High Performance Computing of Fast Independent Component Analysis for Hyperspectral Image Dimensionality Reduction on MIC-based Clusters

Minquan Fang
Academy of Ocean Science and Engineering
National University of Defense Technology
Changsha, China
fmq@hpc6.com

Yi Yu
Academy of Ocean Science and Engineering
National University of Defense Technology
Changsha, China
yuyi2019@nudt.edu.cn

Weimin Zhang
Academy of Ocean Science and Engineering
National University of Defense Technology
Changsha, China
wmzhang104@139.com

Heng Wu
Department of Computer
Texas TECH University
Lubbock, USA
heng05.wu@ttu.edu

Mingzhu Deng
School of Computer
National University of Defense Technology
Changsha, China
dk_nudt@126.com

Jianbin Fang
School of Computer
National University of Defense Technology
Changsha, China
j.fang@nudt.edu.cn

***Abstract*—Fast independent component analysis (FastICA) for hyperspectral image dimensionality reduction is computationally complex and time-consuming due to the high dimensionality of hyperspectral images. By analyzing the FastICA algorithm, we design parallel schemes for covariance matrix calculating, white processing and ICA iteration at three parallel levels: multicores, many integrated cores (MIC), and clusters. Then we present a series of optimization methods for different hotspots, and measure their performance effects. All the work has been implemented in a framework called Ms-FastICA. Our experiments on the Tianhe-2 Supercomputer show that the Ms-FastICA algorithm has a good scalability, and it can reach a maximum speed-up of 410 times on 64 nodes with 192 Intel Xeon Phis.**

Keywords- many integrated cores; fast independent component analysis; hyperspectral image dimensionality reduction; high performance computing; load balancing of lower triangular matrix

I. INTRODUCTION

Hyperspectral Remote Sensing is widely used in military, agriculture, environmental science, geology, oceanography etc. [1] and required real-time processing. Due to many bands, big data size, strong relation and high redundancy, directly processing for the hyperspectral image may lead to hardly train for sample categories, curse of dimensionality and empty space phenomenon [2-4]. Therefore, scholars firstly reduce the dimension for the hyperspectral image by specific mapping, high dimensionality data transforms into low dimensionality without information loss.

There are two dimensionality reduction methods: linear and nonlinear. The former includes the typical principal component analysis (PCA) [5] and independent component analysis (ICA) [6] algorithms. The isometric feature mapping (ISOMAP) [7] and locally linear embedding (LLE) [8] algorithms are nonlinear. Due to the characteristics of hyperspectral image, such as many bands, big data size, dimensionality reduction becomes a time-consuming process. Therefore, scholars focus on innovative algorithms and parallel reduction.

Since Intel released its many integrated cores (MIC) product Intel Xeon Phi in 2012[9], MIC architecture has become a research hotspot. The Tianhe-2 Supercomputer, featuring with 3 MICs per node, has taken the crown of TOP500 for four times [10]. In this paper, we focus on the parallel methods to process hyperspectral images on Tianhe-2 using a single algorithm of dimensionality reduction.

Some scholars focus on hyperspectral image processing and parallel computing and proposed some algorithms. Valencia et al. [11] investigated the parallel processing for hyperspectral images on a cluster using HeteroMPI. Plaza et al. [12] proposed a parallel classification algorithm using neural network. There are also some studies for processing hyperspectral images on CPU/GPU heterogeneous systems. Sánchez et al. [13] implement the hyperspectral image unmixing on GPUs; Platoš et al. [14] research the non-negative matrix factorization on GPU; Ramalho et al. [15] present an implementation of FastICA on a GPU. However, To the best of our knowledge, there is no related work on the MIC-based heterogeneous system such as Tianhe-2.

In this paper, we investigate FastICA [16-17], a typical hyperspectral image dimensionality reduction algorithm, on MIC-based clusters, aiming to achieve high performance and meet the real-time processing requirement. In particular, our contributions include:

- We design parallel schemes for the hotspots of FastICA, including covariance matrix calculation, white processing and ICA iteration.

1530-2016/15 $31.00 © 2015 IEEE
DOI 10.1109/ICPPW.2015.23

- We present a series of constructive optimization methods for FastICA and measure their performance impact on a single node.
- We implement our work, as Ms-FastICA for hyperspectral image dimensionality reduction, and show the overall performance (speedup and scalability) on Tianhe-2 Supercomputer.

II. FAST INDEPENDENT COMPONENT ANALYSIS AND HOTSPOTS

A. Fast independent component analysis

Fast independent component analysis mainly includes three forms: maximum negative entropy, maximum likelihood, and kurtosis. In this paper, we select the one based on maximum negative entropy, and extract the independent source sequentially. This algorithm embodies the traditional linear transform projection pursuit, and uses an optimization method which is fixed-point iteration. Thus, it can make the iteration converge more rapidly and is more robust.

The hyperspectral image data is represented by a $B \times S$ matrix $\boldsymbol{X}$ (where B is the number of bands, and S is the number of pixels in each band. S is computed by $S=W \times H$, W indicates width and H indicates height). To reduce the dimensionality of a hyperspectral image, 6 steps are described as follows:

Step1. Calculate the covariance matrix $\boldsymbol{\Sigma}$:

$$\boldsymbol{\Sigma}^{ij} = \frac{1}{S-1}[\sum_{k=1}^{S}(\boldsymbol{X}_k^i - \overline{\boldsymbol{X}^i})(\boldsymbol{X}_k^j - \overline{\boldsymbol{X}^j})] \quad (1)$$

where i and j is the index of covariance matrix.

Step2. Decompose the covariance matrix $\boldsymbol{\Sigma}$ into eigenvalue vector and eigenvector matrix:

$$\boldsymbol{\Sigma} = \boldsymbol{V}\boldsymbol{D}\boldsymbol{V}^{\mathrm{T}} \quad (2)$$

in which $\boldsymbol{D}$ is the eigenvalue vector and $\boldsymbol{V}$ is the eigenvector matrix. Usually we choose m independent components according to $\boldsymbol{D}$ and a threshold value T (in this paper, we set it to be 99%).

Step3. The white matrix $\boldsymbol{M}$ can be calculated by

$$\boldsymbol{M} = \boldsymbol{D}^{-1/2}\boldsymbol{V}^{\mathrm{T}} \quad (3)$$

Step4. White processing is calculated by

$$\boldsymbol{Z} = \boldsymbol{M}\boldsymbol{X}' \quad (4)$$

where $\boldsymbol{X}'$ is the result of zero mean process for $\boldsymbol{X}$.

Step5. ICA iteration.

Step5.1 Initialize $\boldsymbol{W}_{\mathrm{i}}$, which is the i[th] column of $\boldsymbol{W}$.

Step5.2 Update $\boldsymbol{W}_{\mathrm{i}}$ by

$$\boldsymbol{W}_i = E\{\boldsymbol{Z}g(\boldsymbol{W}_i^{\mathrm{T}}\boldsymbol{Z})\} - E\{g'(\boldsymbol{W}_i^{\mathrm{T}}\boldsymbol{Z})\}\boldsymbol{W}, \quad (5)$$

where g is nonlinear function. When g takes $g(y) = y^3$, the ICA iteration converges rapid and robust in our test.

Step5.3 Orthogonalization for $\boldsymbol{W}_{\mathrm{i}}$:

$$\boldsymbol{W}_i = \boldsymbol{W}_i - \sum(\boldsymbol{W}_i^{\mathrm{T}}\boldsymbol{W}_j)\boldsymbol{W}_j \ , \quad (6)$$

in which $\boldsymbol{W}_{\mathrm{j}}$ express the iterated columns of $\boldsymbol{W}$.

Step5.4 Normalization for $\boldsymbol{W}_{\mathrm{i}}$

$$\boldsymbol{W}_i = \boldsymbol{W}_i / \| \boldsymbol{W}_i \| \ . \quad (7)$$

Step5.5 Check whether the iteration is convergent.

- Goto step5.2 if not.
- Exit if both iteration is convergent and all $\boldsymbol{W}_{\mathrm{i}}$ are iterated.
- Goto Step5.1 and initialize the next $\boldsymbol{W}_{\mathrm{i}}$ (i++) .

Step6. Independent components are transformed by

$$\boldsymbol{Y} = \boldsymbol{W}\boldsymbol{Z} \quad (8)$$

B. Hotspots analysis for FastICA

We run the FastICA program with hyperspectral image data (W=614, H=1087, B=224), and measure the execution time of different steps. The ratio of different steps in the total computing time is shown in Table I. It shows that covariance matrix calculation, white processing and ICA iteration takes up 99% of the total processing time. Therefore these are the hotspots that we need consider for parallelization.

TABLE I. TIME DISTRIBUTION OF FASTICA

Subroutine	Time percent
Covariance matrix calculation	19.8%
Eigenvalue	0.5%
White processing	60.7%
ICA iteration	18.5%
IC transformation	0.5%

III. PARALLELIZATION AND OPTIMIZATION FOR COVARIANCE MATRIX CALCULATION

A. Distributed paralle l scheme for covariance matrix calculation

Covariance use the following formula to calculate the deformation

$$\boldsymbol{\Sigma}^{ij} = \frac{1}{S-1}[\sum_{k=1}^{S}(\boldsymbol{X}_k^i - \overline{\boldsymbol{X}^i})(\boldsymbol{X}_k^j - \overline{\boldsymbol{X}^j})]$$

$$= \frac{1}{S-1}\left[\sum_{k=1}^{S}\left(\boldsymbol{X}_k^i\boldsymbol{X}_k^j + \overline{\boldsymbol{X}_k^i}\,\overline{\boldsymbol{X}_k^j} - \boldsymbol{X}_k^i\overline{\boldsymbol{X}_k^j} - \overline{\boldsymbol{X}_k^i}\boldsymbol{X}_k^j\right)\right]$$

$$= \frac{1}{S-1}\left[\sum_{k=1}^{S} \boldsymbol{X}_k^i \boldsymbol{X}_k^j + S\overline{\boldsymbol{X}^i}\,\overline{\boldsymbol{X}^j} - \overline{\boldsymbol{X}^j}\sum_{k=1}^{S} \boldsymbol{X}_k^i - \overline{\boldsymbol{X}^i}\sum_{k=1}^{S} \boldsymbol{X}_k^j\right]$$

$$= \frac{1}{S-1}\left[\sum_{k=1}^{S} \boldsymbol{X}_k^i \boldsymbol{X}_k^j + S\overline{\boldsymbol{X}^i}\,\overline{\boldsymbol{X}^j} - \overline{\boldsymbol{X}^j} S\overline{\boldsymbol{X}^i} - \overline{\boldsymbol{X}^i} S\overline{\boldsymbol{X}^j}\right]$$

$$= \frac{1}{S-1}\left[\sum_{k=1}^{S} \boldsymbol{X}_k^i \boldsymbol{X}_k^j - S\overline{\boldsymbol{X}^i}\,\overline{\boldsymbol{X}^j}\right]$$

$$= \frac{1}{S-1}\left[\sum_{k=1}^{S} \boldsymbol{X}_k^i \boldsymbol{X}_k^j - \frac{1}{S}\sum_{k=1}^{S} \boldsymbol{X}_k^i \sum_{k=1}^{S} \boldsymbol{X}_k^j\right]$$

$$= \frac{1}{S-1}\left[\begin{array}{l}\left(\sum_{k=1}^{s/N} \boldsymbol{X}_k^i \boldsymbol{X}_k^j + \sum_{k=s/N+1}^{2s/N} \boldsymbol{X}_k^i \boldsymbol{X}_k^j + \cdots + \sum_{k=(N-1)s/N+1}^{s} \boldsymbol{X}_k^i \boldsymbol{X}_k^j\right) \\ -\frac{1}{S}\left(\sum_{k=1}^{s/N} \boldsymbol{X}_k^i + \sum_{k=s/N+1}^{2s/N} \boldsymbol{X}_k^i + \cdots + \sum_{k=(N-1)s/N+1}^{s} \boldsymbol{X}_k^i\right) \\ *\left(\sum_{k=1}^{s/N} \boldsymbol{X}_k^j + \sum_{k=s/N+1}^{2s/N} \boldsymbol{X}_k^j + \cdots + \sum_{k=(N-1)s/N+1}^{s} \boldsymbol{X}_k^j\right)\end{array}\right] \quad (9)$$

The deformation converts the covariance calculation into vector addition reduction ($\Sigma \boldsymbol{X}$) and vector dot product ($\Sigma \boldsymbol{X}^i \boldsymbol{X}^j$), which both have an inborn Severability. By using this Severability, a distributed parallel covariance computing approach can be designed, as shown in Figure 1. Vector addition reduction and vector dot product operation are calculated on each node as sub-tasks, and results are then sent to the root node, where a summary of all the results is conducted and thus covariance value is calculated.

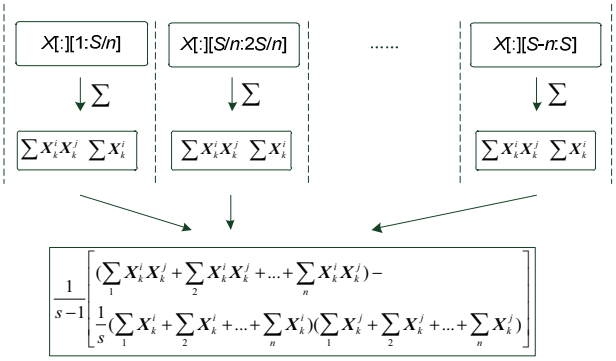

Figure 1. Distributed parallel scheme for covariance

we designed a distributed parallel scheme of covariance matrix calculation for hyperspectral image matrix: 1) evenly divide the hyperspectral image matrix according to the number of pixels; 2) parallel compute vector addition reduction and lower triangular matrix multiplication on each node (covariance matrix is a symmetric matrix, therefore we only need to calculate the lower triangular matrix), and sub-results are generated as such for both the vector (B) and the matrix ($B \times B$); by making use of collective communication, all the sub-results are accumulated to the root node where the overall covariance matrix calculation is performed.

B. Shared memory parallel computing for covariance matrix

In the distributed parallel covariance matrix computing, there are two parallel levels existing within vector addition reduction and the lower triangular matrix multiplication in every node:

- 1) All elements in the sub-results (sum vector and multiplication matrix) are independent, able to be parallelized.
- 2) Any single element in the sub-result has large amount of calculation, which is able to be parallelized.

In order to better use the vector units of CPUs and MICs, we spawn a second level of parallelism for vector operation (which can be regarded as SIMD instruction-level parallelism within a single core), and then we uniformly allocate the first level of parallelism to different processor cores.

C. Optimization for covariance matrix calculation on MIC

In response to the above parallel scheme of covariance matrix calculation, we conduct optimization study based on many integrated cores architecture, and propose a series of optimization methods to enhance its performance.

- 1) Data type deformation. Hyperspectral image data is a kind of image data, which consists of pixel information ranging from 0 to 255, and is stored with the use of unsigned char type (instead of the original float type). There are several advantages behind this data type deformation: (1) Reducing storage usage on the device (MIC), so that larger image data can be handled within a MIC; (2) Reducing the PCI-E communication overhead; (3) Reducing data access latency on MIC.
- 2) Computation decomposition. Compared with the powerful transaction processing capability of CPU, MIC, as coprocessor is better at relatively simple calculations. Therefore we decompose complex computing tasks into a few simple calculations, making it suitable for lightweight threads on MIC coprocessors. By the covariance calculation deformation in equation (9), covariance calculation is transformed into dot product and reduction operation. On this basis, accumulation operation of all the elements in the covariance matrix can be separately extracted, which in fact is to calculate addition reduction and matrix multiplication separately. In this way, the amount of calculation is also reduced, from ($2B$ (B + 1) WH) to (BWH + B (B + 1) WH). That is an around 50% decrease.
- 3) Load balancing for lower triangular matrix: In fact, covariance matrix is symmetric, and only the lower triangular matrix needs to be calculated. Due to the fact that there are more cores in MICs (than a traditional multicore CPU), the parallel proposal is not able to guarantee load balancing of lower triangular matrix. For example, thread 0 calculates one element of the covariance matrix, while the

thread 223 needs to calculate 224 elements, which is load-unbalanced. To this end, we design a new load-balancing approach, shown in Figure 2, whose basic idea is to map the two-dimensional tasks of the lower triangular matrix into one-dimensional tasks, and then distribute all the one-dimensional tasks to ultimately achieve the purpose of load balancing. In this paper, with the help of formula 3 and 4 in Figure 2, we are able to map a two-dimensional index to one-dimensional array and merge the two-layered loop into a one layered loop. In this way, not only load balancing is achieved, but also the degree of parallelism is increased.

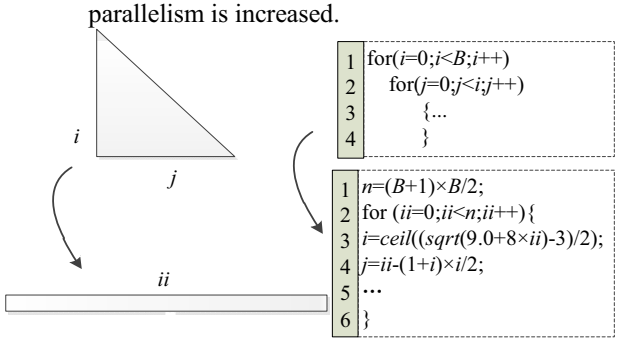

Figure 2. Load balancing for lower triangular matrix

Using the optimizations described above, we can achieve the following results in Figure 3, in which 0 indicates the result without any optimization while 1 to 3 respectively stands for time consumption by adopting optimization methods 1 ~ 3. Experimental results shows that speed increased 3.8 times by using these optimization methods.

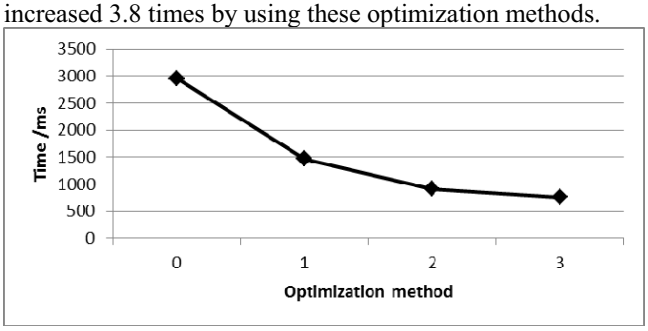

Figure 3. Optimization effect of covariance matrix calculation

IV. PARALLEL WHITE PROCESSING AND OPTIMIZATION STUDIES

A. Parallel white processing scheme

White processing normally includes two procedures: white matrix calculation and white processing. Because the dimension of the white matrix is small, the calculation amount is relatively small, posing no need for parallelism. Then the main consideration goes to white processing $\boldsymbol{Z}=\boldsymbol{MX}$. Extracting white macroscopic model is very similar to matrix multiplication. Therefore we employ a parallel matrix multiplication division plan.

According to the dimension characteristics of $\boldsymbol{M}$ matrix ($m \times B$) and matrix $\boldsymbol{X}$ ($B \times S$), where $S \quad B \quad m$, parallel white processing scheme is proposed in this paper, as shown in Figure 4. In our proposal, distributed parallel white processing method inherits the segmentation method of hyperspectral image in Section III, that is evenly dividing $\boldsymbol{X}$ matrix in the direction of S (the number of pixels). However, in terms of parallel white processing method for multicores in shared memory, matrix $\boldsymbol{M}$ is evenly split.

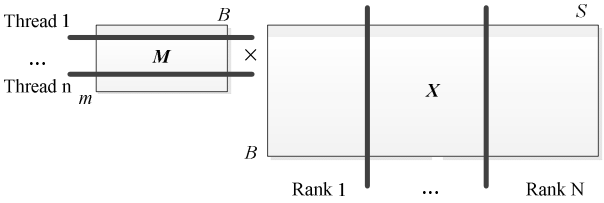

Figure 4. Parallel white processing scheme

B. Parallel white processing optimization on MIC

For parallelizing white processing, based on many integrated cores architecture, we propose a series of optimization strategies, and experiment with different combined test for optimization results, and ultimately get the best combination of optimization strategies.

- 1) Selection for parallel cycles. Hyperspectral image white processing has its own unique characteristics, where the outermost loop is the number of bands m based on the amount of contribution, ranging differently from a dozen more to several dozens. Because of the small degree of loop parallelism of the layer, it is not suitable to be parallelized in the many integrated cores co-processor, where the MIC has over 50 cores, and each has four concurrent threads.
- 2) The selection of Data type. It is similar to the calculation of covariance matrix.
- 3) Loop interchange. When the memory access is not contiguous, cache hit rate will be low, leading to massive memory access overheads. And when the innermost loop is not able to be vectorized, the MIC vector process unit (VPU) would not be effectively utilized, which will greatly affect performance. If the above two aspects exist in the program, loop interchange is needed. Due to the discontiguous memory access cycle, loop interchanging should be performed for the white process, to make data access contiguous by exchanging circulation within the two inner layers.
- 4) Unrolling the nested loop except the innermost cycle. There are totally 3 layers of loops in white processing. The m number of the outermost layer (band number after dimensionality reduction) is small and not suitable for MIC parallelization. In order to make it suitable in many integrated cores architecture, the outer two layers of nested loops are unrolled to increase the cycle number of outermost layer and meanwhile to reduce the overheads of the

parallel fork-join regions. In the meantime of nested loop unrolling, the innermost loop is kept unchanged to ensure vectorization and contiguous memory accessing.

- 5) Matrix transposition for hyperspectral image. White processing accesses hyperspectral image matrix based on the column, and in order to make access contiguous, matrix transpose strategy can be used. By transposing matrix, the rows and columns of original matrix will be exchanged to achieve contiguously accessing. However, this strategy will introduce transpose overheads. If the introduced transpose overhead is larger than the performance gains, it would be the undesirable and abandoned. (In our experiments, transpose overhead is included).

The third and fifth optimization methods cannot be used at the same time, because their purpose is the same, namely making data access contiguous. Figure 5 shows the combined effect of the optimization methods:

- Group 0 is not optimized;
- Group 1 used the method 1;
- Group 2 is the combination of the methods 1 and 2;
- Group 3 is the combination from methods 1 to 3;
- Group 4 is consisted of methods 1 to 4;
- Group 5 is for optimization method 1, 2, 5;
- Group 6 uses method 1,2,4,5.

Experimental results show that the group 6 gains the best performance.

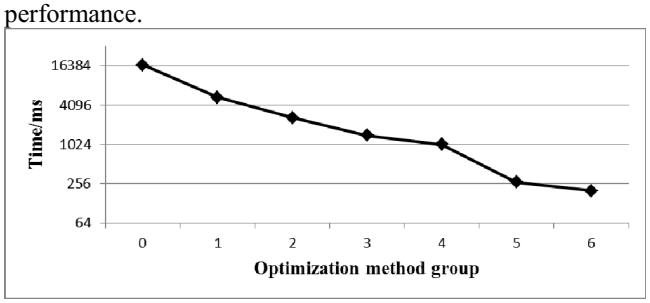

Figure 5. Optimization effect for white process

V. ICA PARALLEL ITERATION AND OPTIMIZATION METHODS

A. ICA parallel iteration method

There are 5 steps in ICA iteration, and calculation is mainly concentrated in Step5.2 formula (5). The operation is complex, where first, an abstraction model is needed. Then given the model, a study of parallel tasks split is conducted to obtain the parallel ICA iteration macro model (Figure 6). In the model, the value of S, which is the number of pixel is maximum while the other parameter values is relatively small, so the parallel programs for distributive storage and shared storage are similar, namely to evenly divide the number of pixels.

B. ICA iteration optimization on MIC

ICA iteration contains a large number of small-scale iterative calculations, and we launch optimization study on a single iteration. Main optimization strategies taken are as follows:

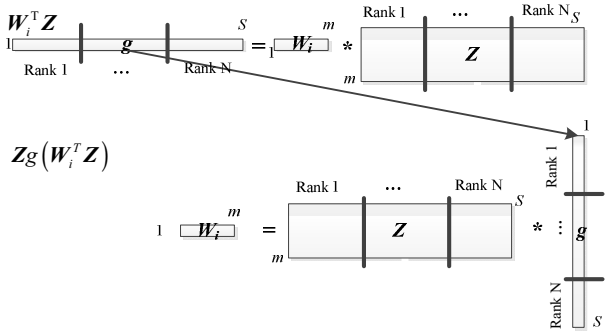

Figure 6. ICA parallel iteration method

- 1) Matrix transposition for contiguous memory accessing. In the calculation of $\boldsymbol{W}_i^T\boldsymbol{Z}$, the access of matrix $\boldsymbol{Z}$ is column-majored while code data is stored by row in C language, making it unable to contiguously access memory. By transposing matrix $\boldsymbol{Z}$ to get involved in computing, contiguous accessing is achieved.
- 2) In ICA iteration, $\boldsymbol{W}_i$ involved in the calculation of matrix $\boldsymbol{W}$ is storing by column and $\boldsymbol{W}_i$ is heavily involved in massive computing in an iteration. Therefore, a temp array is used to store elements in $\boldsymbol{W}_i$ to participate in the calculation. In C language, matrix is stored in rows, making access by column plaguing cache hit rate. By making use of the intermediate array *tmp*[:] to store $\boldsymbol{W}_i$ elements, memory accessing is contiguous, achieving higher cache hit rate and better performance.

In our program, the above optimization methods are implemented respectively, and time cost for ICA iteration step before and after each optimization method is recorded and shown in Table II, where 0 indicates that no optimization is applied while 1~2 respectively represents the time cost after sequentially employing optimization method 1 ~ 2.

TABLE II. OPTIMIZATION EFFECT FOR ICA ITERATION

optimization method	time/ms
0	13.33
1	5.03
2	3.40

VI. A MULTILEVEL PARALLEL ALGORITHM OF MS-FASTICA

A. CPU / MIC heterogeneous optimization

The CPU + MIC system is a typically heterogeneous architecture, and CPU and MIC can compute or communicate at the same time. Thus with collaborative computing and communicating, we can use CPU and MIC simultaneously. In our algorithm, the calculation of covariance matrix (COV) and hyperspectral image matrix $\boldsymbol{X}$ transposition can be executed simultaneously (Figure 7). In this way $\boldsymbol{X}$ transposition overhead can be overlapped.

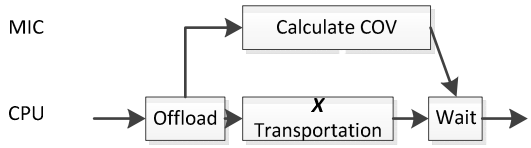

Figure 7. Cooperative computing between COV and X transposition

B. Ms-FastICA multi-level parallel algorithms

By taking the parallel schemes of hotspots and the aforementioned optimizations strategies for CPU/MIC systems, we implement Ms-FastICA, a multilevel parallel algorithm for fast independent component analysis for hyperspectral image dimensionality reduction on MIC-based clusters, with MPI, OpenMP, language extensions for offload (LEO).

Algorithm 1: A multilevel parallel algorithm for FastICA on MIC clusters (Ms-FastICA)

```
MPI_Init();
...
parallel calculate ΣX^i X^j on MIC;//signal(mul)
parallel calculate ΣX on CPU;
parallel transpose X to Xt on CPU;
transfer Xt from CPU to MIC
wait(mul)//synchronization CPU and MIC
MPI_Reduce(ΣX,B);
MPI_Reduce(ΣX^i X^j,B*B);
parallel calculate cov[B*B] on CPU in root node;
...
if (rank=0) then
|   calculate m on CPU;
|   calculate M on CPU;
end
MPI_Bcast(&m,1);
MPI_Bcast(M,m*B);
parallel calculate Z on MIC;
parallel transposition Z to Zt on MIC;
repeat  //parallel ICA iteration
|   if rank=0 then  initialize W_i//on CPU
|   MPI_Bcast(W_i,m)
|   repeat
|   |   W_i=E{Zg(W_i^T Z)}-E{(W_i^T Z)}W //parallel on MIC
|   |   MPI_Reduce(W_i,m);
|   |   MPI_Bcast(W_i,m);
|   |   W_i  orthogonalization //on CPU
|   |   W_i  normalization //on CPU
|   until(converged)
|   count++;
until(count=m)
parallel IC transformation on MIC;
...
MPI_Finalize();
```

VII. EXPERIMENTAL RESULTS

A. Experimental environment

Our programs were compiled by Intel C/C++ compiler 2013 sp1.1.106 and MPICH3, and ran on the TianHe-2 Supercomputer. Each node on TianHe-2 has two 12-core Xeon CPUs and three 57-core Intel Xeon Phi coprocessors. Optimization options -O3 has been selected for the serial program as the reference.

Due to different random numbers in the ICA iteration, ICA iterations differ from each other, which thus lead to different time consumption. To accurately compare the program performance, we use 1000 iterations.

B. Scalability Analysis

For hyperspectral image (W=1562, H=13910, B=224) dimensionality reduction processing, we measure the execution time of the key parallel hotspot respectively on CPUs, 1MIC, 3MICs with nodes number ranging from 1, 2, 4, 8, 16, 32, 64 to 128.

Figure 8 shows the execution time for covariance matrix calculation, Figure 9 shows that for white processing, and Figure 10 is that for ICA iteration computation. Because the hyperspectral image data is too large to be stored within a single MIC, the program cannot run on a single MIC node.

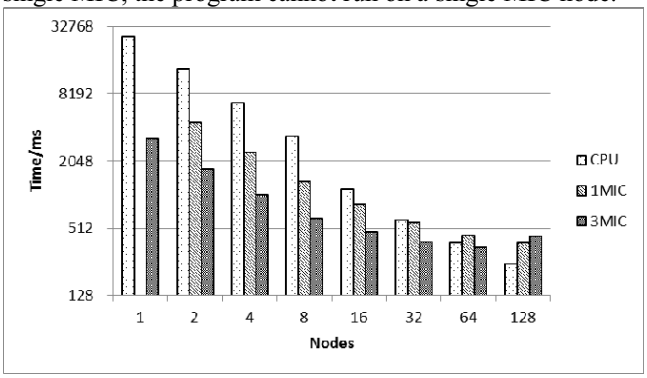

Figure 8. Scalability of covariance matrix calculation

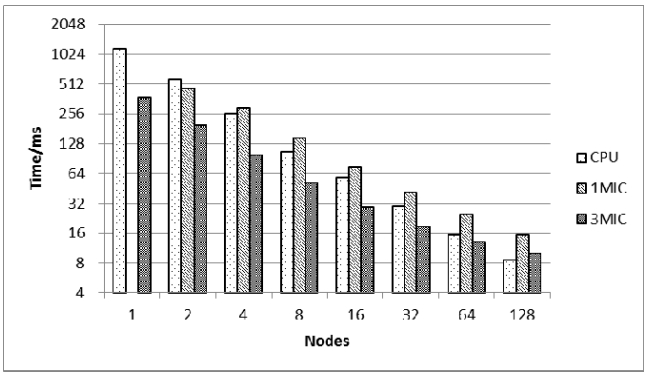

Figure 9. Scalability of white processing

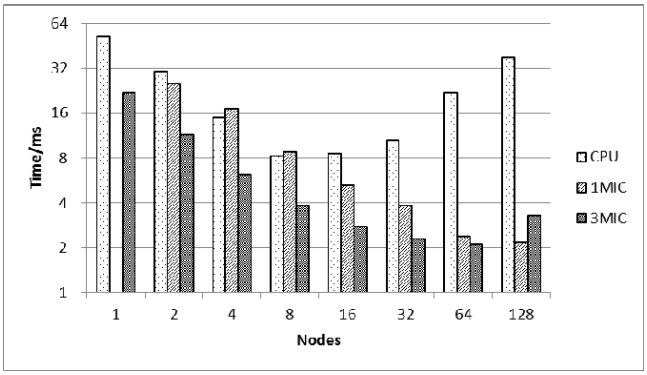

Figure 10. Scalability of ICA iteration

Figure 8 reveals that covariance matrix calculation has a good scalability, We also see that MIC can obtain higher performance than using only CPUs, while using 3 MICs runs

the fastest when the nodes is less than 128 nodes. However, when using 128 nodes, the CPU parallel version runs the fastest. And when we use more than 64 nodes, the CPU version runs faster than that of 1 MIC. Therefore we come to the conclusion that CPU parallel version runs faster than the MIC version when the calculation scale is not large enough.

Figure 9 shows the parallel white processing has a good scalability. Further we notice that 3 MICs on each node can run fastest when the nodes are less than 128, and then the CPU parallel version runs the fastest. We believe that the CPU parallel version can run faster than the MIC version when the calculation scale is not large enough. When the task is divided into 64 partitions, the calculation scale is so small that CPU runs faster than MIC.

Figure 10 shows the parallel ICA iteration has poor scalability on the CPUs, which is only expanded to 16 nodes. When over 16 nodes are used, performance will decrease. In my opinion, because the CPU computing is not stable enough, especially for applications like the ICA iterations which involve massive MPI communication in each iteration. The reasons why CPU is not stable are still under investigation. On the contrary, because the MIC coprocessor is specialized for calculation, it can obtain more stable computing performance, and MIC coprocessors can get the best performance on 64 nodes. Due to the larger proportion of the MPI communication, the performance will suffer when using more nodes.

The total execution time is shown in Figure 11, which tells our Ms-FastICA parallel algorithm has a good scalability. When parallelized on CPU, scalability is poor, and can only reach 16 nodes. But parallel computing on MICs achieve a better scalability. Also, using 3 MICs in single node performs better than using 1MIC.

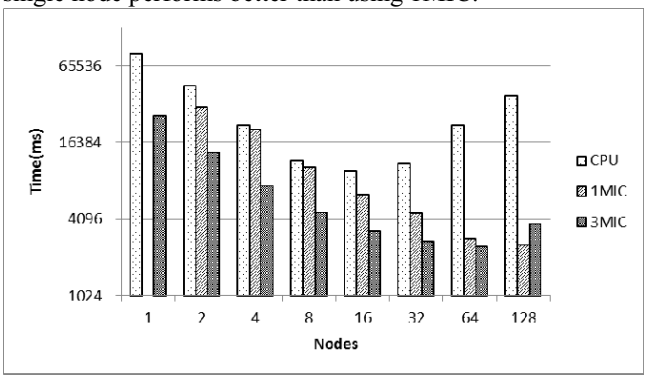

Figure 11. Scalability of Ms-FastICA

C. Speed-up

Statistics of speedup between our code and the optimal serial code on different number of nodes and different parallel computing resources (CPU, 1MIC, 3MIC) is shown in Figure 12. We see that a 104 times speedup on 16-nodes with 384 CPU cores, and a 395 times speedup on 128 nodes with 128 MICs have been achieved by using our approach. In particular, the maximum speed-up of 410 times is gained on the 64 nodes with 192 MICs.

D. performance bottleneck of Parallel algorithm

The ratio of different steps in the total time consumption under the maximum speedup configuration is shown in Figure 13, where column 1~6 respectively represents covariance matrix computing, eigenvalues and eigenvectors calculations, white processing, Z transposition, one time of ICA iteration (1000 times in all), IC transformation.

We see that the covariance matrix calculation still has some optimization space while eigenvalues and eigenvectors computing become a new performance bottleneck. And cumulative time of all ICA iterations is also another performance bottleneck. In order to further enhance performance, the aforementioned three aspects will be looked into for further work.

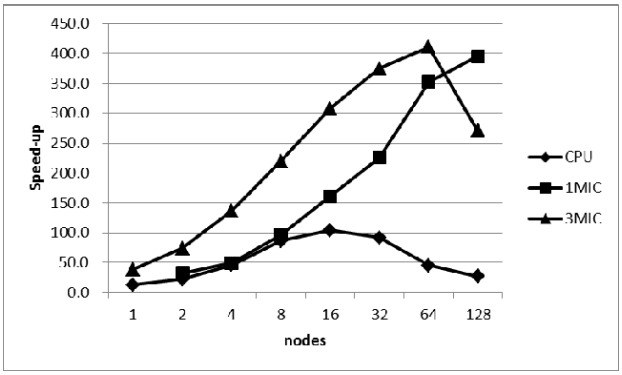

Figure 12. Speed-up of Ms-FastICA

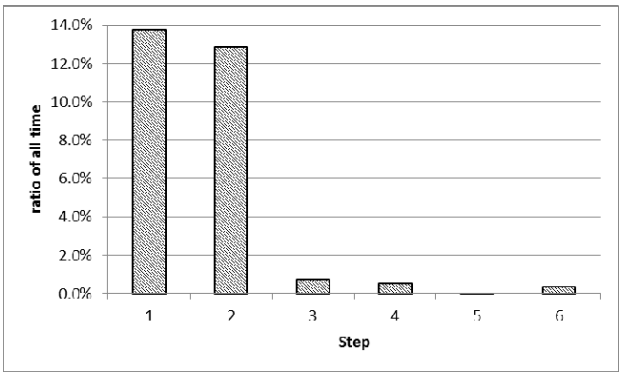

Figure 13. Time distribution of Ms-FastICA

VIII. CONCLUSIONS AND FUTURE LINES

In this paper, based on the analysis of FastICA algorithm for hyperspectral image dimensionality reduction, we design some parallel schemes for covariance matrix calculating, white processing and ICA iteration, and then present a series of optimization methods. Finally, we implement our work as Ms-FastICA on MIC-based clusters. Experimental results approve the Ms-FastICA algorithm achieves an excellent scalability. It can reach a maximum speed-up of 410 times on 64 nodes with 3 MICs per node of the Tianhe-2 Supercomputer.

In our experiments we observe that the FastICA algorithm for hyperspectral image dimensionality reduction can obtain an excellent performance on Intel Xeon Phi

clusters. Therefore, we conclude that the MIC-based heterogeneous system (e.g. Tianhe-2) is a good candidate to process hyperspectral images. For future work, we would like to implement our framework on GPU-based clusters. Also, we plan to investigate other algorithms for dimensionality reduction on many-core architectures such as GPUs and MICs.

ACKNOWLEDGMENT

This is supported by the National Natural Science Foundation of China under Grant Nos: 61272146, 41375113, and 41305101.

REFERENCES

[1] Green RO, Eastwood ML, Sarture CM, et al. Imaging spectroscopy and the airborne visible/infrared imaging spectrometer(AVIRIS)[J]. Remote Sensing of Environment, 1998, 65(3): 227-248

[2] Green AA, Berman M, Switzer P, et al. A transformation for ordering multispectral data in terms of image quality with implications for noise removal[J]. IEEE Trans on Geoscience and Remote Sensing, 2000, 26(1): 65-74

[3] Kaarna A, Zemcik P, Kalviainen H, et al. Compression of multispectral remote sensing images using clustering and spectral reduction[J]. IEEE Trans on Geoscience and Remote Sensing, 2000, 38(2): 1073-1082

[4] Scott DW, Thompson JR. Probability density estimation in higher dimensions[C]// Computer Science and Statistics: Proc of the 15th Symp on the Interface. Amsterdam: North-Holland, 1983: 173–179

[5] Jolliffe IT. Principal Component Analysis[M]. New York: Sprinner, 2002

[6] Hyvärinen A, Oja E. Independent component analysis: Algorithm and applications[J]. Neural Networks, 2000, 13(4/5): 411-430

[7] Tenenbaum J, De Silva V, Langford JC. A global geometric framework for nonlinear dimensionality reduction[J]. Science, 2000, 290(5500): 2319-2323

[8] Roweis ST, Saul LK. Nonlinear dimensionality reduction by locally linear embedding[J]. Science, 2000, 290(5500): 2323-2326

[9] Jeffers J, Reinders J. Intel Xeon Phi Coprocessor High Performance Programming[M]. Translated by Chen Jian, et al. Beijing: Posts & Telecom Press, 2014

[10] Meuer H, Strohmaier E, Dongarra J, et al. TOP500 supercomputer sites[EB/OL]. 2014 [2015-04-27]. http://www.top500.org/lists/2014/06/

[11] Valencia D, Lastovetsky A, O'Flynn M, et al. Parallel processing of remotely sensed hyperspectral images on heterogeneous networks of workstations using HeteroMPI[J]. International Journal of High Performance Computing Applications, 2008, 22(4): 386-407

[12] Plaza J, Plaza A, Pérez R, et al. Parallel classification of hyperspectral images using neural networks[J]. Computational Intelligence for Remote Sensing Studies in Computational Intelligence, 2008, 133: 193-216

[13] Sánchez S, Ramalho R, Sousa L, et al. Real-time implementation of remotely sensed hyperspectral image unmixing on GPUs[J/OL]. Journal of Real-Time Image Processing, 2012 [2015-04-27]. http://link.springer.com/article/10.1007/s11554-012-0269-2

[14] Platoš J, Gajdoš P, Krömer P, et al. Non-negative matrix factorization on GPU[C]// Proc of the 2nd Int Conf on Networked Digital Technologies. Heidelberg: Springer, 2010: 21-30

[15] Ramalho R, Tomas P, Sousa L. Efficient independent component analysis on a GPU[C]// Proc of the 10th IEEE Int Conf on Computer and Information Technology. Piscataway, NJ: IEEE Computer Society, 2010: 1128-1133

[16] Hyvärinen A. Fast and robust fixed-point algorithms for independent component analysis[J]. IEEE Trans on Neural Networks, 1999, 10(3): 626-634

[17] Hyvärinen A. The fixed-point algorithm and maximum likelihood estimation for independent component analysis[J]. Neural Processing Letters, 1999, 10(1): 1-5