries are used. The ACML and MKL routines [3] are two variants of the same algorithm: the former is optimized for AMD processors, the latter for Intel ones. The BLITZ [2] routine is ad hoc designed for vector-vector manipulation. In contrast to optimized algorithm, a naive implementation of the dot product is also depicted. Figure 1 shows that the MKL has the best performance, since an Intel processor equips the employed computer. On the other hand, the naive implementation is the worst solution, since it is unable to exploit hardware resources.

3.2 The matrix-matrix product

The product between two square matrices of dimension $n \times n$ is performed. In this case, the number of involved operations amounts to $2n^3$. In Fig. 2, the CPU time is depicted against the dimension n and a cubic relation is experienced. The ACML and MKL show the best performances. In particular, the MKL routine involves the lower CPU time due to the reason above assessed. The EIGEN library [14] possesses good performance, while BLITZ performs worse the naive implementation, since such library is designed only to handle vectors. Such results are confirmed in Fig. 3, where the number of operation ($\times 10^9$) per second is depicted against the dimension n, both in a logarithmic scale.

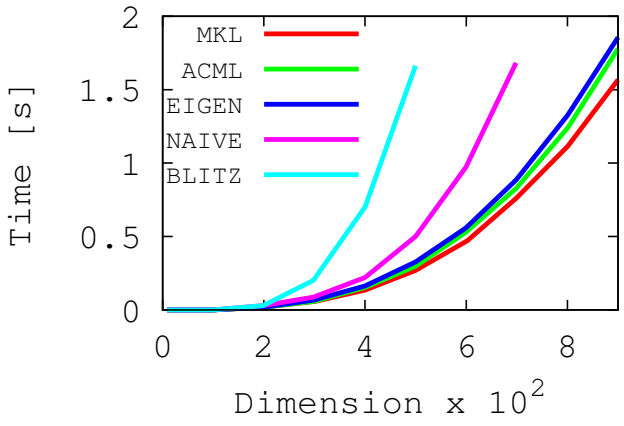

Figure 2. Matrix-matrix product: CPU time [s] vs. dimension.

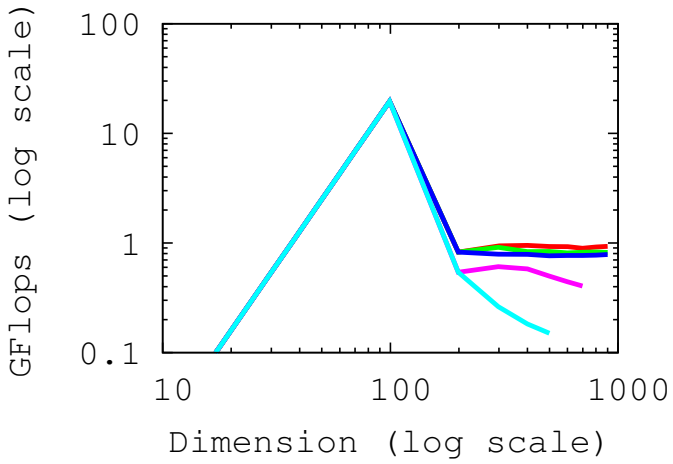

Figure 3. Matrix-matrix product: GFlops vs. dimension in a log-log scale.

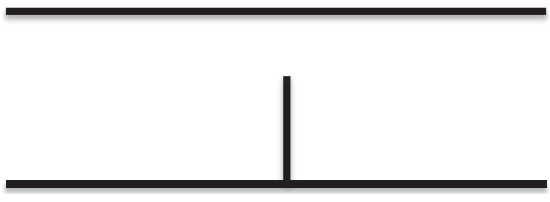

Figure 4. Physical problem.

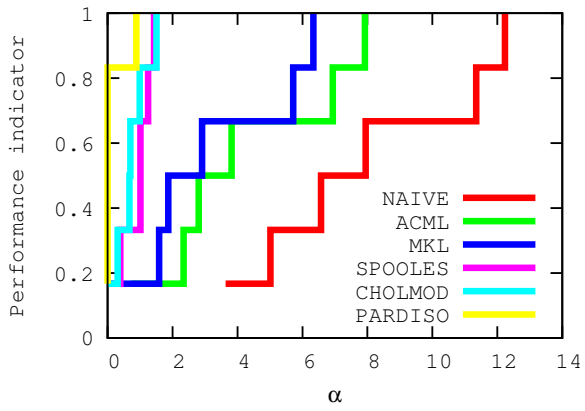

Figure 5. Performance profiles.

3.3 The solution of a linear system

Herein, the solution of the linear system $\mathbf{A} \cdot \mathbf{x} = \mathbf{b}$ is tested, where $\mathbf{A}$ is a matrix of coefficients to be decomposed, $\mathbf{x}$ is the vectors of the unknowns and the right-hand side vector $\mathbf{b}$ is known, [10, 21, 22, 24]. Again, the ACML and MKL routines are two variants of the same algorithm: the former is optimized for AMD processors, the latter for Intel ones. In addition, a naive implementation of the Cholesky–Crout decomposition of a banded matrix, which is not optimized respect to hardware resources, has been implemented. Moreover, the following solvers for sparse storage are used: CHOLMOD [11], SPOOLES [4], and PARDISO [26]. The stiffness matrix of a cantilever beam depicted in Fig. 4 is used in the factorization process. Such cantilever is invested by a viscous fluid.

It is a classical problem of fluid-structure interaction [12, 15, 19, 20], where the fluid is solved through the lattice Boltzmann method, that is known to be a fast and powerful computational tool [6, 8, 9, 27]. The structure solver uses a finite element approach, whose limit is the higher the number of degrees of freedom, the higher the computational cost is. To avoid that the computational effort of the structural solver can affect the overall analysis, a good choice is to use an optimized library in such FEM process. To achieve the dynamics of the cantilever beam, the approach proposed in [23] is adopted. Different matrices are obtained due to different mesh refinements. The performances of different libraries is depicted in Fig. 5, where it is shown that the best solution is to adopt a sparse solver. In particular, PARDISO is the most performant tool, since it is designed for an Intel computer. On the other hand, the very poor performance of the naive implementation of the Cholesky–Crout decomposition is assessed. Routines designed for banded storage represent an effective alternative to the naive implementation, as already affirmed in the previous computations.

4 Conclusions

This work aimed to underline that a naive implementation of an algorithm is an obsolete choice. In fact, the need of increasingly refined and sophisticated analyzes leads to a higher amount of the computational cost; thus, the modern programming techniques can not be separated from the usage of numerical tools optimized respect the exploitation of hardware resources. As shown, the speed-up can be orders of several times larger. Such wide difference is due to the manner in which, through appropriate implementations, the CPU accesses the data it needs.

First, the dot product of two vectors has been investigated. The MKL routine is the most performant routine, since an Intel processor has been used. Notice also the effectiveness of the BLITZ library. Such tool showed very poor performances in the second test, the matrix-matrix product, since it is designed only for vector manipulations. The MKL is confirmed to be the best choice. Finally, performances of numerical libraries in factorizing a matrix through the Cholesky decomposition have been checked. Such operation is as common as expensive in a FEM code. Usually, the matrix to be decomposed is real, symmetric, positive definite and

shows a band pattern. Solvers developed for band and sparse storage have been employed and results show the most performant tool is PARDISO, since an Intel processor has been employed. Thus, it can be stated that the adoption of such sparse solver can be taken into consideration even if the matrix exhibits a band pattern, since it demonstrated performances superior than solvers designed ad-hoc for a band storage. In addition, the MKL library is the best choice among the band solvers, since Intel machine and processor is used.

Therefore, it is possible to assess that the modern programming techniques benefit from the adoption of effective numerical tools for solving a numerical algebraic problem, by draining hardware resources. A future development is to employ the MTL4 library [17], providing an easy and intuitive interface and enabling optimal performance that is expected to improve much more the performances of a FEM software [16].

References

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

[2] D. Abrahams and A. Gurtovoy, *C++ Template Metaprogramming: Concepts, Tools, and Techniques from Boost and Beyond (C++ in Depth Series)*, Addison-Wesley Professional, 2004.

[3] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. Mckenney, and D. Sorensen, *LAPACK Users' Guide (Software, Environments and Tools)*, Society for Industrial and Applied Mathematics, 1987.

[4] C. Ashcraft, *Spooles 2.2: SParse Object Oriented Linear Equations Solver*, 2012.

[5] K. J. Bathe, *Finite Element Procedures*, Prentice Hall, 1996.

[6] R. Benzi, S. Succi, and M. Vergassola, The lattice Boltzmann equation: theory and applications, *Phys. Reports* **222** (1992), 145–197.

[7] A. Buttari, J. Dongarra, J. Kurzak, J. Langou, P. Luszczek, and S. Tomov, The impact of multicore on math software, *Sci. New York* **4699** (2007), 1–10.

[8] H. Chen, S. Chen, and W. H. Matthaeus, Recovery of the Navier-Stokes equations using a lattice-gas Boltzmann method, *Phys. Review Letters* **45** (1992), R5339–R5342.

[9] S. Chen and G. D. Doolen, Lattice Boltzmann method for fluid flows, *Annual Review Fluid Mech.* **30** (1998), 329–364.

[10] T. A. Davis, *Direct methods for sparse linear systems*, Society for Industrial and Applied Mathematics, 2006.

[11] T. A. Davis, *User Guide for Cholmod: a Sparse Cholesky Factorization and Modification Package*, 2012.

[12] A. De Rosis, G. Falcucci, S. Ubertini, F. Ubertini, and S. Succi, Lattice Boltzmann analysis of fluid-structure interaction with moving boundaries, *Commun. Comp. Phys.* **13** (2012), 823–834.

[13] E. D. Dolan and J. J. Moré, Benchmarking optimization software with performance profiles, *Math. Programming* **91** (2002), 201–213.

[14] EIGEN library. http://eigen.tuxfamily.org/dox/

[15] G. Falcucci, M. Aureli, S. Ubertini, and M. Porfiri, Transverse harmonic oscillations of laminae in viscous fluids: a lattice Boltzmann study, *Philosoph. Trans. Royal Soc. - Series A* **369** (2011), 2456–2466.

[16] P. Gottschling and D. Lindbo, Generic compressed sparse matrix insertion: algorithms and implementations in MTL4 and FEniCS, *Matrix* (2009), 0–7.

[17] P. Gottschling and C. Steinhardt, Meta-tuning in MTL4, *Physics* **1281** (2010), 778–782.

[18] K. Kaspersky, *Code Optimization: Effective Memory Usage*, A-List Publishing, 2003.

[19] S. Kollmannsberger, S. Geller, A. Duster, J. Tolke, C. Sorger, M. Krafczyk, and E. Rank, Fixed-grid fluid-structure interaction in two dimensions based on a partitioned lattice Boltzmann and *p*-FEM approach, *Int. J. Numer. Meth. Engrg.* **79** (2009), 817–845.

[20] E. Leriche, P. Lallemand, and G. Labrosse, Stokes eigenmodes in cubic domain: primitive variable and lattice Boltzmann formulations, *Appl. Numer. Math.* **58** (2008), 935–945.

[21] W. M. Lioen and D. T. Winter, Solving large dense systems of linear equations on systems with virtual memory and with cache, *Appl. Numer. Math.* **10** (1992), 73–85.

[22] M. Louter-Nool, Block-Cholesky for parallel processing, *Appl. Numer. Math.* **10** (1992), 37–57.

[23] M. Mancuso and F. Ubertini, An efficient Time Discontinuous Galerkin procedure for non-linear structural dynamics, *Comp. Methods Appl. Mech. Engrg.* **195** (2006), 6391–6406.

[24] C. Meszaros, Fast Cholesky factorization for interior point methods of linear programming, *Computers & Math. Appl.* **31** (1996), 49–54.

[25] E. Mollick, *Establishing Moore Law*, 2006, pp. 62–75.

[26] O. Schenk, *Pardiso Solver Users Guide*, 2012.

[27] S. Succi, *The Lattice Boltzmann Equation for Fluid Dynamics and Beyond*, Clarendon, 2001.