Considerations on Parallelizing Nonnegative Matrix Factorization for Hyperspectral Data Unmixing

Stefan A. Robila, *Member, IEEE*, and Lukasz G. Maciak

***Abstract*—Nonnegative matrix factorization (NMF) is a recently developed linear unmixing technique that assumes that the original sources and transform were positively defined. Given that the linear mixing model (LMM) for hyperspectral data requires positive endmembers and abundances, with only minor modifications, NMF can be used to solve LMM. Traditionally, NMF solutions include an iterative process resulting in considerable execution times. In this letter, we provide two novel algorithms aimed at speeding the NMF through parallel processing: the first based on the traditional multiplicative solution and the second modifying an adaptive projected gradient technique known to provide better convergence. The algorithms' implementations were tested on various data sets; the results suggest that a significant speedup can be achieved without decrease in accuracy. This supports the further use of NMF for linear unmixing.**

***Index Terms*—Hyperspectral data, linear algorithms, linear unmixing, nonnegative matrix factorization (NMF), parallel processing, remote sensing.**

I. Introduction

In HYPERSPECTRAL images, the area covered by one pixel may correspond to a mixture of materials available in the scene. Since in this case the number of mixing materials, their contributions, and the nature of mixing may not be known, a direct match of the pixel spectra with ones from known material databases is not possible. Instead, one must first unmix the pixels, i.e., identify the materials contributing to the pixels and their abundances [1].

Among the various approaches to model the mixing process, a very simple yet extensively used one is given by the linear mixing model (LMM). In LMM, each observed spectra $\mathbf{x}$ can be expressed as [2]

$$\mathbf{x} = \sum_{i=1}^{m} s_i \mathbf{w_i} + \mathbf{a} = \mathbf{W}\mathbf{s} + \mathbf{a} \tag{1}$$

where $\mathbf{W}$ is an $n \times m$ matrix of endmember spectra $(\mathbf{w_1}, \ldots, \mathbf{w_m})$ of the individual composing materials, $\mathbf{s}$ is an m-dimensional vector describing the fractional abundances of the endmembers in the mixture, and $\mathbf{a}$ is the additive noise vector. The abundance vector components are positive and with unit sum

$$s_i \geq 0, \qquad i = 1, \ldots, m \qquad \sum_{i=1}^{m} s_i = 1. \tag{2}$$

Identifying various materials reduces to solving the linear mixing problem, i.e., finding the endmembers and their abundances and then matching them with library spectra. In many situations, the endmembers can be *a priori* identified directly from the scene or through various preprocessing techniques that identify potential pure pixels. In recent years, several algorithms have been developed for autonomous and supervised endmember extraction [3]. They include the pixel purity index that is based on the geometry of convex sets [4], N-FINDR that finds the set of pixels which define the simplex with the maximum volume, potentially inscribed within the data set [5], automated morphological endmember extraction based on mathematical morphology and using both spectral and spatial information [6], and the ORASIS algorithm that eliminates redundant spectra by calculating the angle between spectral vectors [7]. In these algorithms, unmixing is followed by the computation of the abundances of each endmember for each pixel in the scene using a maximum likelihood estimation approach [2].

A different approach is taken when the goal is to produce the endmembers and the abundances simultaneously. Solutions to simultaneous unmixing include techniques adapted from more general source separation theory such as principal component analysis [8] and independent component analysis [9]. More recently, nonnegative matrix factorization (NMF) was offered as an attractive alternative due to its lax constraints on the sources and abundances [10]. At the same time, NMF solutions incur high computational costs. They are based on iterative algorithms that have the complexity dependent on the size of the data. Given a predicted increase in data size (due to improvements in both spatial and spectral resolutions), in order for such techniques to remain feasible, one must investigate approaches to speed them up without compromising accuracy.

Supported by technological advances, high-performance computing has continuously increased its usability for remote-sensing applications with the focus being on commodity systems such as Beowulf clusters [11]. When the goal is processing large data sets in a relatively independent manner, distributed computing based on clusters proves to be the solution to otherwise computational infeasible tasks. However, the speedup obtained by the use of clusters is limited by the communication costs associated to sharing the data over the cluster network and the collection and synthesis of results.

Manuscript received October 17, 2007; revised April 10, 2008, May 12, 2008, and July 23, 2008. First published November 17, 2008; current version published January 14, 2009. This work was supported in part by the Sun Microsystems Academic Excellence under Grant T-US-697950-C.

The authors are with the Department of Computer Science, Montclair State University, Montclair, NJ 07043 USA (e-mail: robilas@mail.montclair.edu; maciakl@mail.montclair.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/LGRS.2008.2005793

Feature extraction and simultaneous unmixing techniques require that large amounts of data be shared among the network's processing units and that frequent synchronization stages be employed in order to maintain data consistency from one algorithm iteration to another. These limitations, in turn, increase the communication cost overhead in a cluster. Compared with it, tightly coupled multiprocessor systems do not suffer from the same issues. Such systems contain two or more central processor units that communicate directly over the system bus and share most of computer's components, including the main memory [12]. While initially parallel architectures based on shared communication bus and memory were significantly more expensive than commodity clusters, in recent years, multiprocessor and multicore architectures have become mainstream technologies in most of the off-the-shelf systems. Such a trend is expected only to increase in the future with predictions of many-core (leading to 1000 cores) systems now being made [12] and prototypes for 80-core chips currently being tested [13].

In support of a multiprocessor multicore architecture, it is valuable to investigate how unmixing techniques can be adapted to them. In the following, we present the parallelization of NMF algorithms that result in significant speedup while producing identical results as the traditional sequential ones. This letter is structured as follows. A short introduction to NMF (Section II) and its use to spectral unmixing is followed by the description of the parallel NMF algorithms (Section III). Results (Section IV) and conclusions complete this letter.

II. NMF

Given the multidimensional observed data set $\mathbf{x}$, the NMF assumes that it was produced by applying a nonnegative linear transform $\mathbf{W}$ to the nonnegative source data $\mathbf{s}$ [10]

$$\mathbf{x} = \mathbf{W}\mathbf{s}. \tag{3}$$

NMF algorithms aim to find the source and the transform. This approach can be understood as factorizing a matrix subject to positive constraints. A direct approach to this problem is based on gradient optimization [10]

$$\mathbf{W} = \mathbf{W} - \frac{\partial f(\mathbf{W},\mathbf{s})}{\partial \mathbf{W}} \qquad \mathbf{s} = \mathbf{s} - \frac{\partial f(\mathbf{W},\mathbf{s})}{\partial \mathbf{s}} \tag{4}$$

where

$$f(\mathbf{W},\mathbf{s}) = \|\mathbf{x} - \mathbf{W}\mathbf{s}\|_F^2 \tag{5}$$

and $\|.\|_F$ designates the Frobenius (or Euclidean) norm. At each step, we also ensure that $\mathbf{W}$ and $\mathbf{s}$ are positive. To explicitly include the nonnegativity constraints, Lee and Seung [10] introduce modified update steps, shown to be equivalent to the original approach

$$\mathbf{s}_{ij} = \mathbf{s}_{ij}\frac{(\mathbf{W}^\mathbf{T}\mathbf{x})_{ij}}{(\mathbf{W}^\mathbf{T}\mathbf{W}\mathbf{s})_{ij}+\varepsilon} \qquad \mathbf{W}_{ij} = \mathbf{W}_{ij}\frac{(\mathbf{x}\mathbf{s}^\mathbf{T})_{ij}}{(\mathbf{W}\mathbf{s}\mathbf{s}^\mathbf{T})_{ij}+\varepsilon} \tag{6}$$

where ε is a relatively small value (i.e., 10^{-8}).

An attractive alternative NMF algorithm, titled adaptive projected NMF(APNMF), is described in [14]. Here, the original steps (4) are converted into

$$\mathbf{W} = P\left(\mathbf{W} - \alpha_W \frac{\partial f(\mathbf{W},\mathbf{s})}{\partial \mathbf{W}}\right)$$
$$\mathbf{s} = P\left(\mathbf{s} - \alpha_s \frac{\partial f(\mathbf{W},\mathbf{s})}{\partial \mathbf{s}}\right) \tag{7}$$

where

$$P(x) = \begin{cases} x, & x \geq 0 \\ 0, & x < 0 \end{cases} \tag{8}$$

and α_s and α_W are scalar factors that are chosen to produce the "best" intermediate $\mathbf{s}$ and $\mathbf{W}$. In case of α_s, this is done by computing the derivative of the objective function f on $\mathbf{s}$ and then taking α_s as

$$\alpha_s = \max\left\{\alpha | f(\mathbf{W},\bar{\mathbf{s}}) - f(\mathbf{W},\mathbf{s}) \leq \sigma \frac{\partial f(\mathbf{W},\mathbf{s})}{\partial \mathbf{s}}(\bar{\mathbf{s}} - \mathbf{s})\right\} \tag{9}$$

where

$$\bar{\mathbf{s}} = P\left(\mathbf{s} - \alpha \frac{\partial f(\mathbf{W},\mathbf{s})}{\partial \mathbf{s}}\right) \tag{10}$$

and σ is typically 0.01 (based on the suggestions from [14]).

The condition included in (9) is not well formed from the point of view of vector arithmetic. An alternative version is provided in [15]

$$(1-\sigma)\left\langle \frac{\partial f(\mathbf{W},\mathbf{s})}{\partial \mathbf{s}}, \bar{\mathbf{s}} - \mathbf{s}\right\rangle + \frac{1}{2}\left\langle \bar{\mathbf{s}} - \mathbf{s}, \mathbf{W}^\mathbf{T}\mathbf{W}(\bar{\mathbf{s}} - \mathbf{s})\right\rangle \leq 0 \tag{11}$$

$$\frac{\partial f(\mathbf{W},\mathbf{s})}{\partial \mathbf{s}} = \mathbf{W}^\mathbf{T}\mathbf{W}\mathbf{s} - \mathbf{W}^\mathbf{T}\mathbf{x}. \tag{12}$$

Following a similar line of reasoning, one can deduct the equations needed to obtain α_W.

The NMF algorithm was employed in [16] to unmix spectra, given only a limited number of mixed spectra. It was further modified to include the abundance constraints for summation to one in [17].

A quick analysis of the update steps in (6) reveals that while they ensure that the matrices remain positively defined, they also require significant computation times. If we assume that $\mathbf{W}$ is $n \times m$, $\mathbf{x}$ is $n \times p$, and $\mathbf{s}$ is $m \times p$, computing a new $\mathbf{s}$ and a new $\mathbf{W}$ requires $\Theta(mnp)$. Overall, NMF has the complexity $\Theta(kmnp)$, where k is the number of iterations.

The APNMF algorithm has a higher computational complexity. At each iteration, to compute the formula in (9), we would need to use (12) that is done in $\Theta(mnp)$. Then, to compute the new $\mathbf{s}$, we need $\Theta(mp)$ followed by the check for (11). This is done in $\Theta(mnp)$. Overall, to compute the new updated $\mathbf{s}$, the algorithm needs $\Theta(k_s mnp)$, where k_s is the number of alphas that need to be checked for the current iteration. Finally, taking in consideration the computation of $\mathbf{W}$ in the same manner, when ANMF runs for k iterations, we get a complexity of $\Theta(knmp(\underline{k_s} + \underline{k_W}))$, where $\underline{k_s}$ and $\underline{k_W}$ are the average numbers of internal iterations to find $\mathbf{s}$ and $\mathbf{W}$, respectively.

1. Start with random $\mathbf{W}$ and $\mathbf{s}$, and t separate threads
2. Compute in parallel $\mathbf{Ws_i}$
3. Synchronize
4. Compute in parallel $\mathbf{W^TWs_i}$, $\mathbf{W^Tx_i}$,and $\mathbf{s_i}$ **and** $\mathbf{Ws_i}$
5. Synchronize
6. Compute in parallel $\mathbf{xs^T_i}$, $\mathbf{WssT_i}$, $\mathbf{W_i}$, normalize $\mathbf{W_i}$
7. Synchronize
8. If $(f(\mathbf{W},\mathbf{s})_{prev} - f(\mathbf{W},\mathbf{s})_{current}) / f(\mathbf{W},\mathbf{s})_{prev} < \delta$ stop
   else go to step 2.

Fig. 1. PNMF algorithm.

1. Start with random $\mathbf{W}$ and $\mathbf{s}$, and t separate threads and initial values for α_s, α_W, β, σ
2. Compute in parallel $\delta f(\mathbf{W},\mathbf{s})/\delta \mathbf{s_i}$
3. Synchronize
4. Compute in parallel eq 11 for current α_s
5. Synchronize
6. If α_s is not satisfying (9),
7. update α_s go back to step 2
8. else $\mathbf{s_i}$=P($\mathbf{s_i}$-α_s $\delta f(\mathbf{W},\mathbf{s})/\delta \mathbf{s_i}$)
9. Synchronize
10. Compute in parallel $\delta f(\mathbf{W},\mathbf{s})/\delta \mathbf{W_i}$
11. Synchronize
12. Compute in parallel W equiv. of eq 11 for current α_W
13. Synchronize
14. If α_W is not satisfying W equiv. of (9)
15. update α_W go back to step 9
16. else $\mathbf{W_i}$=P($\mathbf{W_i}$-α_W $\delta f(\mathbf{W},\mathbf{s})/\delta \mathbf{W_i}$)
17. Synchronize
18. If $(f(\mathbf{W},\mathbf{s})_{prev} - f(\mathbf{W},\mathbf{s})_{current}) / f(\mathbf{W},\mathbf{s})_{prev} < \delta$ stop
    else go to step 2.

Fig. 2. PAPNMF algorithm.

III. Parallel NMF

Parallel computing is identified as the simultaneous execution of multiple instances of the program modules on multiple processors in order to obtain results faster [19].

To speedup the NMF processing, we suggest a parallelization of the NMF and APNMF algorithms by introducing their parallel versions (PNMF and PAPNMF, respectively). Figs. 1 and 2 show the parallel algorithms based on a generic number of t threads. Each parallel computation thread is synchronized in order to compute the succeeding sequential steps. The shaded areas identify the parallel components of our algorithms, while the nonshaded areas correspond to sequential or synchronization steps.

For PNMF, we compute in parallel first $\mathbf{s}$ and then $\mathbf{W}$. We note that, in Fig. 1, the notation $\mathbf{A_i}$ refers to a set of columns from the matrix $\mathbf{A}$ obtained by splitting the matrix in t equally sized submatrices

$$\mathbf{A} = [\mathbf{A_1} \quad \mathbf{A_2} \quad \cdots \quad \mathbf{A_t}]. \tag{13}$$

Once $\mathbf{Ws}$ is computed in parallel, the remaining computations ($\mathbf{W^TWs}$, $\mathbf{W^Tx}$, $\mathbf{xs^T}$, and $\mathbf{Wss_T}$) can be done independently of each other. While an updated $\mathbf{Ws}$ is needed for the computation of $\mathbf{W}$, this is done in parallel at the same time with $\mathbf{s_i}$.

The parallelization leads to a significant reduction in the computational complexity to $\Theta(kmnp/t)$ since, at every step, the computation of the data structure is done independently by each parallel thread. Some overhead will be recorded due to the synchronization and convergence steps. Such parallel approach applies very well to shared memory architectures. In the case of a distributed system, the synchronization steps will need to be replaced by communication of the updated data.

Fig. 3. (a) Hydice data. (b) SOC700 data. (c) Hyperion data.

In the case of PAPNMF, the algorithm follows a similar path replacing the multiplicative update steps by the gradient-based ones (Fig. 2). As in the sequential APNMF, the update of $\mathbf{W}$ is based on the update of $\mathbf{s}$. PANMF has the advantage of providing a faster iterative step, closer to $\Theta(knmp(\underline{k}_s + \underline{k}_W)/t)$.

The stop condition for both algorithms is based on the relative stability from one round to another. When the function $f(\mathbf{W},\mathbf{s})$ is not decreasing significantly (i.e., more than δ) from its previous value, the algorithm stops. This is similar to the approach taken in [10] and [14]. Alternative stop conditions are based on the maximum number of rounds. At the end, however, it is still an open question how a higher accuracy value would affect the data processing.

For the implementation and testing of the four algorithms (NMF, APNMF, PNMF, and PAPNMF), we used Java 1.6 over a symmetric multiprocessor architecture [18] in order to leverage Java's powerful threading implementation.

Threading is a native feature of the language ensuring robustness and scalability and is also supported by a one-to-one mapping of the Java threads into kernel threads on most current operating systems [20]. The code was run on a DellPrecision 8 Intel Xeon 2.66-GHz CPU, 4-GB RAM system running Solaris 8. The hyperspectral data sets were transformed into a 2-D array with each image band corresponding to a row vector and each spectra corresponding to a column. We refer the reader to previous work in [16] and [17] for an explanation on the relevance of NMF for unmixing. Here, we focus the experiments on the comparative analysis between sequential and parallel versions of NMF.

IV. Experimental Results

The first image set [Fig. 3(a)] corresponds to a 100 × 190 pixel 169-band subset from a Hydice foliage scene with a spatial resolution of 1.5 m at wavelengths from the 0.4 to 2.5 μm part of the Forest Radiance set. Various panels are present in the scene organized on eight rows and of different sizes.

The second data set [Fig. 3(b)] was produced using an SOC 700 hyperspectral sensor in our laboratory. The camera generates 320 × 320 pixel images on 120 bands equally spaced within 400 and 900 nm. The setup is formed of an artificial plant arranged in a light brown ceramic pot. Several real leaves (shown enhanced in the picture) were placed between the artificial leaves. The arrangement was placed on a large rock

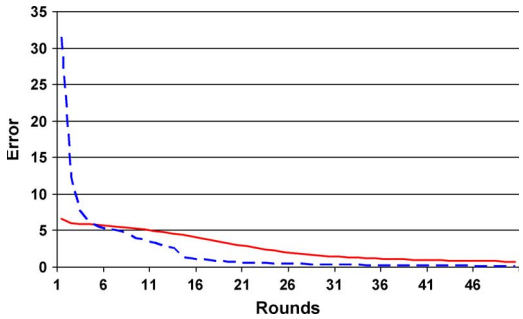

Fig. 4. Error graph for (solid) MNF and (dashed) APNMF—Hydice data.

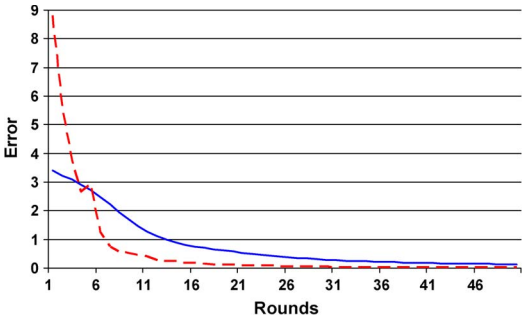

Fig. 5. Error graph for (solid) MNF and (dashed) APNMF—SOC700 data.

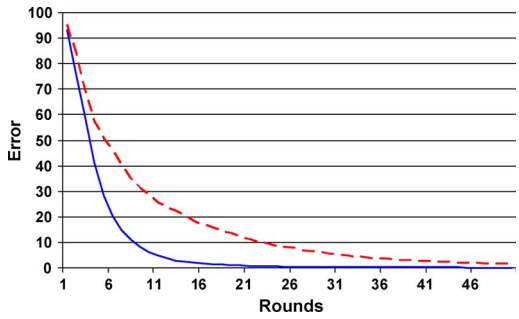

Fig. 6. Error graph for (solid) MNF and (dashed) APNMF—Hyperion data.

formation. The background is formed of a brick wall. Forty bands uniformly extracted from the image cube were used.

The third data [Fig. 3(c)] correspond to a 256 × 256 pixel subset of a Hyperion urban image of Palo Alto, CA, with 181 spectral bands (from 0.4 to 2.5 μm) and 30-m spatial resolution. The scene corresponds to a combination of urban development and water with a significant number of pixels being mixed.

The two parallelizations do not change the semantic of the original algorithms. This was also confirmed in experiments where, for every parallel run, we also measured the sequential algorithm accuracy. In all cases, the accuracy values were identical at each round. Figs. 4–6 show the error versus the number of rounds for the three data sets when applying NMF and APNMF for the first 50 rounds and $\alpha = 0.001$, $\beta = 0.1$, and $\sigma = 0.01$. The solid line corresponds to NMF, and the dashed line corresponds to ANMF. Error is computed as the root-mean-square error between $\mathbf{x}$ and the estimates $\mathbf{Ws}$ (5). In all cases, the relative stability of the error has reached 0.01 (i.e., less than 1% change). However, runs up to 500 rounds continued to show decreases in the error. Previous research showed that while the multiplicative update in NMF can lead

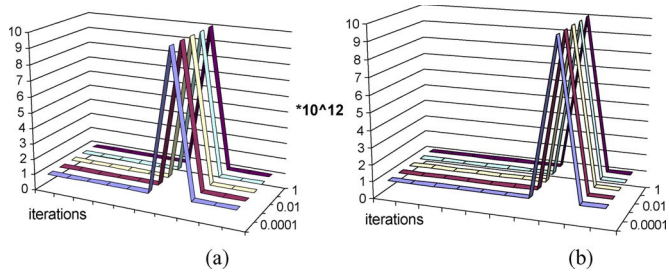

Fig. 7. Fluctuation of (a) α_s and (b) α_W in the first ten rounds for one of the data sets used (Hydice) for various starting values of α (from 0.0001 to 1).

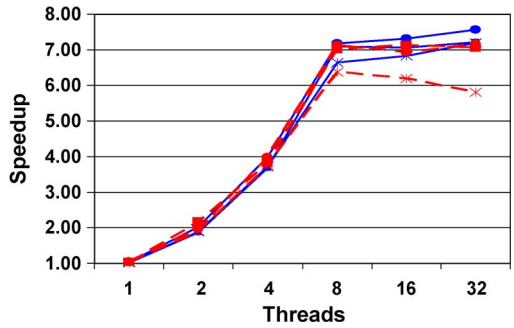

Fig. 8. Speedup for (solid line) PNMF and (dashed line) PAPNMF versus the thread count for (stars) Hydice, (squares) SOC 700, and (circles) Hyperion.

to local minima, the projected gradient approach converges to the absolute minima [14].

The choice of initial value for α in APNMF does not affect the convergence. Previous research suggested that the values for α_s and α_W at each iteration should be computed independent of each other and be based on the respective values obtained in the previous iteration [14]. Fig. 7 shows the result of our experiments on the Hydice data where the initial choices of α were 0.0001, 0.001, 0.01, 0.1, and 1, respectively, and the algorithm was started with the same random $\mathbf{W}$ and $\mathbf{s}$. In all five cases, the same values, both α_s and α_W, were generated. Moreover, the values exhibit relative stability, indicating that, most of the time, the average number internal iterations in APNMF(k_s and k_W) will not have a significant impact on the running times. Similar behavior was noticed in experiments on the other two data sets.

To test the speedup of the parallel algorithms, we generated ten different random initializations for $\mathbf{W}$ and $\mathbf{s}$ and run the four algorithms with varying numbers of parallel processes (one, two, four, eight, 16, and 32, respectively).

Fig. 8 shows the speedup obtained when the parallel algorithms were used. The dashed lines correspond to PAPNMF runs, and the solid lines correspond to PNMF runs. Each image has different markers for their datapoints: Hydice (stars), SOC 700 (squares), and Hyperion (circles). The results are in solid lines for PNMF and in dashed lines for PAPNMF. The speedup was computed as

$$\text{Speedup} = \frac{T_{\text{sequential}}}{T_{\text{parallel}}}. \tag{14}$$

The results indicate that the speedup improves as the number of threads employed increases to eight and decreases or

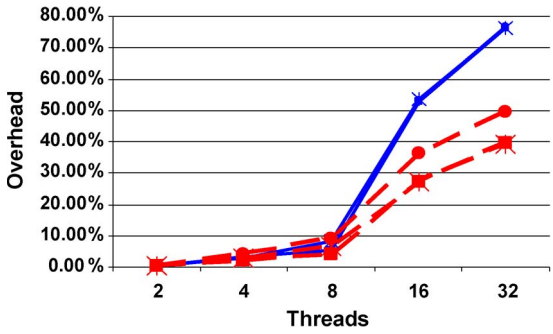

Fig. 9. Synchronization overhead for (solid line) PNMF and (dashed line) PAPNMF versus thread count for (stars) Hydice, (squares) SOC 700, and (circles) Hyperion.

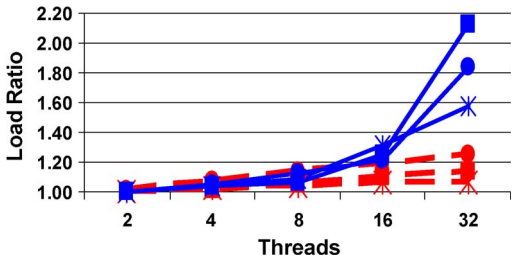

Fig. 10. Load ratio for (solid line) PNMF and (dashed line) PAPNMF versus thread count for (stars) Hydice, (squares) SOC 700, and (circles) Hyperion.

fluctuates afterward. This is due to the fact that the machine used to run the experiments had eight processors.

A multithreaded level higher than eight would result in the threads competing with each other. Since both algorithms required synchronization steps, the thread concurrence leads to a stabilization of performance.

Most encouraging is the fact that, while the number of threads is increased up to fourfold (compared with the processor available), the speedup for each run does not decrease significantly. This suggests that the communication and synchronization costs are low. To better understand the behavior of the two algorithms, we compute the synchronization overhead as

$$\text{Overhead} = \frac{T_{\text{parallel}} - \text{Average Thread Time}}{T_{\text{parallel}}}. \qquad (15)$$

Fig. 9 shows the overhead using the same line type as Fig. 8. The individual thread time excluded the period of time when the thread was blocked waiting for synchronization. The overhead values are low (under 10%) up the number of processor and then start to increase significantly. This was expected since, beyond the number of processors, the threads will start competing with each other, considerably increasing the synchronization costs.

Finally, we computed the load ratio by dividing the maxima by the minima processing times for the threads (Fig. 10). As the number of threads increases, the ratio also increases. This is explained by the fact that the thread loads increase in imbalance (as the number of bands is no longer split evenly), and on the increased concurrence for processor time.

V. Conclusion

Parallel computing is of significant interest in hyperspectral imaging where the design of efficient algorithms is constrained by processing time restrictions and growing data sizes. We have investigated the parallelization of NMF when employed for feature extraction in hyperspectral imagery. Two algorithms previously employed for general blind source separation were adapted to model the linear unmixing problem by scaling the resulting sources. The algorithms were next parallelized, resulting in a computational speedup. Coupled with previous work that suggests that NMF can be successfully employed for spectral unmixing, our approach improves its computational performance and supports its employment for the processing of hyperspectral data. Given an expected increase in spatial and spectral resolutions, speeding up and efficient unmixing method is a worthwhile endeavor.

References

[1] J. A. Richards and X. Jia, *Remote Sensing Digital Image Analysis*. New York: Springer-Verlag, 1999.

[2] D. Landgrebe, *Signal Theory Methods in Multispectral Remote Sensing*. Hoboken, NJ: Wiley, 2003.

[3] P. Martinez, R. M. Perez, A. Plaza, P. L. Aguilar, M. C. Cantero, and J. Plaza, "Endmember extraction algorithms for hyperspectral images," *Ann. Geophys.*, vol. 49, no. 1, pp. 93–101, Feb. 2006.

[4] J. W. Boardman, F. A. Kruse, and R. O. Green, "Mapping target signatures via partial unmixing of AVIRIS data," in *Proc. Summaries V JPL Airborne Earth Sci. Workshop*, Pasadena, CA, 1995.

[5] M. E. Winter, "N-FINDR: An algorithm for fast autonomous spectral end-member determination in hyperspectral data," in *Proc. SPIE Imaging Spectrom. V*, 1999, pp. 266–275.

[6] A. P. Plaza, P. Martinez, R. M. Perez, and J. Plaza, "Spatial/spectral endmember extraction by multi-dimensional morphological operations," *IEEE Trans. Geosci. Remote Sens.*, vol. 40, no. 9, pp. 2025–2041, Sep. 2002.

[7] J. Bowles, P. J. Palmadesso, J. A. Antoniades, M. M. Baumback, and L. J. Rickard, "Use of filter vectors in hyperspectral data analysis," in *Proc. SPIE Infrared Spaceborne Remote Sens. III*, 1995, pp. 148–157.

[8] P. J. Ready and P. A. Wintz, "Information extraction, SNR improvement, and data compression in multispectral imagery," *IEEE Trans. Commun.*, vol. COM-21, no. 10, pp. 1123–1130, Oct. 1973.

[9] S. A. Robila, "Independent component analysis (ICA)," in *Advanced Image Processing Techniques for Remotely Sensed Hyperspectral Data*, P. K. Varshney and M. K. Arora, Eds. New York: Springer-Verlag, 2004, pp. 109–132.

[10] D. Lee and H. Seung, "Algorithms for non-negative matrix factorization," in *Advances in Neural Processing*. Cambridge, MA: MIT Press, 2000.

[11] A. Plaza, D. Valencia, J. Plaza, and C.-I Chang, "Parallel implementation of endmember extraction algorithms from hyperspectral data," *IEEE Trans. Geosci. Remote Sens. Lett.*, vol. 3, no. 3, pp. 334–338, Jul. 2006.

[12] K. Asanovic, R. Bodik, B. C. Catanzaro, J. J. Gebis, P. Husbands, K. Keutzer, D. A. Patterson, W. L. Plishker, J. Shalf, S. W. Williams, and K. A. Yelick, "The landscape of parallel computing research: A view from Berkeley," EECS Dept., Univ. California, Berkeley, CA, Tech. Rep. No. UCB/EECS-2006-183, 2006.

[13] Intel, *Teraflops Research Chip*. accessed online Mar. 2008. [Online]. Available: http://techresearch.intel.com/articles/Tera-Scale/1449.htm

[14] C. J. Lin, "Projected gradient methods for nonnegative matrix factorization," *Neural Comput.*, vol. 19, no. 10, pp. 2756–2779, Oct. 2007.

[15] D. P. Bertsekas, "On the Goldstein–Levitin–Polyak gradient projection method," *IEEE Trans. Autom. Control*, vol. AC-21, no. 2, pp. 174–184, Apr. 1976.

[16] P. Pauca, J. Piper, and R. Plemmons, "Nonnegative matrix factorization for spectral data analysis," *Linear Algebra Appl.*, vol. 416, no. 1, pp. 29–47, 2006.

[17] S. A. Robila and L. Maciak, "New approaches for feature extraction in hyperspectral imagery," in *Proc. IEEE LISAT*, 2006, pp. 1–7.

[18] United States Geological Survey, *Hyperion Urban Sample*. accessed online Jan. 2008. [Online]. Available: http://eo1.usgs.gov/sampleurban.php

[19] J. Dongarra, I. Foster, G. C. Fox, W. Gropp, K. Kennedy, L. Torczon, and A. White, *The Sourcebook of Parallel Computing*. San Mateo, CA: Morgan Kaufmann, 2002.

[20] T. Kielmann, P. Hatcher, L. Bougé, and H. E. Bal, "Enabling Java for high-performance computing," *Commun. ACM*, vol. 44, no. 10, pp. 110–117, Oct. 2001.