Study of Multilevel Parallel Algorithm of KPCA for Hyperspectral Images

Rulin Xu(✉), Chang Gao, and Jingfei Jiang

School of Computer, National University of Defense Technology, Changsha, China
xurulin@gmail.com, gaochangjiyi@126.com, jingfeijiang@126.com

Abstract. Hyperspectral remote sensing image data has been widely used in a variety of applications due to its continuous spectrum and high spectral resolution. However, reducing huge dimensions with high data relevance is time-consuming, and parallel processing is required to accelerate this process. In the previous work, the KPCA (Kernel Principal Component Analysis), a nonlinear dimensionality reduction method was studied, and a parallel KPCA algorithm was proposed based on heterogeneous system with a single GPU, and achieved the desired experimental results. However, as data scale grows, the proposed solution would consume all the available memory on a single node and encounter performance bottleneck. Therefore, to tackle the limitation of insufficient memory caused by the reduction of large-scale hyperspectral data dimension, in this paper the intra-node parallelization using multi-core CPUs and many-core GPUs are exploited to improve the parallel hierarchy of distributed-storage KPCA. Finally, we designed and implemented a multilevel hybrid parallel KPCA algorithm that achieves 2.75–9.27 times speedup compared to the traditional coarse-grained parallel KPCA method on MPI.

Keywords: Hyperspectral image · Nonlinear dimensionality reduction
KPCA · GPU · Heterogeneous system · Multilevel parallel

1 Introduction

Hyperspectral remote sensing exploits the imaging spectrometer to obtain many continuous-spectrum features image with nanometer resolution. With the spatial information, radiation information and spectral information integrated, hyperspectral remote sensing is used in a wide array of applications, including geography, biology, agriculture, forestry, marine science, space exploration, anti-terrorism, and military etc [1].

Evolving from traditional two-dimensional images, hyperspectral remote sensing images introduce the spectral dimension to form a three-dimensional hyperspectral data cube. Continuous feature spectrum information can effectively distinguish spectral characteristic of surface material and excavate the hidden information. However, it also has many problems: (1) high resolution

L. Li et al. (Eds.): NCTCS 2018, CCIS 882, pp. 99–115, 2018.
https://doi.org/10.1007/978-981-13-2712-4_8

leads to high correlation between continuous bands and weakens the characteristics of different ground target classifications, thus impairing the classification accuracy; (2) tremendous information redundancy brings great challenges to reduce the time and space complexity of execution. In view of above issues, hyperspectral data dimensionality reduction is proposed, which is crucial part of hyperspectral image analysis.

Dimensionality reduction is a large-scale and computation-intensive task. The traditional serial processing method is time-consuming and computationally complex, which makes it difficult to apply to real problems. With the rapid development of parallel processing, it has become a predominant strategy to design efficient parallel algorithms that make the best of hardware resources to solve many scientific computing problems. The combination of the dimensionality reduction method and the advanced parallel technology not only brings tremendous performance benefits, but also helps to expand business scale without causing additional costs.

The common dimension reduction methods fall into two categories: linear dimensionality reduction and nonlinear dimensionality reduction. In comparison, the former's result is composed of finite parameters, making it convenient and intuitive to explain the data composition. However, there could be some nonlinear data clusters in the hyperspectral image, on which applying linear processing will cause original characteristic loss and result in the hyperspectral data ineffectiveness. The results in [2] prove that the nonlinear dimension reduction technique is appropriate for data with nonlinear features such as hyperspectral image. However, due to the high computational complexity, long elapsed time and insufficient running space, it is difficult to apply nonlinear methods to massive hyperspectral data.

The kernel-based dimension reduction algorithm is an important branch of the nonlinear dimension-descending algorithm family. Based on the Gaussian kernel and Principal Component Analysis, the KPCA (Kernel Principal Component Analysis) algorithm was initially proposed in the kernel-based dimensionality reduction field, where it has solid theoretical basis and significant application prospects. Hence, its parallelization is of great importance in hyperspectral processing field.

In our previous work [3], we proposed a parallel KPCA algorithm (KPCA_G) based on CPU/GPU heterogeneous system with a single GPU, which achieved good parallel performance. However with the further study, we find that as data scale grows, the proposed solution would consume all the available memory on a single node and encounter performance bottleneck. Therefore, in this paper we are focused on intra-node parallelization and performance optimization of KPCA algorithm, and the content is organized as follows: The second section discusses the related work. The third section briefly describes the KPCA algorithm and pinpoints the accelerating hotspot. In the fourth section, our previous work on single GPU is introduced briefly. In the fifth section, inspired by the multi-level cooperative parallel technology and optimization strategy based on CPU/GPU heterogeneous system, we realize the distributed-storage

KPCA algorithm (KPCA_M) and multilevel parallel algorithms (KPCA_M_O and KPCA_M_O_G). These methods provide effective solutions especially for large-scale hyperspectral data nonlinear dimensionality reduction. Section 6 explores the performance improvements of different parallel algorithms through extensive experiments. The last section provides a summary and some outlooks.

2 Related Work

In recent years, parallel computing on CPU/GPU heterogeneous systems has emerged in the field of hyperspectral remote sensing image processing. Bernabe [4] designed a parallel automatic target detection algorithm (ATDCA) on the GPU platform to meet the real-time processing demands. In some domain of hyperspectral applications, such as feature target detection and anomaly identification [5], the parallel design of algorithms on GPU has become key technology and is widely applied. Agathos [6] and Torti [7] employed the GPU and realized a parallel hyperspectral unmixing algorithm, which handles the tremendous complex computing tasks. For processing larger scale of data, Sanchez [8] achieved real-time hyperspectral unmixing using a GPU cluster. On the other hand, Keymeulen [9] and Santos [10] accelerated hyperspectral image lossless and lossy compression algorithm on GPU respectively; ElMaghrbay [11] proposed a fast GPU algorithm for extracting hyperspectral image features.

As for linear dimensionality reduction of hyperspectral image, Fang [12] proposed a parallel principal component analysis (PCA) algorithm, which obtained 128 times speedup on two GPUs. In the same year, Fang [13] implemented a three-level hybrid parallel FastICA algorithm for hyperspectral image dimensionality reduction on a CPU/GPU heterogeneous system. This method gained 159 times performance speedup, in which the computation component got accelerated by 169 times. Wu [14] improved the maximum noise fractional transform (MNF) algorithm and implemented a GPU-based G-OMNF algorithm. As for nonlinear method, Gao [3] gave a parallel KPCA algorithm based on single GPU platform, which achieved up to 173x speedup over the original serial KPCA.

In summary, GPUs have been used to facilitate the speed of various hyperspectral algorithms due to their powerful computing ability and memory bandwidth, and they gradually become a research and development trend in hyperspectral remote sensing. Regarding the hyperspectral dimensionality reduction, parallel research of the linear algorithm has become more sophisticated, while the parallelization of the nonlinear dimensionality reduction algorithm is still in the infancy and exploration stage. To fill in the gaps, this paper proposes new parallel algorithms and provides performance optimization on the heterogeneous system with multiple CPUs and GPUs.

3 KPCA Algorithm and the Analysis of Hot Spot

Through implicit space transformation, KPCA algorithm maps the original data onto an infinite-dimensional Hilbert Space. The results will have linear

properties and the PCA procedure can be applied to extract features hereafter [15]. For the convenience of presentation, we Define the HSI data set as $\boldsymbol{X}=\{\boldsymbol{X}_1,\boldsymbol{X}_2,\cdots\boldsymbol{X}_I\}=\{\boldsymbol{Y}_1,\boldsymbol{Y}_2,\cdots\boldsymbol{Y}_B\}^{\mathrm{T}}$ (in which $I = W \times H$, W and H are spatial dimensions, and B denotes the spectral dimension). Actually, $\boldsymbol{X}$ can be expressed by a three-dimensional data cube extracted from B images and has a size of $W \times H$. After dimensionality reduction, m (m < B) features are extracted from B original spectral bands. The KPCA approach conceptually involves four steps:

(1) Compute the Gaussian kernel matrix $\boldsymbol{K} = Gauss(\boldsymbol{X})$ and get the centering matrix $\boldsymbol{K}_L = Gaussmodify(\boldsymbol{K})$;
(2) Perform matrix eigenvalue decomposition of $\boldsymbol{K}_L$: $\boldsymbol{V}^{-1}\boldsymbol{K}_L\boldsymbol{V} = \boldsymbol{\Lambda}$, where $\boldsymbol{\Lambda}$ ($diag\{\lambda_1, \lambda_2, \cdots, \lambda_I\}$) is a diagonal matrix, $\boldsymbol{V} = \{v_1, v_2 \cdots, v_I\}$ denotes eigenvalue matrix and v_i represents i^{th} eigenvector that satisfies the condition of $\lambda_i (v_i \cdot v_i) = 1$;
(3) Sort eigenvalues $\{\lambda_1, \lambda_2, \cdots, \lambda_I\}$ in descending order and rearrange eigenvectors in accordance with the eigenvalues' new sequence. Select the first m eigenvectors to form $\boldsymbol{V}_m$;
(4) Execute the KPCA mapping according to the function $\boldsymbol{P} = \boldsymbol{K}_L\boldsymbol{V}_m^T$.

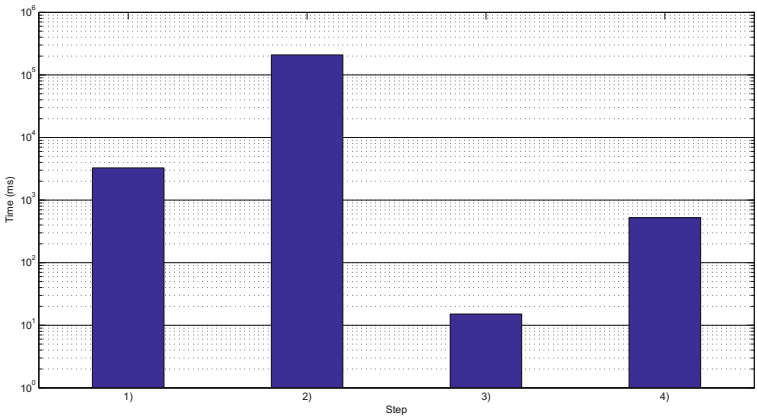

Fig. 1. The elapse time of all steps in serial KPCA.

We have implemented the serial KPCA algorithm according to the above steps, in which the Bilateral Jacobi Iteration is applied to conduct symmetric matrix eigenvalue decomposition. In our implementation, the termination of the Jacobi iteration is when the maximum absolute value is smaller than the setting accuracy (e.g. $ep = 0.001$).

When profiling the algorithm, we cut out a segment with size of 32 * 32 * 224 from the hyperspectral image to excavate its acceleration module. As shown in Fig. 1 (see Fig. 1), Jacobi Iteration, calculating Gaussian kernel matrix and KPCA mapping, which are denoted by step (2), (1) and (4), are the critical bottlenecks.

4 A GPU-Based Parallel KPCA Algorithm

In this section, we briefly present the previous research on parallel design and performance optimization of KPCA algorithm on many-core GPU platform. And here is focused on program design and optimization strategy of Jacobi iteration, which is the main hotspot of KPCA. The details could be found in [3].

4.1 Design and Optimization of Jacobi Iteration on GPU

The Jacobi iteration method reduces the non-diagonal elements of the symmetric matrix KL to zero by plane rotation (Givens transformation). The transformation equation is:

$$(\boldsymbol{Q}_k^T ... (\boldsymbol{Q}_1^T \boldsymbol{K}_{\mathrm{L}} \boldsymbol{Q}_1) ... \boldsymbol{Q}_k) = \boldsymbol{\Lambda} \tag{1}$$

Matrix $\boldsymbol{Q}_i$ represents the Givens transformation matrix. Givens transformation is an orthogonal similarity transformation. The rotation transformation matrix is shown in Eq. 2, in which p and q indicate the number of lines where the parameters located.

$$\boldsymbol{Q}_i(p,q,\theta) = \begin{pmatrix} 1 & & & & & & \\ & \ddots & & & & & \\ & & \cos\theta & & \sin\theta & & \\ & & & \ddots & & & \\ & & -\sin\theta & & \cos\theta & & \\ & & & & & \ddots & \\ & & & & & & 1 \end{pmatrix} \begin{matrix} \\ \\ (p) \\ \\ (q) \\ \\ \\ \end{matrix} \tag{2}$$

Let a_ij be the $i-rowj-column$ element of matrix $\boldsymbol{K}_{L(k)}$ ($\boldsymbol{K}_L$ iteration for k times), bij is the $i-rowj-column$ element of matrix $\boldsymbol{K}_{L(k+1)}$. The Givens transformation can be denoted by $\boldsymbol{K}_{L(k+1)} = \boldsymbol{Q}_{k+1}^T \boldsymbol{K}_{L(k)} \boldsymbol{Q}_{k+1}$. Related data updates are shown in Eq. 3–7:

$$b_{ip} = b_{pi} = a_{pi}\cos\theta - a_{qi}\sin\theta, i \neq p, q \tag{3}$$

$$b_{iq} = b_{qi} = a_{pi}\sin\theta + a_{qi}\cos\theta, i \neq p, q \tag{4}$$

$$b_{pp} = a_{pp}\cos^2\theta + a_{qq}\sin^2\theta - a_{pq}\sin 2\theta \tag{5}$$

$$b_{qq} = a_{pp}\sin^2\theta + a_{qq}\cos^2\theta + a_{pq}\sin 2\theta \tag{6}$$

$$b_{pq} = b_{qp} = a_{pq}\cos 2\theta + \frac{a_{pp} - a_{qq}}{2}\sin 2\theta \tag{7}$$

In order to transform element b_{pq} with the coordinate (p, q) to zero, we set Eq. 7 to zero to get the parameter θ. In Eq. 3–7, only elements in row p, q and column p, q of matrix $\boldsymbol{K}_{L(k)}$ are updated, while other elements are unchanged, which shows the localization of data updates in Givens transformation. In summary, if the corresponding parameters (p, q) in $\boldsymbol{Q}_i$ are not intersect, a number of $I/2$ Givens transformations can be executed in parallel, which reduces $I/2$ non-diagonal elements to zero and updates the entire matrix elements. Figure 2 shows the work-flow of the parallel algorithm on the GPU platform (see Fig. 2).

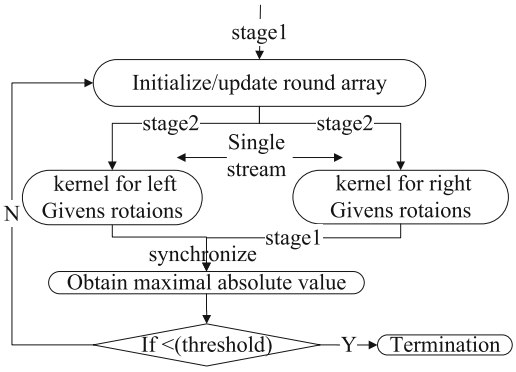

Fig. 2. Bilateral Jacobi iteration parallel algorithm based on GPU.

In the GPU-based parallel framework, step 2 contains the left and right Givens transformation that are executed in a serial manner. In step 3, the element with absolute maximum value is detected after each round of updates, and if it is less than the setting threshold, the Jacobi iteration stops.

4.2 Parallel KPCA on GPU

Finally, we select the optimal optimization of the three bottlenecks to implement GPU-based parallel KPCA algorithm, the model of which is shown here (see Fig. 3).

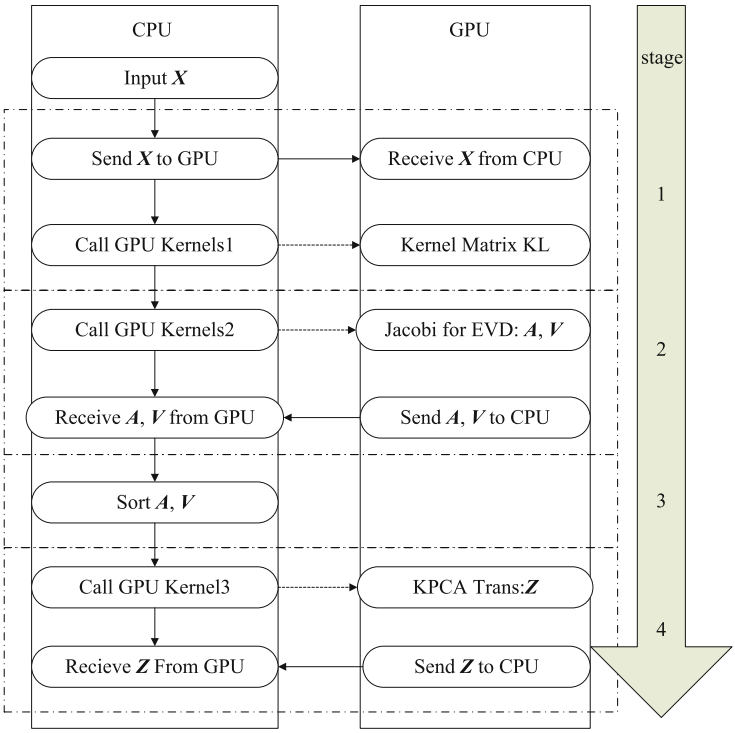

Fig. 3. KPCA algorithm model based on GPU.

5 Parallel Dimensionality Reduction Scheme of Hyperspectral Image Based on Multilevel Parallelism

As data scale grows, the proposed solution would encounter memory bottleneck. In this section, we provide a MPI-based KPCA algorithm based on multi-node systems and exploit parallelization mechanism of intra-node with many-core GPU for better dimensionality reduction performance.

5.1 MPI-Based KPCA Algorithm

(1) MPI-based Gaussian Kernel Matrix calculation

Considering the specialty of hyperspectral data, we divide the data and propose a parallel computing scheme of gauss matrix (see Fig. 4).

In this figure, $\boldsymbol{C}$ is the result matrix. Each processor maintains two buffer (bufA and bufB) to cache the intermediate local data. The computation flow is illustrated in Fig. 4a. P_i reads block_i into bufA and bufB, and then all the processors start calculating the data blocks placed on the diagonal line. After that, the data in bufA remain unchanged but the data in bufB move upwards recursively, which is shown in Fig. 4b.

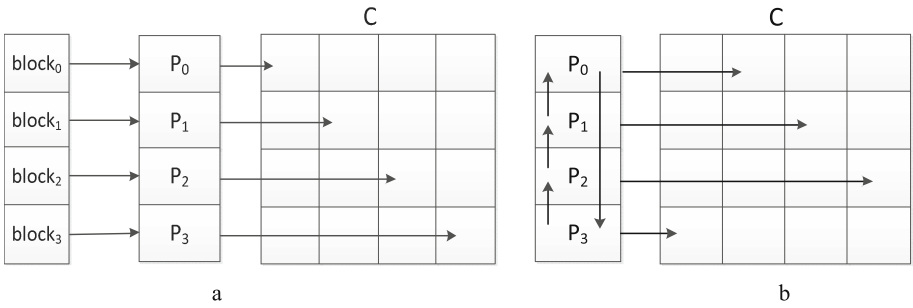

Fig. 4. Parallel program of Gaussian matrix calculation based on the characteristics of hyper-spectral data on GPU.

(2) MPI-based Jacobi iteration

The basic idea of Jacobi iteration is to make any off-diagonal element close to 0 through iterations of Givens transformation, so that the result matrix (denoted as $\boldsymbol{K}_L^*$) is diagonalized. To analyze parallelism, we partition $\boldsymbol{K}_L^*$ into several blocks. The process of transforming off-main-diagonal elements in a data block into 0 is referred to as a sub-task. In each sub-task, data locality is observed. For example, in the process of left-hand Givens transformation, $block(i, j)$ has two input lines i and j, which are independent from each other. However, there are some issues to be addressed if multiple sub-tasks were executed in parallel.

Issue 1: if sub-tasks belonging to different processes are placed on the same row or column, part of the Givens transformation will be applied on the same data block. This part of data may subject to out-of-order updates imposed by different processes, eventually leading to non-reproducible results and data inconsistency.

Issue 2: the bilateral Jacobi iterations in each sub-task involve left transformations (update row block) and right transformations (update column block). If data matrix is partitioned in row, then the column data block to be updated will not be fully preserved on local storage, vice versa.

To solve issue 1, we partition the tasks and data (see Fig. 5) to make sure that the transformation will not cause inter-process interference. Each sub-task therefore has disjoint indices and the data blocks to be updated are independent from each other.

Figure 6 shows the holistic mapping of sub-tasks and data blocks in the Jacob iteration (see Fig. 6). On the left-hand side lies the partitioning of diagonal sub-tasks (the partition granularity is denoted by L). Local diagonal elements in these sub-tasks are global diagonal elements, so they do not need to be transformed into 0. The right-hand sub-figure illustrates how the non-diagonal sub-tasks are assigned to each process (the granularity $\mathrm{R} = \mathrm{L}/2$). The arrow to each process (P_i) represents that all the blocks in the row are preserved locally.

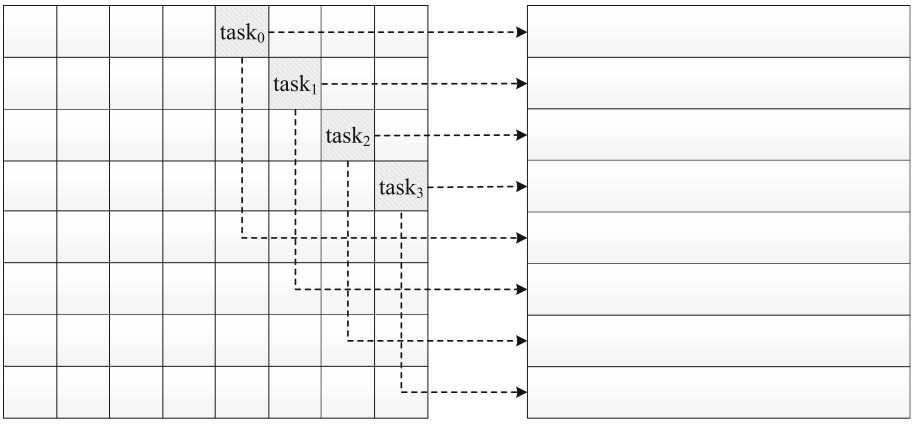

Fig. 5. Task allocation scheme.

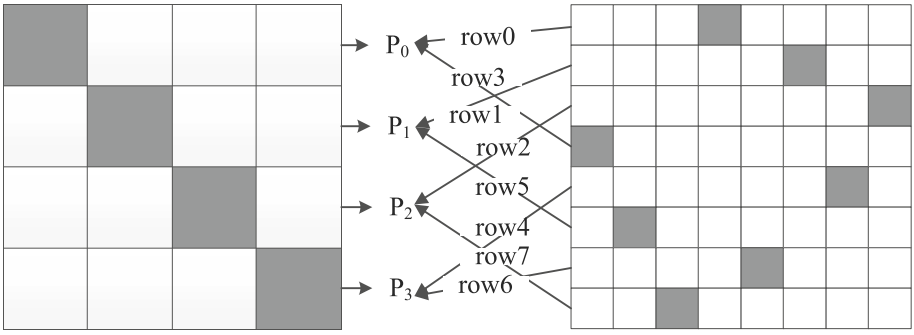

Fig. 6. Global mapping of Jacobi iteration subtasks and related data.

During the right-hand Givens transformation, parts of the data blocks to be updated and parameters of updates are dispersed on different processes. It results in the data dependency among different processes. To tackle the second issue, after updating each row in left-hand Givens rotation, the collection and broadcast of parameters is performed to obtain the global parameter list, so that each process store all the needed data on local storage.

In summary, the parallel KPCA process first partitions data for Gaussian calculation and outputs the Gaussian matrix blocks to the local storage, then it performs the Jacob iteration to integrate principle eigenvectors to form the mapping matrix, and broadcast it to all the processes for them to execute the KPCA mapping independently.

5.2 The KPCA Algorithm on Distributed Storage and GPU

The parallel mechanism of the sub-tasks on diagonal line is similar to the description in Sect. 4. Based on the distributed storage KPCA algorithm, this section studies the fine-grained parallel mechanism of the non-diagonal sub-tasks.

When utilizing multiple threads to perform several Givens transformations concurrently, dependency of the threads would lead to issues such as computing overlaps and out-of-order updates, yielding wrong calculation results. Therefore, before implementing Givens rotations in parallel, we first need to remove relevance of the Givens parameters between different threads. After decorrelation operation, all the p and q are distinctive, so that the operations and data updates of the threads do not affect each other.

Because the line numbers and column numbers of non-diagonal sub-task are different, we can combine each row and column randomly as a return-to-zero element's coordinate, all of the coordinate forming an independent sequence. The max degree of parallelism in non-diagonal sub-task is *blocksize*. For covering all the elements in non-diagonal block, Fig. 7 illustrates how the independent sequence of non-diagonal sub-task is scheduled (see Fig. 7). By enabling parallel computation within the nodes, the independent Givens transforms are processed at the same time. In Fig. 8, we implement the programming model of "MPI + CUDA". The MPI process is responsible for data communication and coarse-grained parallel calculation, while the GPU is focused on high-density floating point operations.

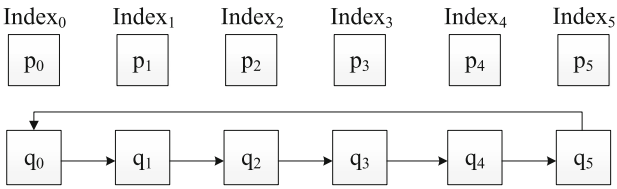

Fig. 7. Scheduling policy of independent sequence.

The execution of the sub-tasks has both independence and correlation, that is, the row-related data blocks are stored locally, and the column-related blocks are scattered at each node. Then independent left-hand Givens transformations launch so that row blocks are updated in parallel, which is referred as a round of circulation (see Fig. 8). Since the Givens-right-transformed data for column update is scattered on different nodes, parameter gather and broadcast are required between the two modules, after which column blocks update begin. The inner loop contains scheduling the independent sequence and repeating the above procedure in a node.

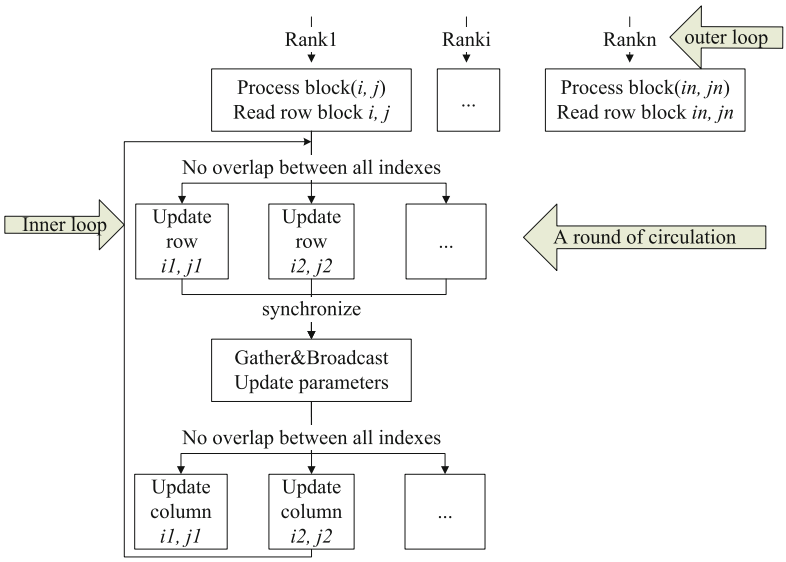

Fig. 8. Jacobi algorithm based on MPI+CUDA.

5.3 KPCA Algorithm Based on Distributed/Shared Storage and GPU

As discussed in Sect. 5.1, the data and tasks are partitioned and dispatched on distributed storage. Section 5.2 has expatiated on how the large-scale computing tasks in Givens transform are handled by GPU. In this section, other suitable parallel tasks, such as Gaussian kernel calculation and Givens parameter calculation, are processed using multi-thread OpenMP, achieving the three-level parallel KPCA algorithm based on MPI + OpenMP + CUDA. The multilevel hybrid parallel KPCA makes full use of different platform resources and is suitable for large-scale data processing with good scalability and portability.

In the MPI implementation of the KPCA algorithm, each process reads its own data and calculates the Gaussian kernel function. The calculation procedure include three circulations, the first two represent data coordinates, while the inner loop carries out the specific Gaussian operations. The operations in outer layer are independent to each other, and therefore can be allocated to different threads to perform parallel execution. In the Givens parameter calculation, the independent sequence is updated, in which the parameters are calculated corresponding to the reducing point in coordinate of (p, q). Without data correlation, the parameter calculation processes are independent and can be executed by multi-threads.

6 Result Analysis

We use the hyperspectral image data provided by AVIRIS in the United States to run the experiments. Table 1 lists the three sets of hyperspectral image data used in the experiment. They are extracted from the original hyperspectral image and the number of pixels is 1024,4096 and 16,384, respectively.

Table 1. Hyperspectral remote sensing image information

ID	Width	Height	Bands
1	32	32	224
2	64	64	224
3	128	128	224

The experimental platform is a heterogeneous cluster, where the CPU/GPU heterogeneous nodes are equipped with two 8-core Intel (R) Xeon (R) CPU E5-2670 and two Kepler architecture NVIDIA Tesla K20c GPUs. The system environment includes Red Hat Enterprise Linux Server release 6.2 (Santiago), GCC-4.4.6 compiler, CUDA release 5.5 toolkit and the MPICH library compiler. The spectral information is transformed into pixel information, the original 16-bit integer data is converted into un-signed char type (unsigned char).

6.1 GPU-Based KPCA Algorithm

The KPCA algorithm is executed on a single node using different processor configurations, including single-core (CPU), multi-core (CPU) and many-core (GPU). Table 2 reports the execution time and the speedup of the three KPCA implementations. These include KPCA_S (the serial KPCA), KPCA_O (a multi-thread parallel implementation on OpenMP using 16 processors) and KPCA_G which is based on the many-core GPU architecture. The experimental result is listed in Table 2. The more details could be found in [3].

Table 2. Time (ms) and speedup of KPCA

	Data1		Data2	
	Time	Speedup	Time	Speedup
KPCA_S	82,017.55	-	15,710,746.06	-
KPCA_O	18,519.99	4.43	6,300,228.22	2.49
KPCA_G	1,817.25	45.13	90,778.75	173.07

The results show that KPCA_O and KPCA_G achieve 2.49–4.43 and 45.13–173.07 times performance improvement compared to KPCA_S, and the acceleration ratio of the GPU-based implementation increases along with the volume

of the test data, which proves that it is more scalable than the multi-thread implementation.

The experimental platform used in Sect. 6.1 is a single CPU/GPU heterogeneous node. Due to the limited running space, only the first two sets of data can be calculated. The subsequent multi-node parallel experiment will test the large-scale data and provide further discussion.

6.2 Analysis of the MPI-Based KPCA

In the Jacobian iteration that is based on distributed storage, obtaining the global maximum absolute value requires frequent data exchange between nodes, which results in huge performance loss in production-scale data processing and gives uncontrollable results. In this paper, we simplify the end-point discrimination process and set the terminating condition of the Jacobi iteration as the fixed number of cycles, i.e., all non-diagonal elements are reduced to zero once. Table 3 shows the execution time of the algorithm.

Table 3. Time (s) of serial KPCA and main modules

Data	1	2	3
KPCA	16.30	1,781.53	170,553.54

Our experiments also evaluate the KPCA algorithm on MPI, where n in KPCA_M (n) is the number of launched processes. We use three sets of data to test the experiment, and record the execution time of the slowest process. Table 4 records the execution time of overall KPCA_M procedure. We also compare KPCA and KPCA_M algorithms and show the acceleration effect of KPCA_M (see Fig. 9). The results show that, the speed ratio increases along with the growing number of processes. However, when the volume of data increases, the performance of KPCA rises and then falls. Among all the test cases, the KPCA_M using 8 processes has the best result, obtaining 7.78–18.24 times performance improvement.

Table 4. Time (s) of KPCA_M and main modules

Data	KPCA_M(2)	KPCA_M(4)	KPCA_M(8)
1	4.77	2.30	1.14
2	438.71	188.15	97.65
3	40, 228.89	28, 506.95	21, 918.38

As the process number grows, the partition granularity of KPCA_M decreases, so the performance gains. When the data scale is large, the overall performance has witnessed a significant degradation, and the performance

of these three types of KPCA_M is not comparable to the previous test cases using smaller size of data sets. Our analysis shows that the main factor is the MPI group communication adopts the "many times, smaller amount" style of transmission, so with the increase of the data scale, the transmission time also increases, and results in poor performance. Therefore, when the data size is large, it may be appropriate to increase the number of nodes to reduce the amount of communication data and avoid performance degradation.

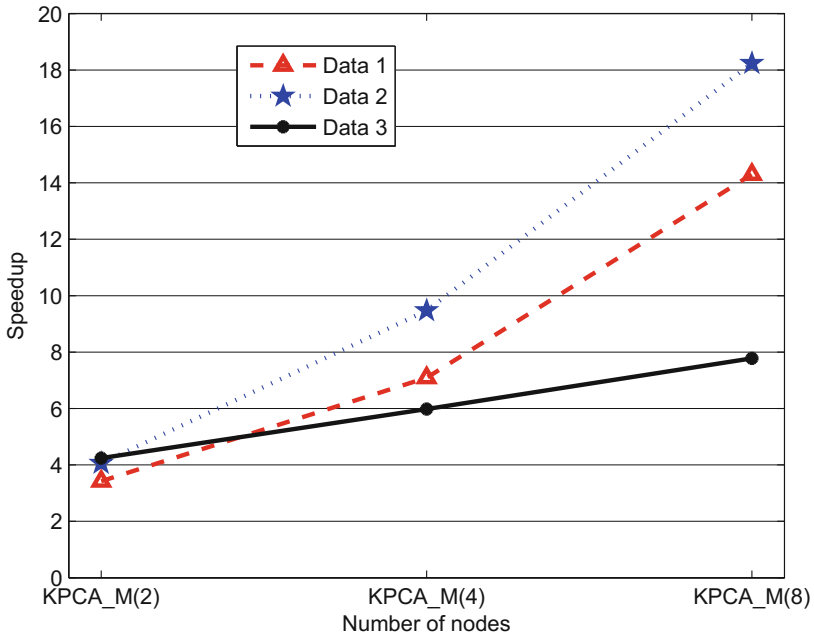

Fig. 9. Speedup of KPCA_M algorithm.

6.3 Analysis of KPCA Algorithm Based on MPI + CUDA

In the experiment of parallel KPCA algorithm based on MPI + CUDA, two processes are launched, each of which is equipped with a GPU.

By recording the execution time of the KPCA_M_G and KPCA_M algorithm, we calculates the performance improvement comparing the KPCA_M_G (default 2 processes) with KPCA_M algorithm (see Fig. 10): the introduction of GPU parallelization in the nodes has enabled the KPCA_M_G to be faster 2.56–9.03 times than the original KPCA_M(2), and KPCA_M_G is faster than the extended 4-process KPCA_M by 1.24–6.4 times. In addition, when using the smallest data set, performance of KPCA_M_G is poorer than the 8-process extended KPCA_M. When the last two sets of data are used, its performance improves by 1.25 and 4.92 times than 8-process extended KPCA_M. The experimental results show

that with the increase of the data volume, the proportion of the calculation in the node increases, so that the advantage of utilizing GPU gradually appears to make it suitable for the fine granularity parallelism in the node.

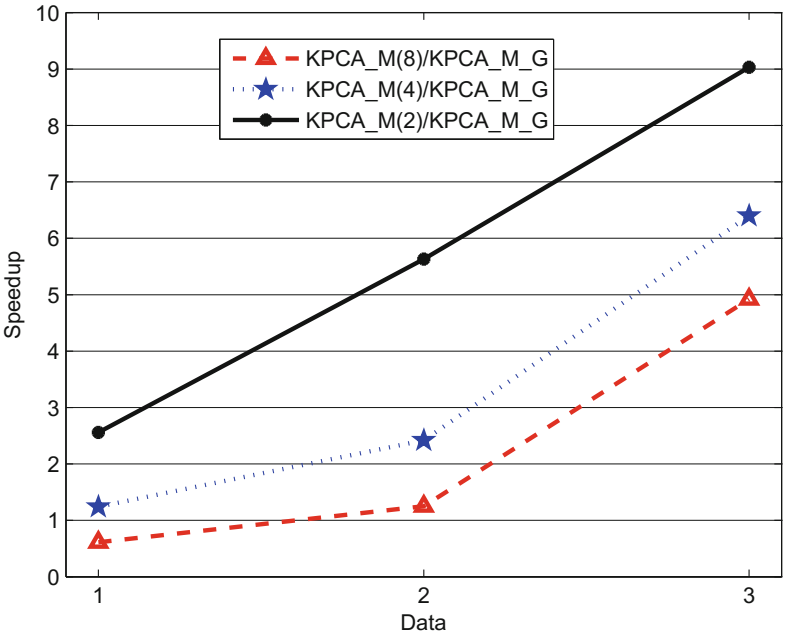

Fig. 10. Performance comparison between KPCA_M and KPCA_M_G.

6.4 Analysis of KPCA Algorithm Based on MPI + OpenMP + CUDA

This section implements the three-level parallel KPCA algorithm based on MPI + OpenMP + CUDA. Table 5 records the execution time of various parallel KPCA algorithms. KPCA_M_O_G uses two CPU + GPU heterogeneous nodes and obtains the acceleration ratio of 2.75–9.27 compared with 2-process KPCA_M, as well as the acceleration ratio of 1.33–6.57 and 1.39–5.05 compared with 4-process and 8-process KPCA_M algorithm. By comparing with the KPCA_M algorithm extending the number of nodes, we can effectively reduce the number of nodes and reduce the cluster size under the same performance requirements by KPCA_M_O_G.

When the minimum data set (Data 1) is used for the KPCA_M_O_G experiment, compared with KPCA_M(8), the performance does not rise but fall, mainly because the use of GPU optimization algorithm will introduce some additional over-head, including GPU warming up, kernel boot, data transmission between host and device. Based on the experimental results, we can draw the following conclusions:

Table 5. Time (s) of KPCA in different ways

Data	KPCA_M_O_G	KPCA_M_G	KPCA_M(2)
1	1.73	1.86	4.77
2	70.22	77.90	438.71
3	4,340.47	4,457.39	40,228.89

(1) When the size of the original hyperspectral data is suitable for a single machine, the KPCA_G algorithm can greatly improve the efficiency of data reduction;
(2) When the data size increases, and the memory requirement for processing is beyond the limit of a single-node, the distributed and parallel heterogeneous cluster platform is an inevitable choice. This paper presents the MPI + OpenMP + CUDA, a multilevel hybrid parallel algorithm and provides a good application demo.

7 Summary and Outlook

Taking KPCA algorithm for example, this paper reviews KPCA_G algorithm on CUDA, and designs KPCA_M_G algorithm on MPI + CUDA and KPCA_G_O_G algorithm on MPI + OpenMP + CUDA. The experiments prove that all these algorithms achieve remarkable performance improvements on the CPU/GPU heterogeneous system.

The research of GPU-based parallel algorithms for nonlinear dimension reduction of hyperspectral images is still on preliminary stage, there are lots of open research questions left to be answered, we put forward the following ideas for the future work: (1) it's necessary to develop a specialized parallel function library for hyperspectral dimensionality reduction, which can unify standard, simplify procedure, and promote application of hyperspectral dimensionality reduction algorithm. (2) With the rapid development of high performance computing technology, optimization of algorithm on different high performance computing system will be a long-lasting topic.

References

1. Zhang, Z., Zhang, L.: Hyperspectral Remote Sensing. Wuhan University Press, Wuhan (2005). (in Chinese)
2. Ainsworth, T.L., Bachmann, C.M., Fusina, R.A.: Local intrinsic dimensionality of hyper-spectral imagery from non-linear manifold coordinate. In: IEEE International on Geoscience and Remote Sensing Symposium, pp. 1541–1542 (2007)
3. Gao, C., Zhou, H., Fang, M.: Parallel Algorithm and Performance Optimization of Kernel Principal Component Analysis on GPUs for Dimensionality Reduction of HIS, HPC China, pp. 611–614 (2016)
4. Bernabe, S., Lopez, S., Plaza, A., Sarmiento, R.: GPU implementation of an automatic target detection and classification algorithm for hyperspectral image analysis. IEEE Geosci. Remote Sens. Lett. **10**(2), 221–225 (2013)

5. Lokman, G., Yilmaz, G.: Anomaly detection and target recognition with hyperspectral images. In: 2014 22nd Signal Processing and Communications Applications Conference (SIU), pp. 1019–1022, 23–25 2014
6. Agathos, A., Li, J., Petcu, D., Plaza, A.: Multi-GPU implementation of the minimum volume simplex analysis algorithm for hyperspectral unmixing. IEEE J. Sel. Topics Appl. Earth Obs. Remote Sens. **7**(6), 2281–2296 (2014)
7. Torti, E., Danese, G., Leporati, F., Plaza, A.: A hybrid CPU-GPU real-time hyperspectral unmixing chain. IEEE J. Sel. Topics Appl. Earth Obs. Remote Sens. **9**(2), 945–951 (2016)
8. Sanchez, S., Ramalho, R., Sousa, L., Plaza, A.: Real-time implementation of remotely sensed hyperspectral image unmixing on GPUs. J. Real-Time Image Process. **10**, 469–483 (2012)
9. Keymeulen, D., Aranki, N., Hopson, B., Kiely, A., Klimesh, M., Benkrid, K.: GPU lossless hyperspectral data compression for space applications. In: 2012 IEEE Aerospace Conference, pp. 1–9, 3–10 March 2012
10. Santos, L., Magli, E., Vitulli, R., Lopez, J.F., Sarmiento, R.: Highly-parallel GPU architecture for lossy hyperspectral image compression. IEEE Sel. Topics Appl. Earth Obs. Remote Sens. **6**(2), 670–681 (2013)
11. ElMaghrbay, M., Ammar, R., Rajasekaran, S.: Fast GPU algorithms for endmember extraction from hyperspectral images. In: 2012 IEEE Symposium Computers and Communications (ISCC), pp. 000631–000636, 1–4 July 2012
12. Fang, M., Zhou, H., Shen, X.: Multilevel parallel algorithm of PCA dimensionality reduction for hyperspectral image on GPU. Dongbei Daxue Xuebao/J. Northeastern Univ. **35**(S1), 238–243 (2014). (in Chinese)
13. Fang, M., Zhou, H., Zhang, W., Shen, X.: A parallel algorithm of FastICA dimensionality reduction for hyperspectral image on GPU. Dongbei Daxue Xuebao/J. Northeastern Univ. **37**(4), 65–70 (2015). (in Chinese)
14. Wu, Y., Gao, L., Zhang, B., Zhao, H., Li, J.: Real-time implementation of optimized maximum noise fraction transform for feature extraction of hyperspectral images. J. Appl. Remote Sens. **8**(1), 084797 (2014)
15. Scholkopf, B., Smola, A.J., Muller, K.: Nonlinear component analysis as a kernel eigenvalue problem. Neutral Comput. **1**, 1299–1319 (1998)