A Hybrid MapReduce Implementation of PCA on Tianhe-2

Wei Yu[1] Yili Qu[2] and Yutong Lu[1,2,3,a]

[1]College of Computer, National University of Defense Technology, Changsha 410073, China
[2]School of Data and Computer Science, Sun Yat-Sen University, Guangzhou 510006, China
[3]National Supercomputer Center in Guangzhou, Guangzhou 510006, China

Abstract. "Big Data" has been a popular word anywhere. Researchers want the data processing more efficient. PCA algorithm is an effective data reduction algorithm applied to almost all big data fields. Meanwhile, there are many Machine Learning Algorithm Library applied to provide commonly-used algorithm, but these algorithms do not make good use of the resources of the supercomputer system. This paper uses MapReduce Model to design and implement PCA algorithm using MPI+OpenMP+SIMD hybrid accelerator programming tools on Tianhe-2 and get a significant speedup.

1 Introduction

With the development of computer and information technology, the explosive growth of data information has promoted the rapid development of big data, which arouses the attention of academia, industry and governments. How to process and analyse huge data and obtain valuable information from it becomes a challenge.

Increasing computing demand drives the development of supercomputing systems, while the growing demand for big data processing drives the development of big data processing technologies. However, the traditional high-performance computing field and the big data analysis field have a certain degree of separation. The traditional high-performance computing field is mainly aimed for "computation-intensive" applications such as numerical simulation, while the big data analysis field aims to process "data-intensive" applications such as massive data.

Due to the different architecture and design goals, there are also huge differences in design concepts. Big data analysis field provides users a wealth of data processing and analysis tools, such as Hadoop MapReduce [1, 2, 3] and Spark [4, 5] based on MapReduce model [1]. The tool itself also provides some commonly-used algorithm library (such as Hadoop Mahout [6], Spark MLlib [7]). The field of high performance computing is usually based on MPI [8], OpenMP [9] and other accelerator programming tools. Some common operations are available to the users in the form of libraries (such as MKL [10], LAPACK [11]). In addition, the big data analysis field mainly uses Java and Scala language for application development, while the high-performance computing field usually uses higher performance programming languages such as C/C++ and Fortran.

Big data analysis tools are usually based on models such as MapReduce, which hide the implementation details from the application, so that the application development efficiency can be greatly improved and the usage threshold can be lowered. Parallel models such as MPI and OpenMP are models that can greatly improve the efficiency of program operation. In the case that supercomputing systems are widely used in scientific computing and the data scale is expanding, both the ideas from big data analysis and the powerful computing resources from supercomputer systems can be merged together, to design more efficient parallel models to achieve more efficient data processing technology.

In the study of empirical problems, in order to analyse problems comprehensively and systematically, researchers must consider many influencing factors. These factors are called variables in statistics. Each variable reflects some information of the research problem to varying degrees, and there is some correlation between them. Researchers hope that when doing quantitative analysis, the variables involved can be as small as possible, and the amount of information obtained can be the most. Principal Component Analysis (PCA) [12] is the ideal tool to adapt to this requirement and solve such problems. Principal component analysis algorithm was proposed by Karl Pearson in 1901 [13]. It was independently developed and named by Harold Hotelling in the 1930s [14]. It is a stable and effective data dimensionality reduction algorithm applied in almost all fields, often applied to pre-processing before data analysis to reduce data dimensions and reduce redundant information. In the case of ever-increasing data size, pre-processing data using the PCA algorithm can compress and simplify data and save memory while minimizing data loss. However, in practical applications, the data size is too large, exceeding the capacity of the computer memory, and the sample data cannot be completely read into the memory. How to make the PCA

[a] Corresponding author: ytlu@nudt.edu.cn

algorithm as efficient as possible when dealing with large-scale data is a problem worth studying.

This paper optimizes the PCA algorithm on the Tianhe-2 platform. The main contributions are as follows:

- Design a PCA parallel processing algorithm based on MapReduce Model, to make it suitable for application on Tianhe-2.
- Implement the algorithm on Tianhe-2 using OpenMP, MPI and hybrid MPI+OpenMP+SIMD accelerator programming tools separately and compare their performance.

2 Related Works

Carlos Ordonez [15] and others focus on how to calculate PCA in parallel. They extend the PCA serial algorithm to a highly parallel algorithm in a multi-core architecture. The algorithm reduces the big data set to a summary matrix to calculate the PCA at a time. They use the MKL and LAPACK libraries to implement efficient calculations of memory-based SVD by combining user-defined aggregations with parallel dataset summaries. Their work also includes integrating algorithms with DBMS. The algorithm theoretically achieves linear acceleration and linear scalability of the data set. Compared with the software statistics package of R language, it has a great performance improvement.

Austin R. Benson [16] et al. propose an optimized implementation of the PCA algorithm in a MapReduce cluster environment. They implement the QR decomposition iteration and implement it on the ICME Hadoop cluster, which achieves very good results. At the same time, they propose a measurement model to measure the performance of the algorithm, and the experiment proved that their optimization effect is in line with theoretical expectations.

3 Algorithm Design

PCA algorithm, as its name implies, removes redundant information and extracts the main features of the data to achieve the purpose of compressing the simplified data and removing the data noise on the premise of minimum loss information. It is a statistical method, the core idea is to map N-dimensional features to K-dimensional (N>K). The specific mathematical calculation theory is not deduced here. This paper use a fast PCA calculation method [16] directly.

3.1. Algorithm Flow

Suppose there are N samples to be processed, and each sample has M elements. Now K (K<M, N) elements are needed to represent the information of the original N samples. The elements that need to be removed do not contain sample information as much as possible. Each sample is represented as an M-dimensional vector, M samples form N vectors, and the vectors are stored in columns, thus forming a matrix of M*N. The processing of the matrix is as follows:

1. Data decentralization. That is, each data is subtracted from the average of each column, and the geometric meaning is to translate all the data points so that the data center falls on the coordinate origin.
2. Solve the covariance matrix.
3. Solve the eigenvalues and eigenvectors. Since the most real world matrices are not square matrices, SVD decomposition algorithm is used to solve eigenvalues and eigenvectors.
4. Construct the projection matrix using the feature vector.
5. Obtain the data of the dimensionality reduction using the projection matrix.

3.2 Parallel Optimization

In most calculations, the input contains a map operation applied to the data in order to compute a set of intermediate <key, value> pairs. And then all the data that sharing the same key will be performed a reduce operation in order to properly combine the derivations data. Combining the advantages of big data technology and high performance computing, this section proposes a parallel optimization based on the MapReduce model and can make full use of the computing resource environment of Tianhe-2.

In the MapReduce model, data is defined as a collection of <key, value> pairs. Here, key is defined as the row of the input matrix, and the value is all the elements of the row. At this time, the Matrix A_{m*n} can be represented as a collection of <key, value> pairs:

$$A_{m*n} = \{< 1, A\_1 >, < 2, A\_2 >, < 3, A\_3 >, \dots, < m, A\_m > \} \quad (1)$$

Obviously, in this collection, the key in each <key, value> is unique.

The process of solving R using the Cholesky QR method can be expressed intuitively using the MapReduce model, as the Figure 1 shows:

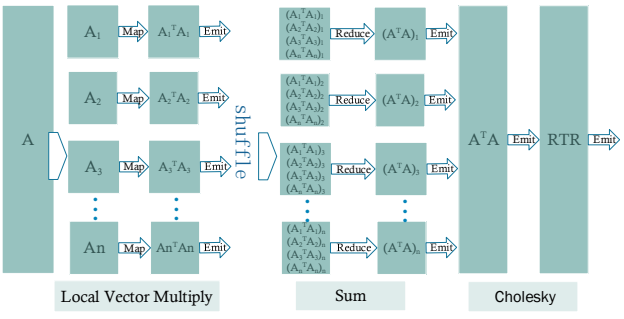

Figure 1. Cholesky QR Method Flow

In the map phase, each task collects the key value, that is, the row value of the matrix, forms a local matrix A_i. Then calculates $A_i^T A_i$. Since $A_i^T A_i$ is symmetrical, we only need to calculate half of its value to get the other half. In the reduce phase, each independent reduce function gets multiple instances of the $A^T A$ row from the map function. These instances combine to get the value of $A^T A$:

$$A^T A = \sum_{i=1}^{map\ tasks} A_i^T A_i \quad (2)$$

After getting R, the right singular vector obtained by SVD decomposition of R is the feature vector needed.

Since R is a small-scale matrix of n*n, we can easily derive the value of the right singular vector V^T, and the new matrix N after dimensionality reduction is:

$$N = AV^T \quad (3)$$

The MapReduce process for solving N is shown in Figure 2:

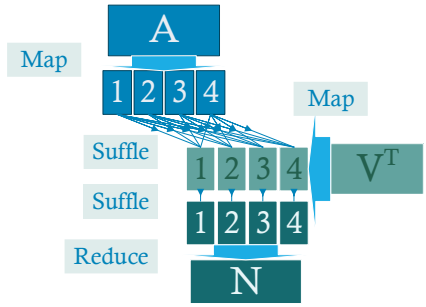

Figure 2. Flow Chart for solving N

The algorithm pseudo code for the entire calculation process is as follows:

ALGORITHM 1: PCA Algorithm

INPUT: MATRIX $\mathbf{A}_{mn}$(m>n)

OUTPUT: MATRIX $\mathbf{N}_{nn}$

```
for i←0 to m do
  for j← 0 to n do
    mean ← MEAN(A_i)
    if mean = 0
      break
    else
    A_ij ← A_ij − MEAN(A_i)
    end if
  end for
end for
get R after MapRedue(Cholesky QR)
get V after SVD(R)
get N after MapReduce(A*V)
```

4 Implementation and Test

The algorithm is implemented on Tianhe-2 using OpenMP + SIMD, MPI and MPI + OpenMP + SIMD, respectively. Since there are already many mature math libraries that efficiently implement matrix operations, this paper no longer implement matrix operations. Eigen is chosen here to implement matrix operations, which is a widely-used C++ template library for linear algebra such as matrices, vectors, numerical solvers, and related algorithms.

4.1 Implementation

For OpenMP + SIMD implementation, we use OpenMP to implement MapReduce Model among processors and SIMD for further optimization. For MPI implementation, we use MPI to implement task distribution among computing nodes. For MPI + OpenMP + SIMD hybrid implementation, we use MPI to implement task distribution among computing nodes, use OpenMP to implement task distribution among processors and use SIMD for further optimization.

4.2 Test

The data set of the algorithm test uses a random generation floating point matrix of the specified scale, and the data are distributed in the range of (-1, 1).

4.2.1 OpenMP + SIMD

For OpenMP + SIMD implementation, the test is on the single computing node. Each computing node of Tianhe-2 is equipped with two Intel Ivy Bridge multi-core CPUs - Intel Xeon E5-2692 v2, with 12 processors integrated on each CPU and a data width of 64 bits. All cores are tightly interconnected via a high speed bus. Each processor can expand a virtual processor, so it can achieve parallel operation of up to 24 threads.

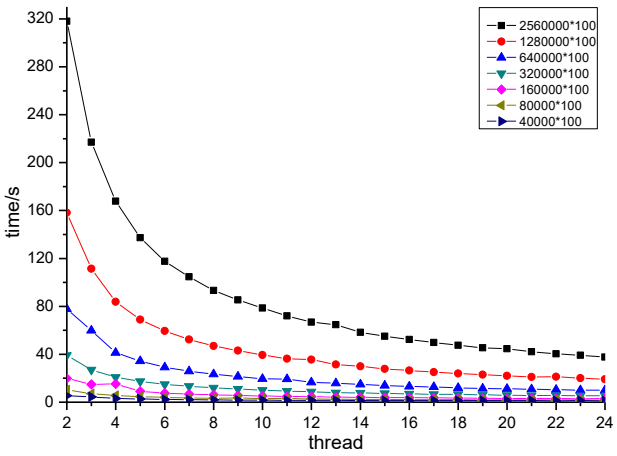

Figure 3. Strong Scalability of OpenMP + SIMD Implementation

In this test, the problem scale is fixed separately in a matrix of 2560000*100 (data size 2.3G), 1280000*100 (data size 1.2G), 640000*100 (data size 575M), 320000*100 (data size 288M), 160000*100 (data size 144M), 80000*100 (data size 72M) and 40,000*100 (data size 36M), to test the strong scalability when the number of threads changes from 2 to 24, the results shows in Figure 3.

4.2.2 MPI

In this test, the problem scale is fixed separately in a matrix of 2560000*100 (data size 2.3G), 1280000*100

(data size 1.2G), 640000*100 (data size 575M), 320000*100 (data size 288M), 160000*100 (data size 144M), 80000*100 (data size 72M) and 40,000*100 (data size 36M), to test the strong scalability when the number of computing nodes is 2, 4, 8, 16, 32, 64, 128, 256, respectively. The results shows in Figure 4:

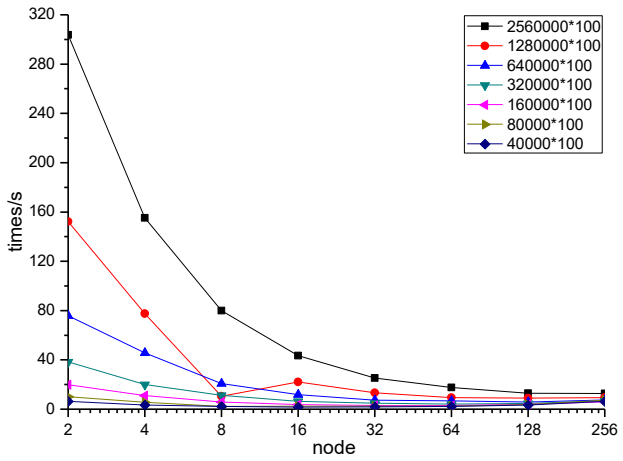

Figure 4. Strong Scalability of MPI Implementation

4.2.3 Hybrid MPI + OpenMP + SIMD

For Hybrid MPI + OpenMP + SIMD implementation, the test records the best performance (minimum time overhead) in every test case in the MPI implementation test and summarized in the following table 1. The NODE = 1 line represents the time overhead of serial implementations using the same algorithm, this shows that the algorithm this paper proposed can reach 203X speedup under the tested conditions.

Table 1(a): Time Overhead of Every Test Case

	4	8	16	32
1	29.337	86.213	115.198	234.77
2	1.675	2.45	3.782	6.362
4	1.345	1.737	2.658	4.42
8	1.247	1.506	2.139	3.251
16	1.308	1.413	1.95	2.654
32	1.297	1.504	1.906	2.474
64	1.606	1.671	1.957	2.609
128	2.609	2.726	2.687	3.378
256	3.772	3.742	3.792	4.458

Table 1(b): Time Overhead of Every Test Case

	64	128	256
1	464.634	937.672	1849.786
2	11.579	24.721	47.568
4	7.629	12.835	29.004
8	5.16	9.604	16.567
16	3.956	7.129	13.428
32	3.938	5.831	10.739
64	3.973	5.869	10.477
128	4.364	5.918	9.125
256	5.547	6.647	10.065

5 conclusion

This paper designs and implements an efficient algorithm to apply PCA on Tianhe-2 using MPI, OpenMP and SIMD hybrid accelerator programming tools and get a speedup of up to 203X. In the future, we will also complete the algorithm implementation on the GPU platform.

6 Acknowledgements

Thanks for the support of National Key R&D Program of China： 2018YFB0203904, NSFC： U1611261，61433019，61872392 and the Program for Guangdong Introducing Innovative and Enterpreneurial Teams under Grant NO. 2016ZT06D211.

References

1. Dean J, Ghemawat S. MapReduce: simplified data processing on large clusters [M]. ACM, 2008.
2. Junqueira B F, Reed B. Hadoop: The Definitive Guide [J]. Journal of Computing in Higher Education, 2001, 12(2):94-97.
3. Apache Hadoop. http://hadoop.apache.org/.
4. Apache Spark. https://spark.apache.org/.
5. Zaharia M, Chowdhury M, Franklin M J, et al. Spark: cluster computing with working sets [C]// Usenix Conference on Hot Topics in Cloud Computing. USENIX Association, 2010:10-10.
6. Apache Mahout. http://mahout.apache.org/.
7. MLlib. https://spark.apache.org/mllib/.
8. MPI: The Message Passing Interface standard. https://www.mcs.anl.gov/research/ projects/mpi/.
9. OpenMP. https://www.openmp.org/.
10. MKL. https://software.intel.com/en-us/mkl.
11. LAPACK. http://www.netlib.org/lapack/.
12. PCA: Principal Component Analysis. https://en.wikipedia.org/wiki/Principal_component_analysis.
13. Pearson, K. "On Lines and Planes of Closest Fit to Systems of Points in Space" [J]. Philosophical Magazine, 1901, 2 (11): 559–572.
14. Hotelling, H. Analysis of a complex of statistical variables into principal components. Journal of Educational Psychology [J], 1933, 24, 417–441, and 498–520.
15. Ordonez C, Mohanam N, Garcia-Alvarado C. PCA for large data sets with parallel data summarization [J]. Distributed & Parallel Databases, 2014, 32(3):377-403.
16. Benson A R, Gleich D F, Demmel J. Direct QR factorizations for tall-and-skinny matrices in MapReduce architectures[C]// IEEE International Conference on Big Data. IEEE, 2013:264-272.