J Supercomput (2014) 70:564–576
DOI 10.1007/s11227-013-1083-8

Parallel approach to NNMF on multicore architecture

P. Alonso · V. M. García · F. J. Martínez-Zaldívar · A. Salazar · L. Vergara · A. M. Vidal

Published online: 22 January 2014

Abstract We tackle the parallelization of Non-Negative Matrix Factorization (NNMF), using the Alternating Least Squares and Lee and Seung algorithms, motivated by its use in audio source separation. For the first algorithm, a very suitable technique is the use of active set algorithms for solving several non-negative inequality constraints least squares problems. We have addressed the NNMF for dense matrix on multicore architectures, by organizing these optimization problems for independent columns. Although in the sequential case, the method is not as efficient as the block pivoting variant used by other authors, they are very effective in the parallel case, producing satisfactory results for the type of applications where is to be used.

P. Alonso (✉)
Departamento de Matemáticas, Universidad de Oviedo, Oviedo, Spain
e-mail: palonso@uniovi.es

V. M. García · A. M. Vidal
Departamento de Sistemas Informáticos y Computación,
Universitat Politècnica de València, Valencia, Spain
e-mail: vmgarcia@dsic.upv.es

A. M. Vidal
e-mail: avidal@dsic.upv.es

F. J. Martínez-Zaldívar · A. Salazar · L. Vergara
Instituto de Telecomunicaciones y Aplicaciones Multimedia,
Universitat Politècnica de València, Valencia, Spain
e-mail: fjmartin@dcom.upv.es

A. Salazar
e-mail: asalazar@dcom.upv.es

L. Vergara
e-mail: lvergara@dcom.upv.es

For the Lee and Seung method, we propose a reorganization of the algorithm steps that increases the convergence speed and a parallelization of the solution. The article also includes a theoretical and experimental study of the performance obtained with similar matrices to that which arise in applications that have motivated this work.

Keywords NNMF · Parallel computing · Multicore architectures · Alternating least squares method · Lee and Seung method

1 Introduction

As far as we know, the use of Non-negative Matrix Factorization (NNMF) in multiple applications starts from a couple of papers by Lee and Seung [1,2] dealing with the perception of the parts of objects and from some papers on factor analysis by Paatero et al. [3,4]. In recent years, the NNMF has become an essential tool in many fields such as document clustering, data mining, machine learning, data analysis, image analysis, audio source separation, bioinformatics, etc. [5–11].

Let us consider the problem of approximating a matrix $\mathbf{A} \in \mathbb{R}^{m\times n}$ with non-negative elements by means of a product of two matrices with non-negative elements, $\mathbf{W} \in \mathbb{R}^{m\times k}$, $\mathbf{H} \in \mathbb{R}^{k\times n}$, of low rank $k \leq \min(m, n)$, such that $\mathbf{A} \approx \mathbf{WH}$. The problem can be addressed as the computation of two matrices $\mathbf{W}_0$, $\mathbf{H}_0$ such that

$$\|\mathbf{W}_0\mathbf{H}_0 - \mathbf{A}\|_F = \min_{\mathbf{W}\geq\mathbf{0},\mathbf{H}\geq\mathbf{0}} \|\mathbf{WH} - \mathbf{A}\|_F \,. \tag{1}$$

We have used throughout this document the Frobenius norm. It is also possible to use another kind of distance instead of it depending on the application in which this decomposition is desired (see for example [2] where the generalized Kullback–Leibler divergence is used).

The importance of this decomposition lies in the fact of building an approximation of the matrix $\mathbf{A}$, from a positive basis, $\mathbf{W}$, with a small number of columns as compared with the size of the matrix $\mathbf{A}$. This allows that each column vector of $\mathbf{A}$ can be approximated as a linear combination of vectors of a base, $\mathbf{W}$, considerably smaller than $\mathbf{A}$, thus converting the NNMF into a very useful compression tool in many applications.

Often, the size of the matrices appearing in these problems is large enough to require a significant computational effort. The use of parallel computing techniques is a good alternative to reduce the computation time of the NNMF. Many algorithms have been developed for the calculation of this factorization; notably, for example, those presented in [2] by its simplicity and its ability to be used effectively in many applications. But the convergence speed of these algorithms is slow and their cost per iteration is large. Also are worth mentioning algorithms based on the gradient method [12]. Among the newer methods the HALS method [13,14] and others that use some kind of divergence approach can also be cited to evaluate the distance between $\mathbf{A}$ and $\mathbf{WH}$. See for example [15].

In our work, we have used two methods for computing the NNMF: an alternating least squares method, similar to that used in [16–18] and the Lee and Seung algorithm shown in [1,2].

The problem (1) is not a convex problem but it can be solved by considering successive iterations where one of the two matrices to be calculated, either **W** or **H**, is fixed. Thus, only a convex problem must be solved each time, and this can be carried out using a least squares method to obtain the best approximation to matrix **W** or to matrix **H**. This is the essence of alternating least squares methods. In [16] for example, a good algorithm of this kind is presented. The algorithm is combined with a Block Principal Pivoting algorithm to simultaneously solve the optimization problems with inequality constraints in the case of several vectors. The algorithm thus obtained is very efficient.

The Lee and Seung algorithm provides very good performance in terms of simplicity and opportunities for parallelization. In this work, we have reorganized the algorithm to obtain a quality approach from the point of view of minimizing the error and to parallelize it to obtain a low cost per iteration.

Our goal is to develop parallel algorithms that can be used in solving problems of audio source separation [19]. We have focused our implementations on multicore architecture because the audio application may be executed in general purpose laptops where it is easier to find a multicore processor than a GPU suitable to be programmed (i.e. Intel graphics cards vs. Nvidia or AMD GPUs). Our parallelization for the NNMF decomposition is based on two algorithms: Lee and Seung algorithm and Alternating Least Square approach.

Previous experiences in parallelization of NNMF algorithms can be found in [20], where the authors develop an algorithm on GPU based on the Lee and Seung algorithm. Also it is noteworthy that the parallel approach in [21] for distributed memory clusters or the work described in [5] where the authors also used the Lee and Seung algorithm for solving some audio signal problems. Performance of this approach is valuable in their respective parallel machines but all of them focuses exclusively on Lee and Seung algorithms and no information about the error of algorithms is presented in comparison with other methods.

The rest of the paper is organized as follows: Sect. 2 describes the sequential algorithms to be parallelized and analyzes their theoretical computational cost. In Sect. 3, we discuss the parallelization strategy and the parallel algorithms are proposed. We also study the theoretical computational cost of the parallel algorithms. Section 4 presents an experimental analysis of the performance of the parallel algorithms from the point of view of both the numerical accuracy and the execution time improvement with regard to the sequential algorithms. Finally, we show some conclusions in Sect. 5.

2 Computing the NNMF

2.1 The Lee and Seung algorithm

The simplest approach for calculating NNMF is provided by Lee and Seung in [2]. The idea is quite simple and only involves matrix–matrix product type operations and component-wise matrix operations. In [2] the convergence of this algorithm to a solution of (1) is also shown. Algorithm 1 has the advantage of its simplicity but also the disadvantages of slow convergence towards the solution or a high residue if a small number of iterations is carried out. A modification that increases the speed of

convergence of the factorization is shown in Algorithm 2. The idea for increasing the speed of convergence of LSA and improving the error bounds for a fixed number of iterations, is to update matrix $\mathbf{H}$ with the newly computed version of the matrix $\mathbf{W}$. This variation is used in the results obtained in the present paper. In both algorithms $\odot$ and $\oslash$ denote matrix element-wise multiplication and division, respectively.

Algorithm 1 Lee and Seung Algorithm (LSA)

Inputs:
$\mathbf{A} \in \mathbb{R}^{m \times n}$, $A(i, j) \geq 0 \ \ \forall i, j, k \leq \min(n, m)$
Outputs:
$\mathbf{W} \in \mathbb{R}^{m \times k}$, $\mathbf{H} \in \mathbb{R}^{k \times n}$, $W(i, j)$, $H(i, j) \geq 0 \ \ \forall i, j$
1: Choose random $\mathbf{W} \in \mathbb{R}^{m \times k}$, $\mathbf{H} \in \mathbb{R}^{k \times n}$, $W(i, j)$, $H(i, j) \geq 0$
2: **for** $cont = 1, 2, \ldots \rightarrow$ Convergence **do**
3: $\mathbf{B} = \mathbf{W}^T\mathbf{W}\mathbf{H}$
4: $\mathbf{C} = \mathbf{W}^T\mathbf{A}$
5: $\mathbf{D} = \mathbf{W}\mathbf{H}\mathbf{H}^T$
6: $\mathbf{E} = \mathbf{A}\mathbf{H}^T$
7: $\mathbf{H} = \mathbf{H} \odot \mathbf{C} \oslash \mathbf{B}$
8: $\mathbf{W} = \mathbf{W} \odot \mathbf{E} \oslash \mathbf{D}$
9: **end for**

Algorithm 2 Modified Lee and Seung Algorithm (MLSA)

Inputs:
$\mathbf{A} \in \mathbb{R}^{m \times n}$, $A(i, j) \geq 0 \ \ \forall i, j, k \leq \min(m, n)$
Outputs:
$\mathbf{W} \in \mathbb{R}^{m \times k}$, $\mathbf{H} \in \mathbb{R}^{k \times n}$, $W(i, j)$, $H(i, j) \geq 0 \ \ \forall i, j$
1: Choose random $\mathbf{W} \in \mathbb{R}^{m \times k}$, $\mathbf{H} \in \mathbb{R}^{k \times n}$, $W(i, j)$, $H(i, j) \geq 0$
2: **for** $cont = 1, 2, \ldots \rightarrow$ Convergence **do**
3: $\mathbf{B} = \mathbf{W}^T\mathbf{W}\mathbf{H}$
4: $\mathbf{C} = \mathbf{W}^T\mathbf{A}$
5: $\mathbf{H} = \mathbf{H} \odot \mathbf{C} \oslash \mathbf{B}$
6: $\mathbf{D} = \mathbf{W}\mathbf{H}\mathbf{H}^T$
7: $\mathbf{E} = \mathbf{A}\mathbf{H}^T$
8: $\mathbf{W} = \mathbf{W} \odot \mathbf{E} \oslash \mathbf{D}$
9: **end for**

The main operations involved in Algorithm 2 are matrix multiplications (lines 3, 4, 6 and 7) and the element-wise matrix–matrix multiplications/divisions (lines 5 and 8) of the Algorithm 2. This results in a computational cost that can be expressed in flops per iteration as:

$$T_{\text{SEQ}_i} = (4mnk + 2k(m+n)(k+1)) \approx \left(4mnk + 2k^2(m+n)\right) \quad \text{flops/iter.}$$

2.2 Alternating least squares algorithm

Another good approach for calculating the NNMF is the alternating least squares method [9,16,17]. The basic idea is to transform the non-convex problem (1) into two convex optimization problems by setting the values of the matrices $\mathbf{W}$ or $\mathbf{H}$ as fixed. This idea can be expressed as:

1: Choose random $\mathbf{W}^{(0)} \in \mathbb{R}^{m\times k}$, $\mathbf{H} \in \mathbb{R}^{k\times n}$, $W(i,j)$, $H(i,j) \geq 0$
2: **for** $i = 1, 2, \ldots \rightarrow$ Convergence **do**
3: $\mathbf{H}^{(i)} = \underset{\mathbf{H}\geq\mathbf{0}}{\arg\min} \mathbf{W}^{(i-1)}\mathbf{H} - \mathbf{A}$
4: $\mathbf{W}^{(i)} = \underset{\mathbf{W}\geq\mathbf{0}}{\arg\min} \mathbf{W}\mathbf{H}^{(i)} - \mathbf{A}$
5: **end for**

In [22], the convergence of this method to a stationary point of the problem (1) is shown, provided that the solution of (3:) and (4:) is obtained using a non-negative least squares (NNLS) algorithm, that is, by solving the optimization inequality constraints problems by an adequate method, as for example the active sets method which is described in [23] or [24] or implemented in the `nnls` subroutine of Matlab, [25]. The solution of problems (3:) or (4:) may be tackled independently for each column of matrix $\mathbf{H}$ in (3:) or each row of $\mathbf{W}$ in (4:), that is to say

$$\mathbf{H}^{(i)} = \underset{\mathbf{H}\geq\mathbf{0}}{\arg\min} \left\| \mathbf{W}^{(i-1)}\mathbf{H} - \mathbf{A} \right\|,$$

which is equivalent to

$$\begin{aligned} &\text{for } j = 1, 2, \ldots, n \\ &\qquad \mathbf{H}(:, j) = \underset{\mathbf{H}(:,j)\geq\mathbf{0}}{\arg\min} \left\| \mathbf{W}^{(i-1)}\mathbf{H}(:, j) - \mathbf{A}(:, j) \right\| \\ &\text{end for} \end{aligned} \tag{2}$$

In [16] a good algorithm based on this idea is presented. There, the authors exploit the possibility of applying the optimization process to a set of columns with the same set of active constraints when applying the method of the active sets. The algorithm, called Block Pivoting Algorithm, provides good performance in terms of execution time and accuracy in the calculation of the matrices $\mathbf{W}$ and $\mathbf{H}$, although with few possibilities of parallelization. In our case, we have chosen to work with an alternating least squares algorithm based on an optimization of each column in Eqs. (2), (3) independently, using an NNLS type algorithm to perform this optimization and using the fact that

$$\min_{\mathbf{h}\geq\mathbf{0}} \|\mathbf{W}\mathbf{h} - \mathbf{f}\| = \min_{\mathbf{h}\geq\mathbf{0}} \|\mathbf{Q}\mathbf{R}\mathbf{h} - \mathbf{f}\| = \min_{\mathbf{h}\geq\mathbf{0}} \left\| \mathbf{R}\mathbf{h} - \mathbf{Q}^T\mathbf{f} \right\|, \tag{3}$$

for any vector $f \in \mathbb{R}^{m\times 1}$.

Thus, the minimization routine NNLS acts on an upper triangular matrix. The computation of the QR decomposition of $\mathbf{W}$, and the consequent matrix multiplication $\mathbf{Q}^T\mathbf{A}$ can be performed before at the beginning of the optimization column loop. The algorithm can be expressed as follows using pseudocode language:

The main operations involved in this algorithm are two QR decompositions (lines 3 and 8), two matrix multiplication (lines 4 and 9) and the computation of the solution of two non-negative vector least squares problem (lines 6 and 11), just inside loops 5–7 and 10–12. These operations have not been implemented explicitly, but optimum

Algorithm 3 Alternating Least Squares Algorithm (ALSA)

Inputs:
$\mathbf{A} \in \mathbb{R}^{m \times n}$, $A(i, j) \geq 0 \ \ \forall i, j, k \leq \min(n, m)$
Outputs:
$\mathbf{W} \in \mathbb{R}^{m \times k}$, $\mathbf{H} \in \mathbb{R}^{k \times n}$, $W(i, j)$, $H(i, j) \geq 0 \ \ \forall i, j$
1: Choose random $\mathbf{W} \in \mathbb{R}^{m \times k}$, $W(i, j) \geq 0$
2: **for** $cont = 1, 2, \ldots \rightarrow$ convergence **do**
3: $[\mathbf{Q}, \mathbf{R}] = qr(\mathbf{W})$;
4: $\mathbf{B} = \mathbf{Q}^T \mathbf{A}$;
5: **for** $j = 1, 2, ..., n$ **do**
6: $\mathbf{H}(:, j) = \arg\min_{\mathbf{h} \geq \mathbf{0}} \|\mathbf{R}\mathbf{h} - \mathbf{B}(:, j)\|$
7: **end for**
8: $[\mathbf{Q}, \mathbf{R}] = qr(\mathbf{H}^T)$;
9: $\mathbf{B} = \mathbf{Q}^T \mathbf{A}^T$;
10: **for** $j = 1, 2, ..., m$ **do**
11: $\mathbf{W}(j, :) = \arg\min_{\mathbf{w} \geq \mathbf{0}} \|\mathbf{R}\mathbf{w} - \mathbf{B}(:, j)\|$
12: **end for**
13: **end for**

LAPACK subroutines have been used instead. This results in a computational cost per iteration which can be expressed in flops as:

$$T_{\text{SEQ}_i} = 2k^2\left(m - \frac{k}{3}\right) + 2m^2 n + nT_{\text{NNLS}} + 2k^2\left(n - \frac{k}{3}\right) + 2n^2 m + mT_{\text{NNLS}} \quad \text{flops/iteration.}$$

Note that T_{NNLS} is an estimate for the cost in flops of the solution of a non-negative vector least squares problem. This cost may vary depending on the initial active constraints encountered for each problem. We have approximated this cost as a fixed value, depending only on the size of the $k \times k$ upper triangular part of matrix $\mathbf{R}$, assuming that the NNLS algorithm is applied n times for computing the columns of matrix ($\mathbf{h}$, lines 5–7 of the Algorithm 3), and m times for computing the columns of matrix ($\mathbf{w}$, lines 10–12 of the Algorithm 3). The assumption has no special consequences in the performance analysis of neither the sequential algorithm nor the parallel algorithm. After some arithmetics, if n_{iter} denotes the number of iterations to reach the convergence, T_{SEQ} can be approximated as

$$T_{\text{SEQ}} \approx \left[\left(2k^2 + mn\right)(m + n) + (m + n)\, T_{\text{NNLS}}\right] \times n_{\text{iter}} \quad \text{flops.} \tag{4}$$

3 Parallel approach

The objective of this work is to obtain efficient algorithms for calculating the NNMF on a multicore architecture. Assume, therefore, that a parallel computer with shared memory and the ability to release different threads simultaneously on various processor cores are available. Computers that use processors such as recent versions of Intel

Core i7 or AMD Opteron are examples of suitable parallel machines to execute our algorithm.

OpenMP programming environment allows efficient parallel programming on such computers. There are also specialized high performance libraries for these machines, as BLAS or LAPACK, allowing the use of threads that run simultaneously on the different cores of the processor and therefore enable us to efficiently parallelize numerical linear algebra operations (QR decomposition, matrix–matrix multiplication, etc.).

3.1 Parallelization of the Lee and Seung algorithm

One possibility to accelerate LSA and improve the error bounds for a fixed number of iterations is to update matrix **H** with the newly computed version of matrix **W** (algorithm MLSA). Besides, since the highest cost operations in MLSA are basically matrix–matrix products, the simplest approach to parallelize it is to use threaded numerical linear algebra libraries. This option allows, first of all, validation of algorithms and realization of a performance analysis in terms of accuracy, number of iterations required for convergence, comparisons with other standard algorithms, etc. On the other hand, it provides a set of reference implementations, which allows to analyze other specific implementations, explicitly organized by the programmer, with regard to runtime, speedup or other parameters. In the case of the parallel approach for dense matrices on multicore processors, a threaded version of BLAS3/LAPACK has been used to design the parallel implementation. This has allowed the efficient use of CPU resources with multiple cores, as well as automatic organization of blocks of parallel algorithms.

3.2 Parallelization of the ALS algorithm

The main operations which can be performed in parallel in the ALSA version are those loops associated with each column optimizations (loops in lines 5–7 and 10–12 in Algorithm 3). For that, we just run simultaneously each step of the loop in the different cores available in the processor. Besides, matrix operations as calculation of the QR decomposition (`DGEQRF`) and application of an orthogonal matrix (without its formation: `DORMQR` routine, steps 4 and 9 in the algorithm) can be run in parallel using threaded versions of LAPACK. Now, the matrix operations are performed by means of calls to a threaded library, and their computational cost can be considered as a fraction of the sequential cost (the average cost of executing a flop is smaller than in the sequential case). The concrete value of this fraction depends on the matrix size, block size, number of threads, etc. We express this fraction with a constant value α such that $0 \leq \alpha \leq 1$. Besides, we assume that loops in lines 5–7 and 10–12 in Algorithm 3 are perfectly parallelized. Thus, if p denotes the number of threads utilized, the computational cost of the parallel algorithm can be expressed in

flops as:

$$T_{\mathrm{PAR}} \approx \left[\left(2k^2 + mn \right) (m+n)\, \alpha + \frac{(m+n)}{p} T_{\mathrm{NNLS}} \right] \times n_{\mathrm{iter}} \quad \text{flops.} \tag{5}$$

4 Performance of the algorithms

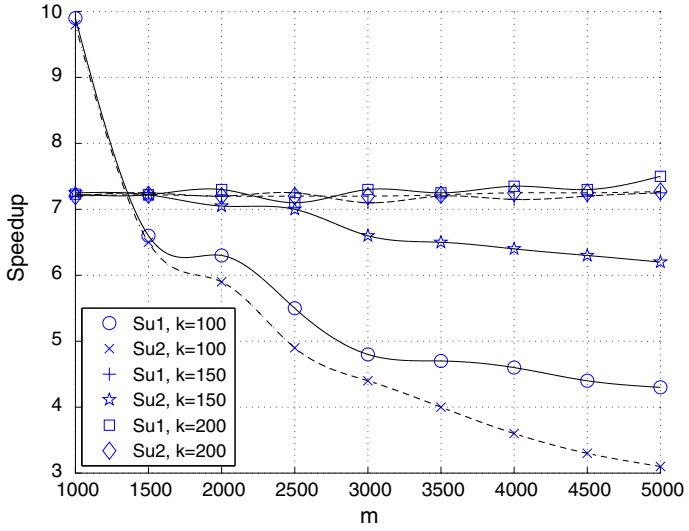
We analyze in this section the performance of the proposed algorithms from two points of view: speedup and accuracy. The machine where the tests ran has two Intel(R) Xeon(R) CPU X5675 processors working at a frequency of 3 GHz; each processor has six cores with the hyperthreading capability disabled, so that the machine has twelve cores in total. The total amount of its main memory is 128 GB. The C/C++ compiler is the Intel C/C++ Composer XE 13.0 2013 running on a 64 bits Ubuntu 10.04 Linux operating system. Every time result has been obtained with five tests, discarding the best and the worst values and averaging the remaining intermediate ones. The double precision matrix operations have been computed using the Intel MKL numerical linear algebra library release 11.0 threaded version using either one thread or six or twelve (the maximum number of cores). With respect to the parallelization of Algorithm 3, the loops in lines 5–7 and 10–12 *(the optimization loops)* are executed either sequentially (one thread) or in parallel (twelve/six threads, using the OpenMP API of the compiler).

The execution time of the algorithm is due to two concepts: on the one hand, the QR factorizations and the orthogonal matrix applications (without explicit formation of orthogonal matrix $\mathbf{Q}$), and on the other hand, the optimization loops. Let us define two different *sequential* execution times: the first one, T_{seq_1}, where the library subroutines that compute the QR factorizations and matrix applications use one thread; and the second one, T_{seq_2}, where the library subroutines use twelve threads. We will consider the *parallel* execution time T_{par} using twelve threads in the numerical linear algebra library and in the optimization loop parallelization. Hence, we obtain two different speedups $\mathtt{Su1} = T_{\mathrm{seq}_1}/T_{\mathrm{par}}$ and $\mathtt{Su2} = T_{\mathrm{seq}_2}/T_{\mathrm{par}}$.

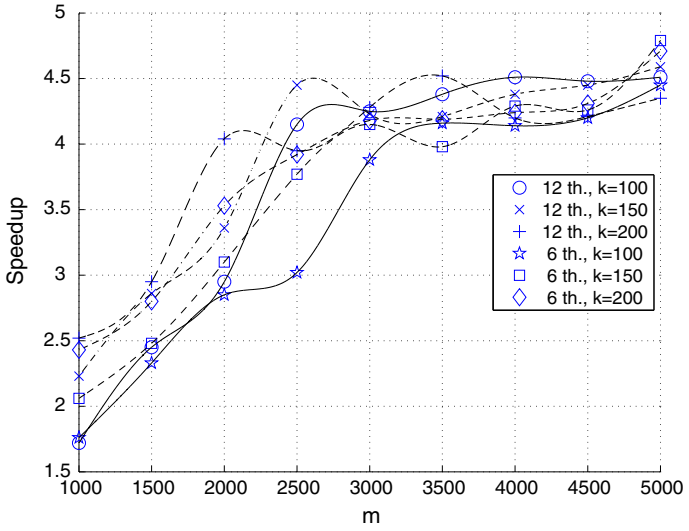
Figure 1a shows the speedup of the parallelization considering sequential (`Su1`) and parallel (`Su2`) executions, respectively, of the numerical linear algebra subroutines. Obviously, `Su1` is always higher than `Su2` because the sequential execution time T_{seq_1} (1 thread used) is higher than T_{seq_2} (12 threads used) and the parallel execution time T_{par} is the same in both speedups computations.

In general, both speedups remain relatively constant for higher values of k because with such values, the most time consuming part of the algorithm is due to the optimization loops. The difference between both speedups is higher when m increases for a fixed k, and lower when k increases for a fixed m. For a fixed k, when m increases, the parallelization of the numerical linear algebra library is more efficient, so T_{seq_2} gets lower, so `Su2` is lower and hence the difference with `Su1` is higher. For a fixed m, the effect of the parallelization of the numerical linear algebra library is irrelevant because the optimization loops execution time is the most time consuming part of the algorithm.

Figure 1b shows the speedup of the Lee and Seung algorithm parallelization for different values of k using either 12 or 6 threads. We can observe how the parallel efficiency using 12 threads is lower than using 6 threads due to the cache coherence



(a) Alternating Least Squares

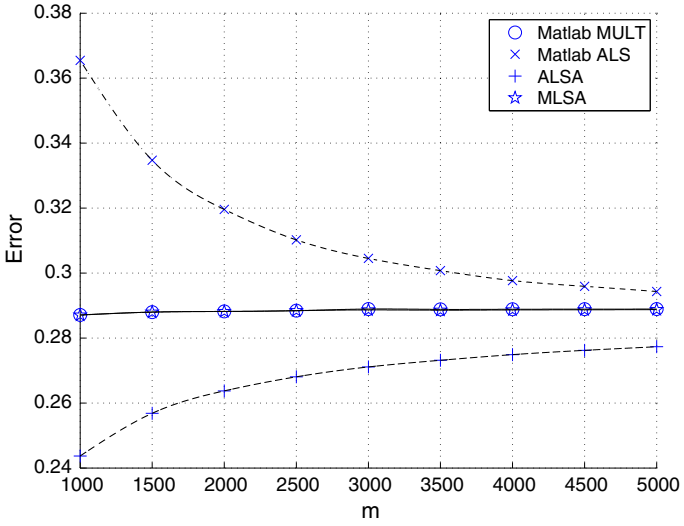


(b) Lee and Seung

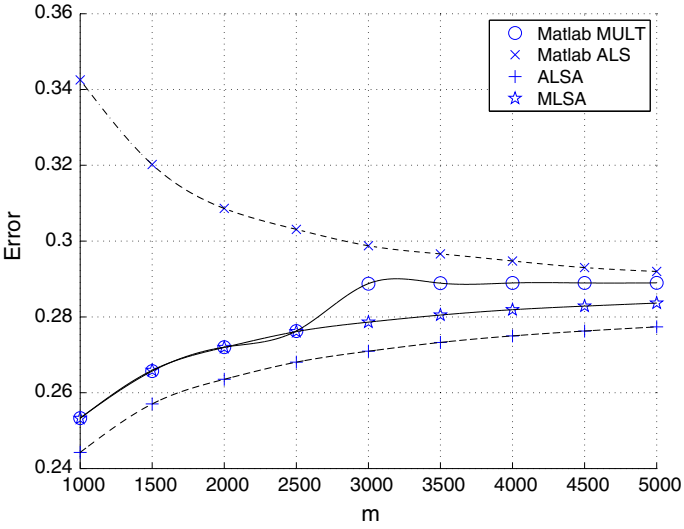
Fig. 1 NNMF speedup ($n = m/2$)

overhead of the algorithm (this problem does not appear in the ALSA version). It can be observed how the speedup increases when m increases and reaches a saturation value around 4.5.

Figures 2 and 3 show the factorization error comparing four methods: the Matlab MULT (Multiplicative update algorithm used in Matlab for different kind of test matrices: `nnmf(A,k,'algorithm','mult')`), the Matlab ALS (Alternating



(a) 5 iterations

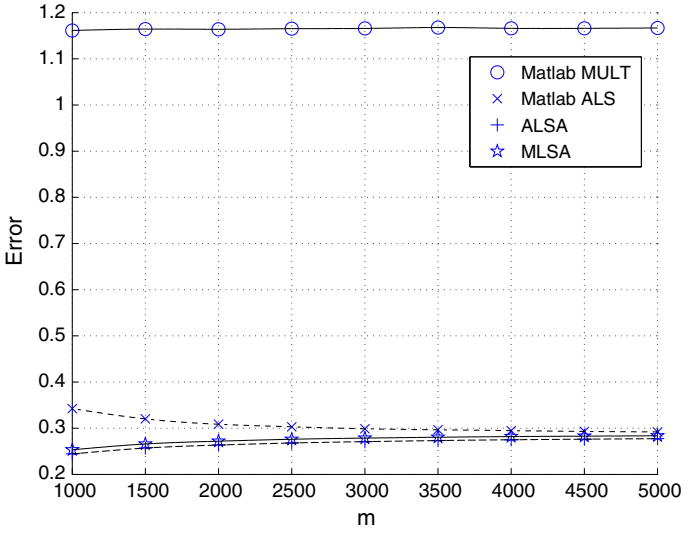


(b) 100 iterations (5 iteration in ALSA)

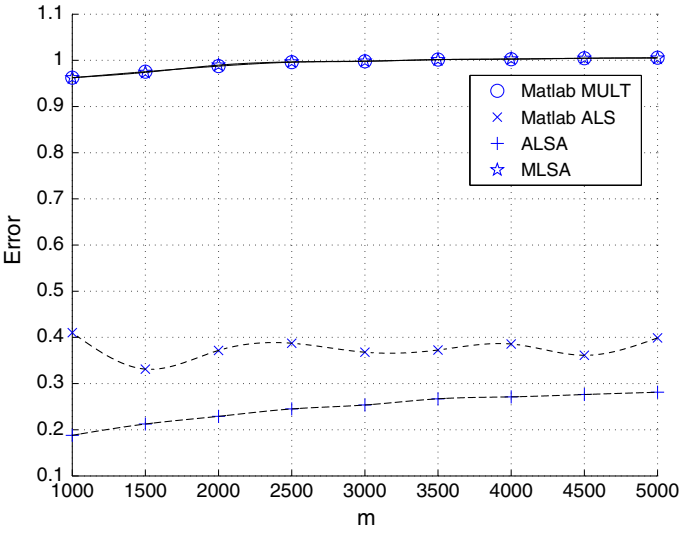
Fig. 2 NNMF error (non biased random matrix, $n = m/2$, $k = 100$)

Least Squares algorithm used in Matlab: `nnmf(A,k,'algorithm', 'als')`), and our proposed methods for ALSA and MLSA algorithms, and for full rank ($A =$ `rand`(m, n)) and biased full rank ($A =$ `rand`$(m, k) *$ `rand`$(k, n) + \epsilon *$ `rand`(m, n), with ϵ small) random matrix cases.

The error of the factorization is computed as $\|\mathbf{WH} - \mathbf{A}\|_F / \sqrt{m \times n}$. The values of m are between 1,000 and 5,000, with $n = m/2$ and $k = 100$. The residue calculated in our case is the one obtained after applying either 5 or 100 iterations, except for



(a) 5 iterations



(b) 100 iterations (5 iteration in ALSA)

Fig. 3 NNMF error (biased random matrix, $n = m/2$, $k = 100$)

ALSA algorithm that always uses 5 iterations, since the advantage of this algorithm is that with only five iterations achieves a better accuracy than the others. From the standpoint of the accuracy, the implemented algorithms obtain a residue better than Matlab implementations except for the MLSA method with a biased matrix after 100 iterations where the residue obtained with Matlab ALS is lower although our ALSA method outperforms it. This can be observed in Figs. 2, 3.

5 Conclusions

We have implemented parallel algorithms that compute the NNMF on a multicore computer. The algorithms have good performance in terms of speedup. The accuracy of the algorithm is even superior to some standard algorithms implemented in existing libraries. The algorithms have been organized so that they can take advantage of two types of parallelism independently, by separating numerical linear algebra operations that can be parallelized through threaded versions of standard libraries as BLAS/LAPACK, and some loops that can be immediately parallelized using OpenMP directives.

Acknowledgments This work has been supported by European Union ERDF and Spanish Government through TEC2012-38142-C04 project and Generalitat Valenciana through PROMETEO/2009/013 project.

References

1. Lee DD, Seung HS (1999) Learning the parts of objects by non-negative matrix factorization. Nature 401:788–791
2. Lee DD, Seung HS (2001) Algorithms for non-negative matrix factorization. Advances in neural information processing systems, vol 13. MIT Press, Cambridge, pp 556–562
3. Paatero P, Tapper U (1994) Positive matrix factorization: a non-negative factor model with optimal utilization of error estimates of data values. Environmetrics 5:111–126
4. Paatero P (1997) Least squares formulation of robust non-negative factor analysis. Chemometr Intell. Lab. Syst. 37:23–35
5. Battenberg E, Freed A, Wessel D (2010) Advances in the parallelization of music and audio applications. International computer music conference. Stony Brook, New York
6. Wnag J, Zhong W, Zhang J (2006) NNMF-based factorization techniques for high-accuracy privacy protection on non-negative-valued datasets, data mining workshops, 2006. ICDM Workshops 2006. Sixth IEEE International Conference on Computing and Processing, pp 513–517
7. Rodriguez-Serrano FJ, Carabias-Orti JJ, Vera-Candeas P, Virtanen T, Ruiz-Reyes N (2012) Multiple instrument mixtures source separation evaluation using instrument-dependent NMF models. In: Theis F (ed) LVA/ICA 2012. Springer-Verlag, Berlin, pp 380–387
8. Xu W, Liu X, Gong Y (2003) Document clustering based on non-negative matrix factorization, Proceedings of the 26th annual international ACM SIGIR conference on Research and development in information retrieval. SIGIR03, Toronto, pp 267–273.
9. Berry MW, Browne M, Langville A, Pauca V, Plemmons R (2007) Algorithms and applications for aproximate nonnegative matrix factorization. Comput Stat Data Anal 52:155–173
10. Kim J, Park H (2007) Sparse non-negative matrix factorizations via alternating non-negativity-constrained least squares for microarray data analysis. Bioinformatics 23:1495–1502
11. Devajaran K (2008) Nonnegative matrix factorization: an analytical and interpretative tool in computational biology. PLoS Comput Biol 4(7):e1000029
12. Guan N, Tao D, Luo Z, Yuan B (2012) NeNMF: an optimal gradient method for nonnegative matrix factorization. IEEE Trans Signal Proc 60(6):2882–2898
13. Cichocki A, Phan A-H (2009) Fast local algorithms for large scale nonnegative matrix and tensor factorizations. IEICE Trans Fund Electron Commun Comput Sci E92– A:708–721
14. Cichocki A, Zdunek R, Amari S-I (2007) Hierarchical ALS algorithms for nonnegative matrix and 3d tensor factorization, independent component analysis and signal separation, lecture notes in computer science, vol 4666. Springer, NewYork, pp 169–176
15. Cichocki A, Cruces S, Amari S-I (2011) Generalized alpha–beta divergences and their application to robust nonnegative matrix factorization. Entropy 13:134–170
16. Kim J, Park H (2011) Fast nonnegative matrix factorization: an active-set-like method and comparisons. SIAM J Sci Comput 33(6):3261–3281

17. Kim D, Sra S, Dhillon IS (2007) Fast newton-type methods for the least squares nonnegative matrix approximation problem, Proceedings of the seventh SIAM international conference on data mining. SIAM, Philadelphia, pp 343–354
18. Kim J, Park H (2008) Nonnegative matrix factorization based on alternating nonnegativity constrained least squares and active set method. SIAM J Matrix Anal Appl 30:713–730
19. Carabias-Orti JJ, Virtanen T, Vera-Candeas P, Ruiz-Reyes N, Caadas-Quesada FJ (2011) Musical instrument sound multi-excitation model for non-negative spectrogram factorization. IEEE J Select Top Signal Proc 5(6):1144–1158
20. Mejia-Roa E, Garcia C, Gomez JI, Prieto M, Tenllado C, Pascual-Montano A, Tirado F (2012) Parallelism on the non-negative matrix factorization, applications, tools and techniques on the road to exascale computing, vol 22. IOS Press, Netherlands
21. Dong C, Zhoa H, Wang W (2010) Parallel nonnegative matrix factorization algorithm on the distributed memory platform. Int J Par Prog 38:117–137
22. Grippo L, Sciandrone M (2000) On the convergence of the block nonlinear Gauss–Seidel method under convex constraints. Oper Res Lett 26:127–136
23. Lawson CL, Hanson RJ (1995) Solving least squares problems. SIAM, Philadelphia
24. Bjorck A (1996) Numerical methods for least squares problems. SIAM, Philadelphia
25. MATLAB version 8.0 (R2012b) (2012) The MathWorks Inc, Natick, Massachusetts.