Preliminary Contest Proposal

Ocean University of China

March 2, 2016

Index

1 Background Description

1.1 Hardware and Software Platforms

In order to meet the growing demand for computing, our school has introduced our own computing cluster in recent year. At present, our school has a super computing platform which consists of fifteen sets of blade computing nodes, two GPU nodes, two MIC nodes and a management node. The calculation ability of the entire computing platform can reach 13.4704TFLOP/s, equipped with 96TB memory array. The communication of computing platform based on IB network, which can provide a platform with the exchange capacity of 4.032Tbps. So, The platform can meet the need of college supercomputer task.

1.2 Related Study

Over the past year, we have spontaneously organized and set up an interest group to study and discuss the related knowledge, such as related hardware, GRID, MIC, MPI and so on, and we have carried on detailed record for the progress of each group learning in order to review conveniently. Team members and teachers will also listen to some lectures and participate in related training to improve our ability and to expand the range of knowledge.

Figure 1.1: activities of our interest group

It is worth mentioned that some members in our team took part in ASC15. It is a pity that we missed out on the finals. Fortunately, we have accumulated a lot of experience which is actually beneficial to the contest for our team this year. we will go all out and try our best to get a higher level in ASC16.

Figure 1.2: our team in ASC15

At the same time, the team members have participated in The third student RDMA Programming competition organized by the High Performance Computing Advisory Council and won the third prize. This is the first time for our team to participated in the competition. During the competition, the team members have overcome all kinds of challenges and difficulties and achieved great success through stern executive power and mutual efforts. And the report we submitted has been fully affirmed by the organizing committee. As is known to us all, RDMA is the basis of Infiniband, and Infiniband is the basis of parallel computing. Participating in the RDMA programming competition and having won the third prize is not only the affirmation of our ability, but also the encouragement of our group learning in this period of time.

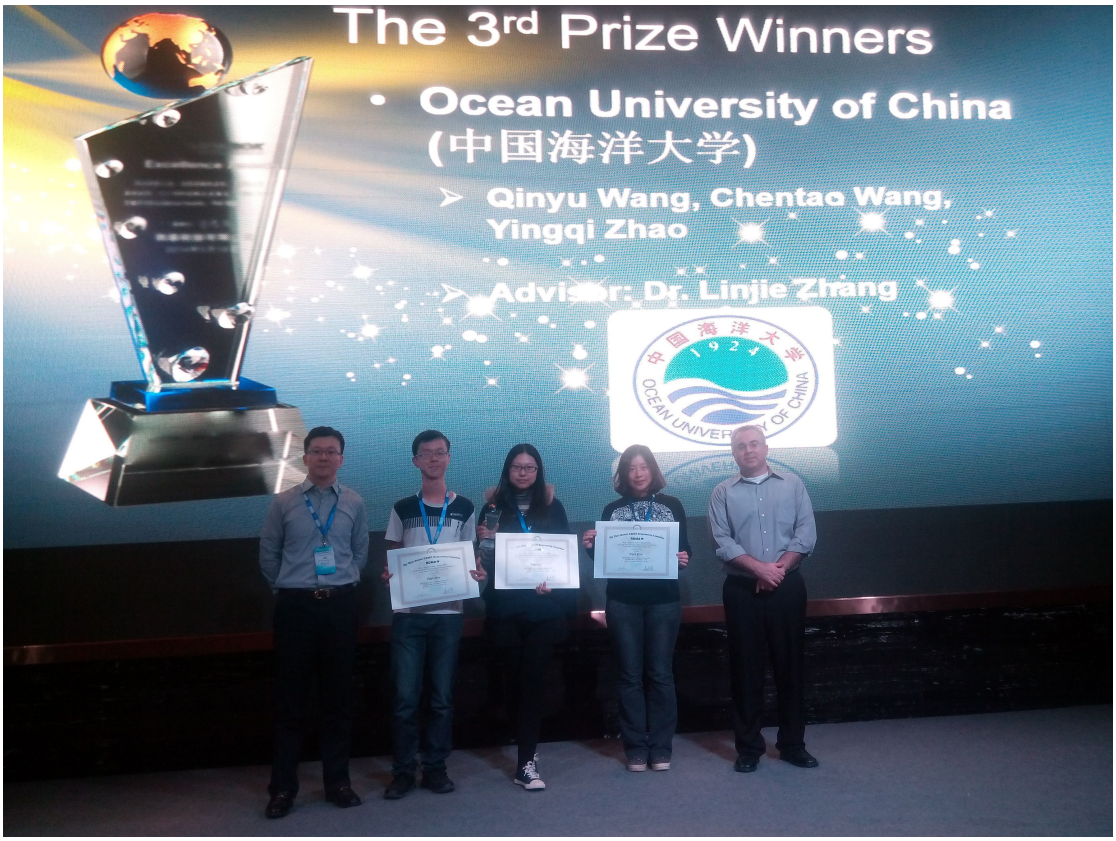


Figure 1.3: The award ceremony of RDMA Programming Competition

1.3 Related Research

With the rapid development of computer and calculation method, almost all subjects are oriented towards quantitative and accurate, which lead to the expansion of the scale of the problem. Therefore, we have to rely on high-performance computing technology. Related research and application can be divided into two aspects:

(1) Numerical solution of partial differential equations. This aspect give a detailed description of a variety of natural and physical phenomena of partial differential equation boundary value problem in science and engineering. After being discretized by the finite element method and the central difference method, the problem has been transformed into a new problem which is based on a large sparse matrix linear. With the requirement of accuracy becoming higher and higher, the scale of the equation is larger. And it is common that the scale of the matrix is over a million. In order to solve such a large scale problem in an acceptable time, it is necessary to take advantage of the high performance computing technology based on parallel computing. At present, we already have five terms focusing on this relevant research, which account for 60% of the total number of teachers.

(2) Remote sensing data processing. Remote sensing technology is an important means of ocean monitoring. In recent years, with the large demand for quantitative and accurate, remote sensing discipline ushered in the era of remote sensing data. Despite the rapid development of computer performance in recent years, it still cannot meet the needs of large data computing and data

processing capabilities. Therefore, it is of vital significance for us to apply high performance computing in remote sensing data processing. And its application is mainly reflected in the following two parts:

a. Quantitative analysis, target detection and recognition based on remote sensing data: With the development of remote sensing technology, the number of remote sensing data is greatly increased, which makes the length of time of the traditional algorithm too long. And it is not conducive to timely information processing.

b. Applying high performance computing to simulation of remote sensing data, we can obtain a large number of remote sensing data in different parametric condition. And it can provide a plenty of simulation test data for target detection and identification. At the same time , it can establish a large scale sample library for the recognition of objects. Nowadays, we already have three teams focusing on related research, accounting for 40% of the total number of teachers.

1.4 Key Achievements

a. **Research:** Double parameter Constant False Alarm Rate (CFAR) is a common algorithm used for ship target detection. In recent years, the resolution of Synthetic Aperture Radar (SAR) is improving continuously and the width of SAR image increases. We all hope to keep the ship profile for subsequent ship target recognition during the detection. However, although this algorithm can meet the requirements of target detection, it costs too much time, which is not conducive to timely information processing. When assigning a test task to each process, the traditional MPI (Message Passing Interface) parallel solution does not consider the problem that the distribution of the detected points in the image is uneven caused by pre-treatment, such as land mask, geometric correction and so on.

   Aiming at this problem, we propose an improved MPI parallel solution. Compared with the traditional MPI parallel solution method, this solution can balance the detection task of each process. The experimental results on the cluster computer show that the parallel standard efficiency is improved by about 43% after being improved. In order to meet the demands of real-time ship target detection for airborne SAR, we experience in multi-core PC and the results show that the algorithm mentioned in this paper can effectively shorten the detection time on multi-core PC, which has a positive impact on the realization of real-time ship target detection of airborne SAR.

   The paper will be published in Journal of remote sensing, 2016(2). **Detailed information seen Appendix C: Key Achievements.**

b. **Teaching:** At the same time, the team members have participated in The third student RDMA Programming competition organized by the High Performance Computing Advisory Council and won the third prize.

The picture of the certificate of RDMA Programming Competition was shown in Appendix C: Key Achievements.

2 Introduction of the Team

As is known to us all, with the rapid development of information construction, High performance computing has been widely known as the third largest scientific method and the first productivity, and it start to develop in the wider areas of information technology and business computing. Therefore, the research on high performance computing is becoming more and more significant. Having realized this development trend, professor Linjie Zhang from school of mathematical science organized and set up an interest group focused on the students who have great interest in this field.

After a period of study, group members have a certain understanding about the related knowledge of high performance computing, such as related hardware, GRID, MIC, MPI and so on. Team members and teachers also listened to some lectures and participated in related training to improve our ability and to expand the range of knowledge. So, when we heard the news that ASC was about to come, everyone is eager to attend this contest. After a series of evaluation and discussion, we finally established a team - deep blue. Interestingly, when talking about our team name, what occur to us first is the super international computer manufactured by American IBM company, which has historic significance. In addition, “blue” is reminiscent of the ocean which represents the Ocean University of China. Also, it is a symbol that our team has a galaxy of talents. Moreover, “deep” represents the depth of our thoughts.

Team members were studying together, struggling together. For ASC, we have done our best! And our slogan is “Surmount limit and Chase dream， The Deep Blue, Stride forward！ ASC, here we are ！"

Here are the team logo and introduction of our group members.

- Team Logo

Figure 2.1 : Team logo

❖ Team Introduction

Team members:

Name	Qinyu Wang
Sex	Male
Education	Undergraduate
University	School of Mathematical Science, Ocean University of China
Grade& Specialty	Senior/Information &Computational Sciences
Personal hobby	System & Software

Name	Peiwen Zhang
Sex	Male
Education	Undergraduate
University	School of Mathematical Science, Ocean University of China
Grade& Specialty	Senior/Information &Computational Sciences
Personal hobby	Computer DIY & Model making and collection

Name	Yingqi Zhao
Sex	Female
Education	Undergraduate
University	School of Mathematical Science, Ocean University of China
Grade& Specialty	Junior/Information &Computational Sciences
Personal hobby	English & System

Name	Chentao Wang
Sex	Male
Education	Undergraduate
University	School of Mathematical Science, Ocean University of China
Grade& Specialty	Junior/Information &Computational Sciences
Personal hobby	Sports

Name	Yanhong Dong
Sex	Female
Education	Undergraduate
University	School of Information science and Engineering, Ocean University of China
Grade& Specialty	Junior/Computer Science and Technology
Personal hobby	Photograph &Sports

Probationary member:

Name	Na Han
Sex	Female
Education	Undergraduate
University	School of Mathematical Science, Ocean University of China
Grade& Specialty	Junior/Information &Computational Sciences
Personal hobby	Skating & Clothes designing

Instructor:

Name	Linjie Zhang
Sex	Female
Academic title	Associate Professor
Department	School of Mathematical Science, Ocean University of China
Education/ Degree Personal hobby	Ph.D. of Science in Engineering

3 Construction of high-performance platform based on Inspur server NF5280M4

3.1 Introduction of hardware parameters and performance analysis.

According to subject, we use INSPUR server NF5280M4 platform as the basis for our design. Before the design of platform, we have a certain understanding about the hardware parameters of this server.

The main parameters of NF5280M4 are as follows:

Table 3.1: main parameters of NF5280M4

Component Name	Model	Power consumption（TDP）
CPU	Intel Xeon E5-2680V3*2	240w
Memory	16G ECC Registered DDR4 2133Mhz *8	7.5w
Hard disk	1T SATA *1	10w

NF5280M4 use Intel Xeon E5-2680v3 processor, which has 12 CPU cores, clocked at up to 2.6GHz, QPI up to 9.6GT / s. The theoretical double precision floating point performance can be achieved 124.8Gflops and Power consumption ratio can up to 1.04Gflops/w. We used DDR4 ECC memory, clocked at 2133 MHz, with a total capacity of 64GB, core memory ratio 2.667GB / core.

As for accelerator, we have the following options:

Table 3.2 : Information of accelerator

Component Name	Parameters	Power consumption（TDP）
Intel Xeon Phi 7110P	61cores 1.208Tflops	300w
Nvidia Kepler K40M	2280 cuda cores 1.43Tflops	235w

In terms of transmission, we chose Infiniband Mellanox ConnectX®-3 HCA card which can support 56Gb/s of FDR Infiniband and 40/56Gb Ethernet connection. SwitchX™ FDR InfiniBand switch selected can provide up to 56Gb/s full bidirectional bandwidth per port. It is designed to carry converged LAN and SAN traffic with the combination of assured bandwidth and granular Quality of Service (QoS).

The parameters of Infiniband router and HCA card are as follows：

Table 3.3: Main parameters of Infiniband router and HCA card

Component Name	Parameters	Power consumption（TDP）
HCA Card	Infiniband Mellanox ConnectX®-3 HCA card, single port QSFP, FDR IB	9w
Switch/GbE switch	10/100/1000Mb/s，24 ports Ethernet switch	30w
Switch/FDR-IB switch	SwitchX™ FDR InfiniBand switch, 36 QSFP port	130w

Performance comparison among MIC, GPU and CPU were showed below.

Table 3.4: Performance comparison

	Intel Xeon E5-2680V3	Intel Xeon Phi 7110P	Nvidia Kepler K40M
Double precision floating-point computation ability	124.8Gflops	1208Gflops	1430Gflops
Number of cores	12	61	2280
Memory Capacity	64GB	16GB	12GB
Power consumption	120w	300w	235w
Power consumption ratio	1.04Gflops/w	4.027Gflops/w	6.085Gflops/w
Core memory ratio	2.667GB/core（Total）	0.262GB/core	0.0053GB/cuda core

Power consumption ratio describes cluster computing performance which can be provided under the power unit. In the limited power, the higher the power consumption ratio is, the higher the computing performance of cluster is. The core memory ratio reflects the number and proportion of computing core memory. The larger the value is, the larger the scale of a single MPI process is. So that we can reduce the communication time.

As for performance perspective, K40M can reach the highest power consumption ratio. However, in the case of limited power, in order to obtain maximum performance with K40M, we need to make major changes to the program if we want to migrate the applications to the CUDA environment, which is troublesome. By contrast, although the power consumption of 7110P is relatively high, the transplantation is simple. Also, it has a high compatibility and flexibility as it can participate in operation as a stand-alone node.

3.2 Constructing method of platforms

3.2.1 Proposal A (INSPUR-CPU Cluster)

The main idea about CPU cluster is that we need to increase to the number of nodes as much as possible to improve the computing ability of the cluster. The design table showed below.

Table 3.5 : Design table

Name	Power consumption
Switch/GbE switch	30w
Switch/FDR-IB switch	130w
(Nodes of CPU)*10	2665w
Total	2825w

Topology graph for the design about cluster showed as follow.

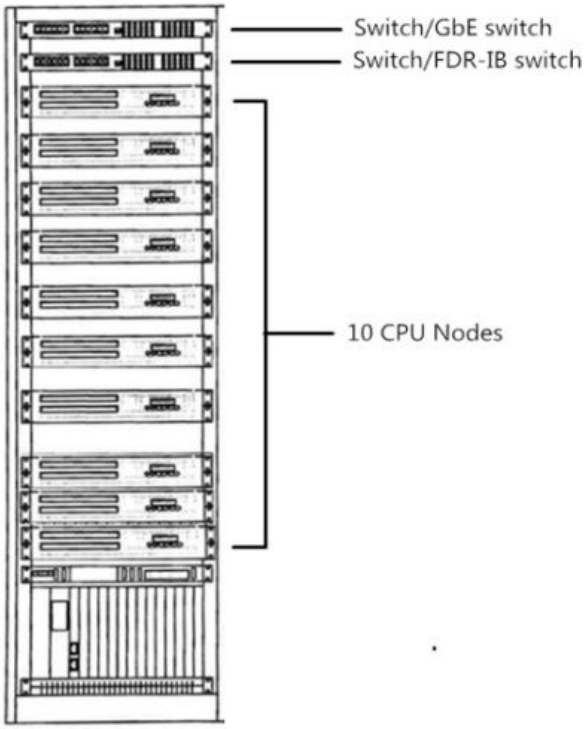


Figure 3.1 : Topology graph for proposal A

Wherein, the power of CPU node comprises two Intel Xeon E5-2680V3 which has 64GB of memory in total, 1TB hard disk and HCA Card which is 266.5w in total.

For this CPU cluster, the theoretical double-precision floating point peak 2496Gflops while power consumption ratio is 0.8835Gflops/w.

The software environment of the platform showed below.

Table 3.6: Software environment

Name	Versions
System	Redhat Enterprise Linux Server release 6.5
Function base	i_mkl_2015.1.133
MPI	OpenMPI-1.6.5 IMPI-5.0.2.044
Compiler	ifort-15.0.1 icc-15.0.1
Grid data processing software	Netcdf-3.6.3
Performance analysis software	Vtune_amplifier_xe_2015.1.1.38310 Inspur TEYE

3.2.2 Proposal B (INSPUR-GPU Cluster)

The main idea about GPU cluster is that the whole cluster can play a greater performance through the use of high power consumption ratio of GPU in the case of limited power. The design table showed below.

Table 3.7 : Design table

Name	Power consumption
Switch/GbE switch	30w
Switch/FDR-IB switch	130w
(Acceleration nodes of GPU)*3	2209.5w
(Nodes of CPU)*2	533w
Total	2902.5w

Topology graph for the design about cluster showed as follow.

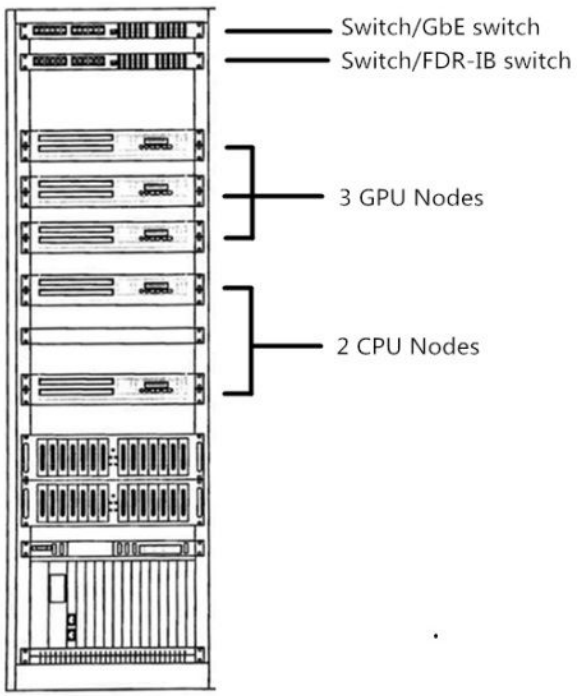


Figure 3.2 : Topology graph for proposal B

Wherein, the power of CPU node comprises two Intel Xeon E5-2680V3 which has 64GB of memory in total, 1TB hard disk and HCA Card and two Nvidia K40M card, which is 736.5w in total. The power of GPU node comprises two Intel Xeon E5-2680V3 which has 64GB of memory in total, 1TB hard disk and HCA Card which is 266.5w in total.

For this cluster, the theoretical double-precision floating point peak 9828Gflops while power consumption ratio is 3.386Gflops/w.

The added software was listed below on the basis of software environment in proposal A.

Table 3.8: Added software

Name	Versions
CUDA	CUDA-6.5

3.2.3 Proposal C (INSPUR-MIC Platform)

The main idea about MIC cluster is mainly focus on high performance of MIC accelerator card and simple code porting. Also, obtaining higher performance while maintaining compatibility for different applications should be taken into consideration. The design table showed below.

Table 3.9: Design table

Name	Power consumption
Switch/GbE switch	30w
Switch/FDR-IB switch	130w
(Acceleration nodes of MIC)*3	2599.5w
Total	2759.5w

Topology graph for the design about cluster showed as follow.

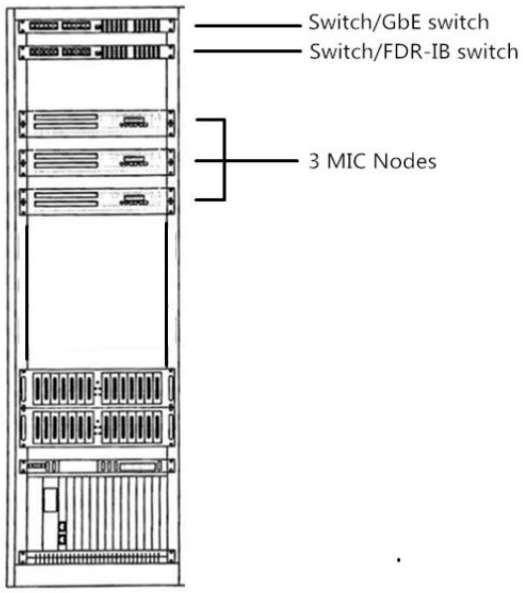


Figure 3.3 : Topology graph for proposal C

Wherein, the power of MIC node comprises two Intel Xeon E5-2680V3 which has 64GB of memory in total, 1TB hard disk and HCA Card and two Intel Xeon Phi 7110P card, which is 866.5w in total.

For this cluster, the theoretical double-precision floating point peak 7996.8Gflops while power consumption ratio is 2.8979Gflops/w.

The software environment of the platform is same as proposal A

3.3 Comparison and analysis between proposals

After comparison, the results were as follows:

Table 3.10 : Comparison

	Proposal A	Proposal B	Proposal C
	INSPUR-CPU cluster	INSPUR-GPU cluster	INSPUR-MIC cluster
Number of nodes	10	5	3
Total power consumption	2825w	2902.5w	2759.5w
the theoretical double-precision floating point	2496Gflops	9828Gflops	7996.8Gflops
Power consumption ratio	0.8835Gflops/w	3.386Gflops/w	2.8979Gflops/w

From the point of view of three design proposals , although CPU cluster computing performance is weak, the CPU itself has a very strong application compatibility. So it has greater flexibility for different tasks. The GPU cluster has the highest double precision floating point computing ability and the highest power consumption ratio. Also, its calculated performance is great. However, it needs to make major changes to the program if we want to migrate the applications to the CUDA environment, which reflects weak compatibility. Besides, the ability to adopt to complex tasks is weak. MIC clusters have higher double-precision floating point capability and a higher power consumption ratio compared with GPU and CPU cluster. And it has great portability. This cluster can be guaranteed to maintain high performance when treating different tasks by offload, SMP mode switching, etc. In addition, the application to MIC platform transplantation is simple and the code changes little, even no change. It can be said that MIC cluster gives consideration to both performance and compatibility. Among these three proposals , we refer to MIC cluster proposal for the reason that it can gives consideration to both performance and compatibility. For security, proposal C also set aside a portion of the power consumption as the fluctuation range of the whole system in order to ensure the power limit of not more than 3000W of the cluster in the process of operation.

4 Construction of high-performance platform based on our team available resources.

4.1 The introduction of hardware parameters and performance analysis.

At present, our team has a super computing platform consists of fifteen sets of blade computing nodes, two GPU nodes, two MIC nodes and a management node. The calculation ability of the entire computing platform has reached 13.4704TFLOP/s. The memory capacity has reached 736GB, equipped with 96TB memory array. The communication computing platform based on IB network, which can provide a platform for the exchange capacity of 4.032Tbps. So, The platform can meet the college supercomputer task.

Here are detailed parameters:

- Blade computing nodes:

Table 4.1: Parameters of blade computing nodes

Component Name	Model	Power consumption（TDP）
CPU	Intel Xeon E5-2630V3*2	170w
Memory	Samsung 8G ECC Registered DDR4 2133Mhz *4	7.5w
Hard disk	160G MLC SSD	5w

- GPU acceleration nodes:

Table 4.2: Parameters of GPU acceleration node

Component Name	Model	Power consumption（TDP）
CPU	Intel Xeon E5-2630V3 *2	170w
Memory	Samsung 8G ECC Registered DDR4 2133Mhz *8	7.5w
Hard disk	160G MLC SSD	5w
Accelerator card	Nvidia Kepler K40M *2	470w

- MIC acceleration node:

Table 4.3: Parameters of MIC acceleration node

Component Name	Model	Power consumption（TDP）
CPU	Intel Xeon E5-2630V3 *2	170w
Memory	Samsung 8G ECC Registered DDR4 2133Mhz *8	7.5w
Hard disk	160G MLC SSD	5w
Accelerator card	Intel Xeon Phi 7110p *2	600w

As cluster system, the communication ability among nodes directly affects actual operation ability of the whole cluster. Our cluster chose the Mellanox MSX6025 switch which can provides 56Gpbs Bidirectional no-blocking Bandwidth and support VPI that can provide interconnection between Infiniband and Ethernet.

Comparison among hardware performance of running platform were showed as follows:

Table 4.4 : Comparison among hardware performance

	Blade computing nodes	GPU computational nodes	MIC computational nodes
the theoretical double-precision floating point	153.6Gflops	3013.6Gflops	2569.6Gflops
Power consumption	187.75w	657.75w	787.75w
Power consumption ratio	0.818Gflops/w	4.5817Gflops/w	3.2619Gflops/w
Accelerator card	No card	Nvidia Kepler K40M	Intel Xeon Phi 7110p

In contrast, GPU computational modes have the highest power consumption ratio, high performance and low power consumption, which have more advantages than MIC nodes in the case of limited power (3000w). The power consumption of CPU nodes is relatively low. We can obtain good test results in HPCG test through the rational allocation of CPU and memory in proportion to the size of the relationship, which is pretty flexible.

4.2 The design of testing platforms

Limited by the number of hardware we have, we cannot fully realized the test done by the platform provided by competition. Thus, we choose the hardware we have to build the experimental platform to test the software performance to find the regular. We also designed three test platform to complete the test of our work.

4.2.1 Platform A (OUR-CPU cluster)

The CPU testing platform consists of 13 blade computing nodes, while each node is equipped with two Intel Xeon 2630v3 CPU. Each CPU with 8 cores, clocked at 2.4GHz. The cluster theoretical double precision floating-point computing power can reach 1996.8Gflops and the power

consumption ratio is 0.7884Gflops/w. In the design of the experimental platform, we consider the power conversion efficiency of its own under the PDU detection power case. Because of the power conversion efficiency of the power supply itself, there will be some power consumption caused by power itself which lead to the increase of cluster power consumption. So, it is necessary to appropriately relax the design of experimental platform. The design was showed as follow:

Table 4.5 : The design of testing platform A

Name	Power consumption
Switch/GbE switch	30w
Switch/FDR-IB switch	130w
(Nodes of CPU)*13	2372.5w
Total	2532.5w

Topology graph for the design of testing platform A showed as follow

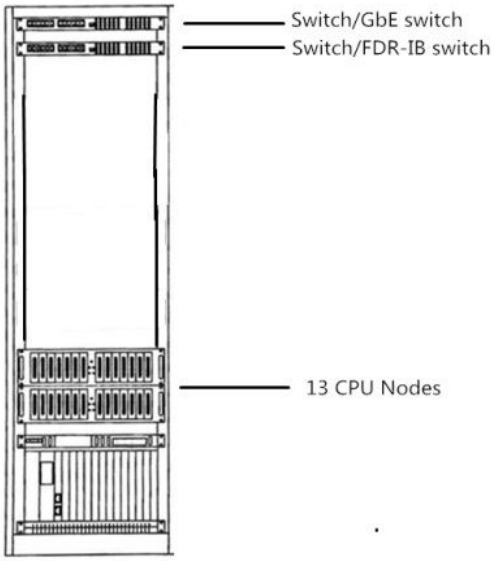


Figure 4.1 : Topology graph for testing platform A

Detailed information about the software platform we have showed below.

Table 4.6 : Software platform

Name	versions
System	Redhat Enterprise Linux Server release 6.5
Function base	i_mkl_2015.1.133
MPI	OpenMPI-1.6.5 IMPI-5.0.2.044
Compiler	ifort-15.0.1 icc-15.0.1
Performance analysis software	Vtune_amplifier_xe_2015.1.1.38310 Inspur TEYE

4.2.2 Platform B (OUR-GPU cluster)

The GPU testing platform consists of two GPU accelerated nodes, while each node is equipped with two Intel Xeon 2630v3 CPU. And this platform was also equipped with two NVIDIA K40M accelerator card. The cluster theoretical double precision floating-point computing power can reach 6027.2Gflops and the power consumption ratio is 4.114Gflops/w. The design was showed as follow:

Table 4.7 : The design of testing platform B

Name	Power consumption
Switch/GbE switch	30w
Switch/FDR-IB switch	130w
(Acceleration nodes of GPU)*2	1305
Total	1465w

Topology graph for the design of testing platform B showed as follow

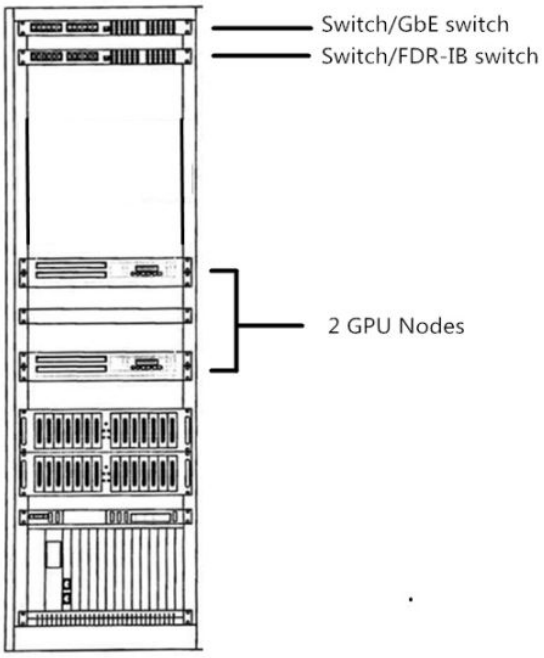


Figure 4.2 : Topology graph for testing platform B

The added software was listed below on the basis of software environment on testing platform A.

Table 4.8: Added software

Name	Versions
CUDA	CUDA-6.5

4.2.3 Platform C (OUR-MIC cluster)

The MIC test platform consists of two MIC acceleration nodes, while each node is equipped with two Intel Xeon 2630v3 CPU, also equipped with two Intel Xeon Phi 7110p accelerator. The cluster theoretical double precision floating-point computing power can reach 5139.2Gflops and the

power consumption ratio is 2.979Gflops/w. The design was showed as follow:

Table 4.9: The design of testing platform C

Name	Power consumption
Switch/GbE switch	30w
Switch/FDR-IB switch	130w
(Acceleration nodes of MIC)*2	1565
Total	1725w

Topology graph for the design of testing platform C showed as follow

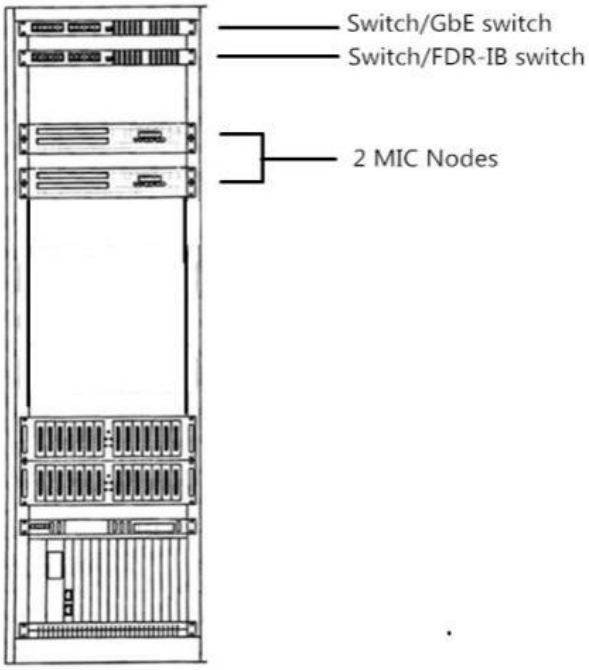


Figure 4.3 : Topology graph for testing platform C

The software environment of the testing platform is same as testing platform A.

5 HPCG test

5.1 Introduction

High Performance Conjugate Gradients (HPCG) benchmark is a performance evaluation tool for super computer cluster system. At present, HPCG is available as a standard of global top 500 supercomputer rankings. But compared with HPCC, HPCG can better reflect the performance of cluster in many aspects.

HPCG test pays more attention on the calculation and data access process during the running process of the applications. Emphasizing the matching between calculation and data storage and

the pursuit of processor performance, memory capacity and the balance between bandwidth and network transmission lead the result that HPCG test is more stringent than HPCC test.

The substance of HPCG test is solving a discrete three-dimensional partial differential equations and using the Preconditioned Conjugate Gradient (PCG) method to obtain iterative sparse system results in the solution process. A single degree of freedom of the thermal problem is similar to a Dirichlet boundary condition of diffusion model. In the problem, the global problem size is

$$(nx * npx) \times (ny * npy) \times (nz * npz)$$

Wherein,

- (nx ×ny ×nz) means a three-dimensional grid in the MPI process,
- the size of (nx ×ny ×nz) is set automatically by HPCG applications,
- nz means the number of nodes,
- (nx * ny) means a two-dimensional grid in the process of single node,
- (npx ×npy ×npz) means a MPI process corresponding to the scale of the problem, which can pass the size thorough the command line parameters.

The core of HPCG test is the Conjugate Gradient method. The conjugate gradient method is a specific method for solving linear equations Ax=b, but it needs frequently read irregular data in the process of iteration, which increases the burden of I/O and reduces the performance. Besides, in the actual calculation, the conjugate gradient method is prone to the problem of slow convergence because of the large condition number of the matrix. The Preconditioned Conjugate Gradient (PCG) method use the preconditioned method on the basis of the Conjugate Gradient method, including G-S iteration, incomplete LU decomposition and incomplete cholesk decomposition method.

The Preconditioned Conjugate Gradient (PCG) method has four steps: The matrix vector product, preconditioned, vector update and vector inner product. In the calculation process, we use SPMD mode to achieve distributed memory parallel storage and this process is achieved by MPI. The matrix A is divided into blocks, and each block matrix, vector x and b were assigned to the corresponding MPI process.

In the Preconditioned Conjugate Gradient (PCG) method, it mainly includes two kinds of communication modes. One is Neighborhood collective, which is applied to the calculation of the matrix vector product. During the process, each process has data exchange with at least one other process. The other is the All-reduce collective, which is applied to the calculation of the vector inner product. During the process, each MPI process first calculate the local matrix dot product value and then exchange the values with other processes, which each processor can receive the final results. In two modes of communication, All-reduce collective mode affects the performance of cluster system more for the reason that a MPI process must wait for the vector inner product results to complete the calculation. The performance of the All-reduce operation is usually regarded as an important feature of evaluation of high performance parallel system.

5.2 Analysis of makefile and compiler optimization

When installing HPCG, the most important is to modify and determine the parameters in makefile.

During the installation of HPCG, different systems need different choices. Take our cluster for example, what we chose is Linux.MPI. The parameters in makefile which need to be modified were listed below.

Table 5.1: The parameters in makefile which need to be modified

Parameter's Name	Meaning of parameter
TOPdir	Installation path of HPCG
MPdir	Installation path of MPI
HPCG_OPTS	Installation options of HPCG, including the opening of MPI, the opening of OpenMP, the opening of DEBUG, the opening of DETAILED DEBUG
CXX	Installation path of C++ compiler
CXXFLAGS	Compiler options of C++

The use of compiler options can improve the running efficiency of the code and the efficiency of memory, and make the code better match Hardware Architecture from the agency in operating. The degree of optimizing single code compiler options are limited, so we mainly use combinatorial optimization in the process of compiler optimization.

Next, we gave a detailed analysis about commonly used compiler options in Intel compiler in order to improve the performance of HPCG program from the compiler optimization level.

1) -ffast-math

Violating the IEEE/ANSI standards to increase the calculation speed of floating point. It is relatively dangerous. So, we considered this option only when the compiler does not need to strictly comply with IEEE standard and the floating-point computation of the program is intensive. Considering that our HPCG does not need to strictly comply with the IEEE standard and there are a great amount of calculation in grid computing, so the optimization effect is obvious.

2) -ftree-vevtorize

The fundamental idea of vectorization is to exchange space for time. According to vectorization, it can significantly reduce the use of loop body, which can reduce calculation fundamentally. Although vectorization may produce a huge matrix, it is not a fatal problem in the face of massive storage today.

3) -O3

Optimization level of O3 is to open several low security options based on O2. The optimization option has enabled the vast majority of security in O2. The added optimization.

options are:

- -finline-functions : Allows the compiler to choose some simple functions to be opened at the place where they are called, relatively safe, suitable for server level CPU architecture whose level two cache is large ;

- -funswitch-loop : Put the variable which doesn't change its value out of loop body.a it can significantly reduce reduce the calculation of loop body.

- -fgcse-after-reload : In order to remove excess overflow, perform an additional elimination step after the heavy load. It can improve the memory efficiency.

The O3 option is common in compiler optimization for the reason that the security of O3 options are high in optimization. Vector optimization is the optimization method which has been proved very effective, especially for matrix operation, vectorization has great advantage.

NOTE: In HPCG makefile, it has already used optimization options, including O3, ffast-math, ftree-vectorize, ftree-vectorizer-verbose=0, which has a high level of optimization.

5.3 Analysis of main parameters and output file

Main parameters which should be tested in HPCG are listed below:

Table 5.2 : Main parameters

Name	Meaning
nx	x-dimensional size of block matrix calculated in each MPI process
ny	y-dimensional size of block matrix calculated in each MPI process
nz	z-dimensional size of block matrix calculated in each MPI process
T	The longest test time allowed
OMP_NUM_THREADS	The corresponding number of threads per process

In the next experiment, we experimentally confirm the above parameters influence on the test results, and explore the regular in order to achieve the best settings.

The HPCG test will generate a test report whose suffix is yaml after the test. In this test report, all the results are presented in detailed. Here is the illustration for output file:

Table 5.3: Illustration for output file

Content	Illustration
Machine Summary:	
Distributed Processes	Number of process
Threads per processes	Number of threads of per processes
Global Problem Dimensions:	
Global nx	Global dimension of nx
Global ny	Global dimension of ny
Global nz	Global dimension of nz
Processor Dimensions	
npx	Processor Dimension of npx
npy	Processor Dimension of npy

npz:	Processor Dimension of npz（equals to the number of nodes）
Local Domain Dimensions:	
nx	Local Domain Dimension of nx
ny	Local Domain Dimension of ny
nz	Local Domain Dimension of nz
Benchmark Time Summary:	
Optimization phase	
DDOT	DDOT time overhead
WAXPBY	WAXPBY time overhead
SpMV	SpMV time overhead
MG	MG time overhead
Total	Total time over head(less than T)
Floating Point Operations Summary:	
Raw DDOT	Raw DDOT operating frequency
Raw WAXPBY	Raw WAXPBY operating frequency
Raw SpMV	Raw SpMV operating frequency
Raw MG	Raw MG operating frequency
Total	Total operating frequency
Total with convergence overhead	Total with CG overhead
GFLOP/s Summary:	
Raw DDOT	Raw DDOT performance
Raw WAXPBY	Raw WAXPBY performance
Raw SpMV	Raw SpMV performance
Raw MG	Raw MG performance
Raw Total	Raw Total performance
Total with convergence overhead	Total with convergence overhead
Total with convergence and optimization phase overhead	Total with convergence and optimization phase overhead
User Optimization Overheads:	
Optimization phase time (sec)	Optimization phase time overhead
Optimization phase time vs reference SpMV+MG time	Optimization phase time vs reference SpMV+MG time
DDOT Timing Variations:	
Min DDOT MPI_Allreduce time	Min DDOT MPI_Allreduce time overhead
Max DDOT MPI_Allreduce time	Max DDOT MPI_Allreduce time overhead
Avg DDOT MPI_Allreduce time	Avg DDOT MPI_Allreduce time overhead
********** Final Summary **********:	
HPCG result is VALID with a GFLOP/s rating of	HPCG result is VALID with a GFLOP/s rating of (The most important value)

Among these parameters, there are six parameters which should be pay more attention to:

1) Benchmark Time Summary : Time-use for each part.
2) Floating Point Operations Summary : Floating Point Operations

3) GFLOP/s Summary : Test result. From this parameter, it can be seen that the CG method has higher efficiency than other methods.
4) User Optimization Overheads : The user cost of optimization, mainly reflects the effect of compiler options
5) DOT Timing Variations : Time-consuming in MPI communication. It is the parameter that evaluate MPI communication efficiency.
6) ★HPCG result is VALID with a GFLOP/s rating of : the most worth of attention. It gives the final evaluation of the HPCG test.

5.4 Experimental environment

We make use of the three platforms given in chapter 4, that is platform A (OUR-CPU cluster), platform B (OUR-GPU cluster) and platform C (OUR-MIC cluster). See chapter 4 for details.

5.5 Test results on platform A (OUR-CPU Cluster)

The test platform consists of 13 nodes, while each node core number is 16. So each node's number of processes is fixed at 16.

5.5.1 Test 1: Selection for optimal parameters values (nx, ny,nz)

In the following test, we use all the nodes in our platform and obtain the relationship between hardware platform and parameters nx, ny, nz through test.

There are a great amount of combination of nx, ny, nz. We first verified from the circumstances which the values of three parameters are different. Whether it has influence on the results,we get several groups of nx, ny, nz. Here is the result:

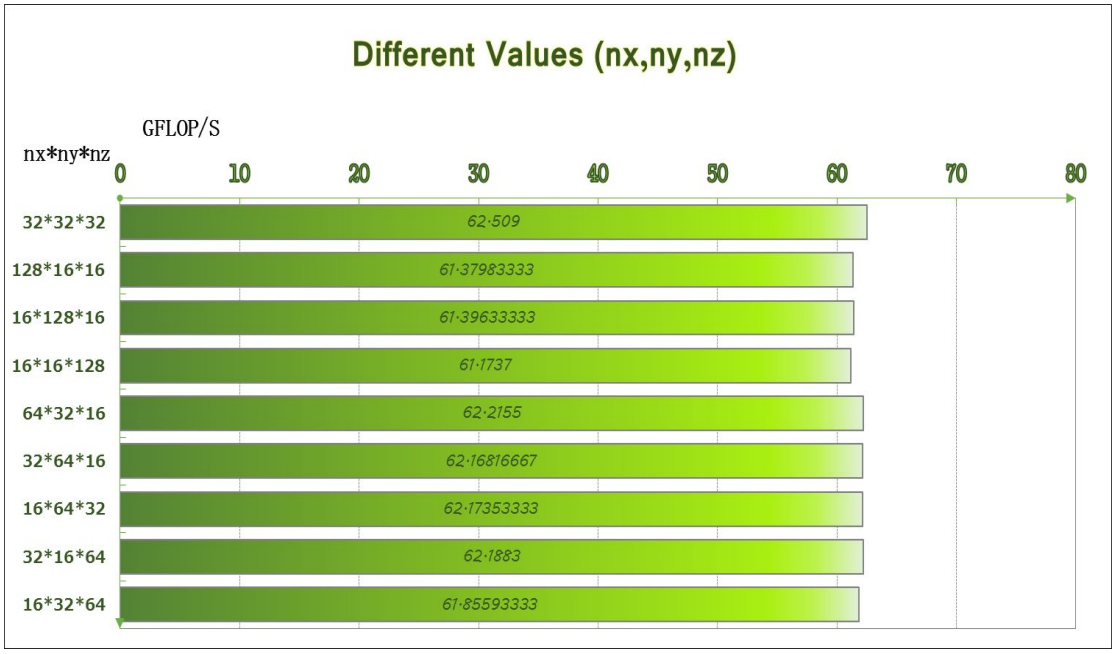


Figure 5.1 : Test results in different values

As is shown in the graph above, the gap among HPCG results is very small under the combination of different collocation, which is only about 1%, less than the influence brought by system itself. However, the test results are relatively high when three parameters nx, ny, nz are equal. Therefore, we make the following experiment.

Here, we investigate the optimal value of nx (or ny, or nz). According to the requirements of HPCG, the parameter should at least be 16 and must be in multiples of 8. Set the step length of each step is 16, We tested until the required memory is greater than the machine memory (that is nx=ny=nz=128). Here is the result:

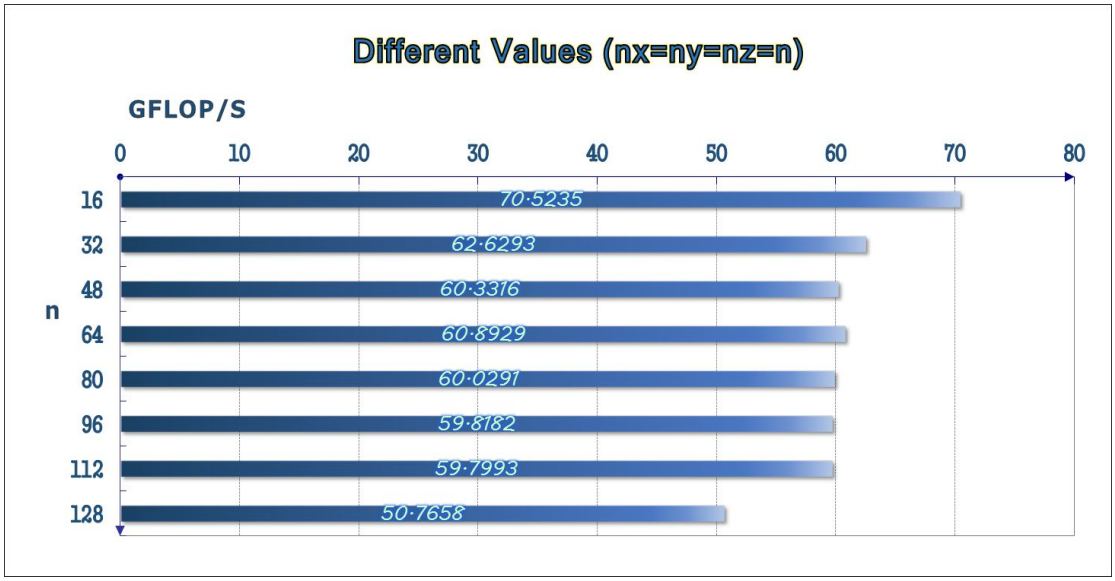


Figure 5.2: Test results when three parameters are equal.

Seen from the graph above, the larger the value of nx is, the worse the result is. This is because with the increase of nx, the scale of the problem is increasing which leads to the increase of memory request. However, the size of memory provided by the system is limited. There would be waiting and operation continuing to request memory which caused the extension of calculating time for HPCG test. So, the HPCG test value decreased. From the figure above, for our platform, we can obtain best effort when nx=ny=nz=16.

5.5.2 Test 2: Influence of parameter T on experimental results

As for the parameter T, we think there is an optimal test time. Next, we use all nodes on our platform to determine the value through the experiment. In the case that three parameters (nx, ny, nz) all equal to 16, the test results were shown as follow.

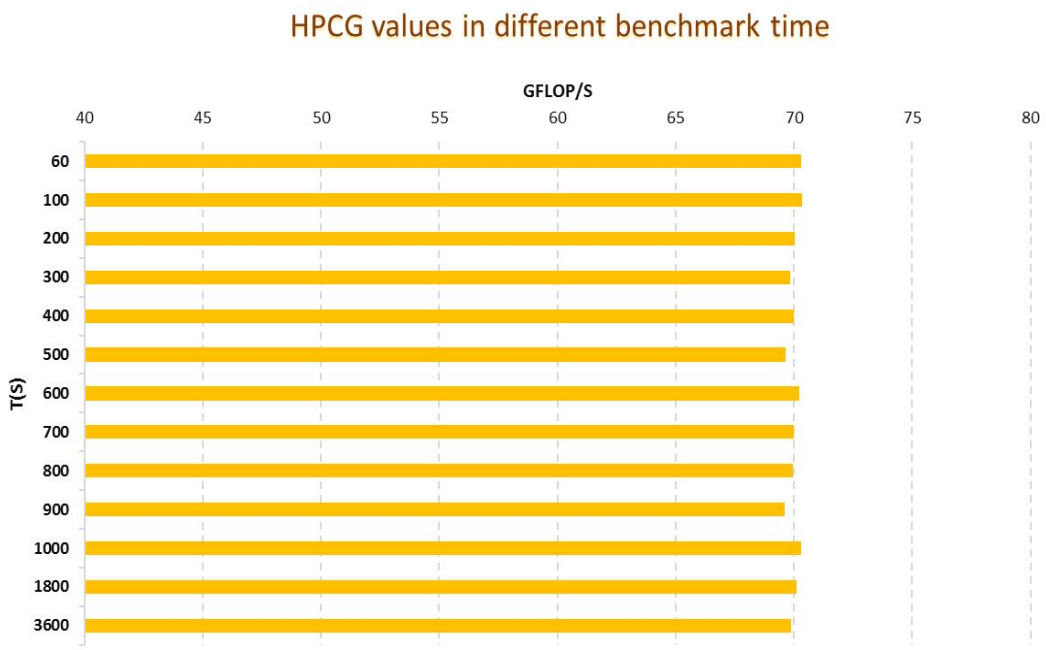


Figure 5.3 : HPCG values in different benchmark time

We can clearly find that there are no obvious regular between the results and the value of parameter T. And it is unable to find a definite relationship showing the relationship between the T and HPCG test results. Therefore, in the process of testing, it is necessary to test multiple groups of values of T to get the best results.

5.5.3 Test 3: The test results with the increase of the number of nodes

Here, we gave the test results with the increase of the number of nodes, see Table 5.4 and Figure 5.4. The number of threads per node is fixed to 16 (the number of cores with each node of the same).

Table 5.4: Test result with different number of nodes (NN)

NN	HPCG test result(GFLOP/S)	Performance improvement ratio
1	8.69	1.00
2	13.79	1.59
4	21.96	2.53
6	32.76	3.77
8	40.27	4.63
10	49.19	5.66
12	59.40	6.84
13	70.32	8.09

Figure 5.4 : Performance improvement ratio

Through the test, we found that with the increase of the number of nodes, namely, the number of processes, HPCG test value is a linear growth and accorded with proportional relationship. So, we can predict in matches cluster HPCG application performance results now.

5.5.4 Conclusion

1) When selecting the collocation of parameters, we should choose collocation like nx=ny=nz. But the scale of the problem size need to be consider according to the hardware environment. The local dimension of the HPCG program describes the size of the problem in a MPI process of computing, and the process of a MPI corresponding to a CPU core. Therefore, the scale of the problem is corresponded to the memory size which each CPU can be assigned to. The value is calculate as

$$memory\ per\ core = \frac{total\ memory}{number\ of\ cores}$$

The more memory each CPU can be assigned to, the larger the parameter can be set. That is to say, compared with HPCC test, HPCG test pay more attention to reasonable collocation of CPU and memory. In our view, on the basis of current hardware environment, optimal parameters can be set to 32 only when the size of memory is doubled.

2) In the aspect of the selection of parameter T, selecting large value has more advantage than selecting small value. We think that we can start to use default T=60 for the first test, then set the step length of each step is 60 to gradually increase the size of the T so that we can choose a good value of parameter T.

3) As for expandability, it can be found from test 3 that with the increase in the number of nodes, HPCG results and the number of nodes is proportional. Thus, HPCG values can be predicted

by the fitting function while adding the same computing nodes.

On our testing platform A(OUR-CPU Cluster), optimal parameters collocation is nx=ny=nz=16, T=400. Under these optimal parameters, the HPCG test value is 70.9577 Gflops/s. The platform calculation capacity is 1996.8 Gflops/s. So, the proportion could reach 3.553%, a relatively good result. **Optimized results of HPCG test were shown in Appendix A: HPCG(CPU).**

5.6 Test results on platform B (OUR-GPU Cluster)

5.6.1 Test 1: Selection for optimal parameters values (nx, ny, nz)

According to the experience we learn from testing platform A, we only consider nx=ny=nz. The official website of HPCG test has been given a reference to the optimal value under the version of CUDA. We still set 16 as a start, set the step length of each step is 16. The test results are as follows:

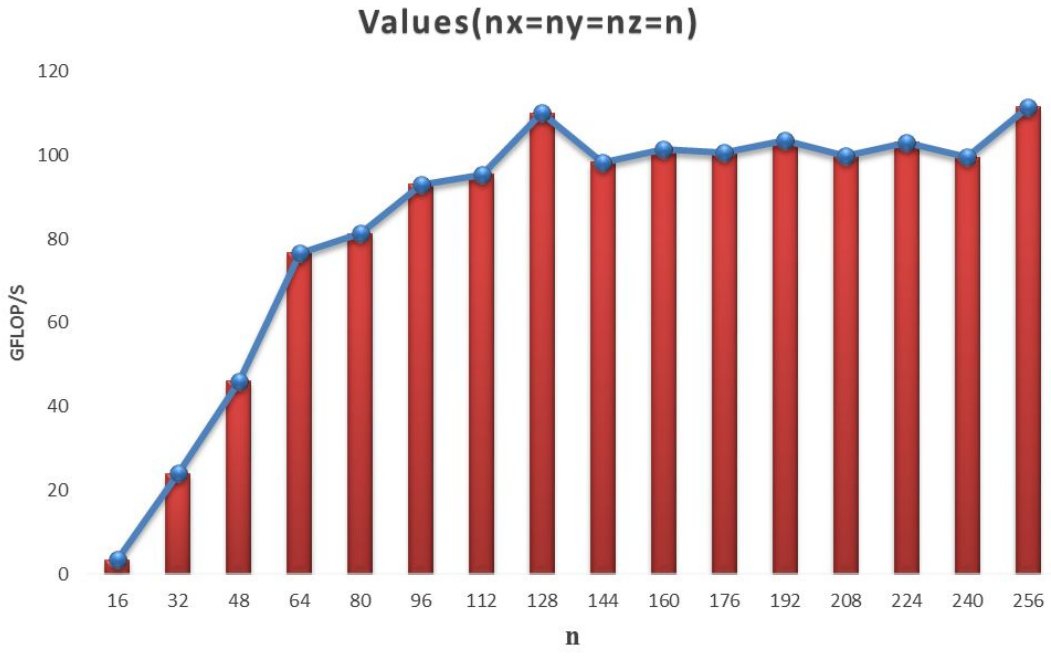


Figure 5.5 : Test results

The optimal parameters given by the official is 256*256*256. According to the test, we found that the value of 256*256*256 is actually optimal. But, there is a “peak” at 128*128*128, whose value is close to the value in 256*256*256. We believe that the reason why this value appear is related to the architecture of NVIDIA graphics card.

5.6.2 Test 2: Influence of parameter T on experimental results

Determined the value of nx, ny, nz, we tested the optimal value of parameter T. Here is the result:

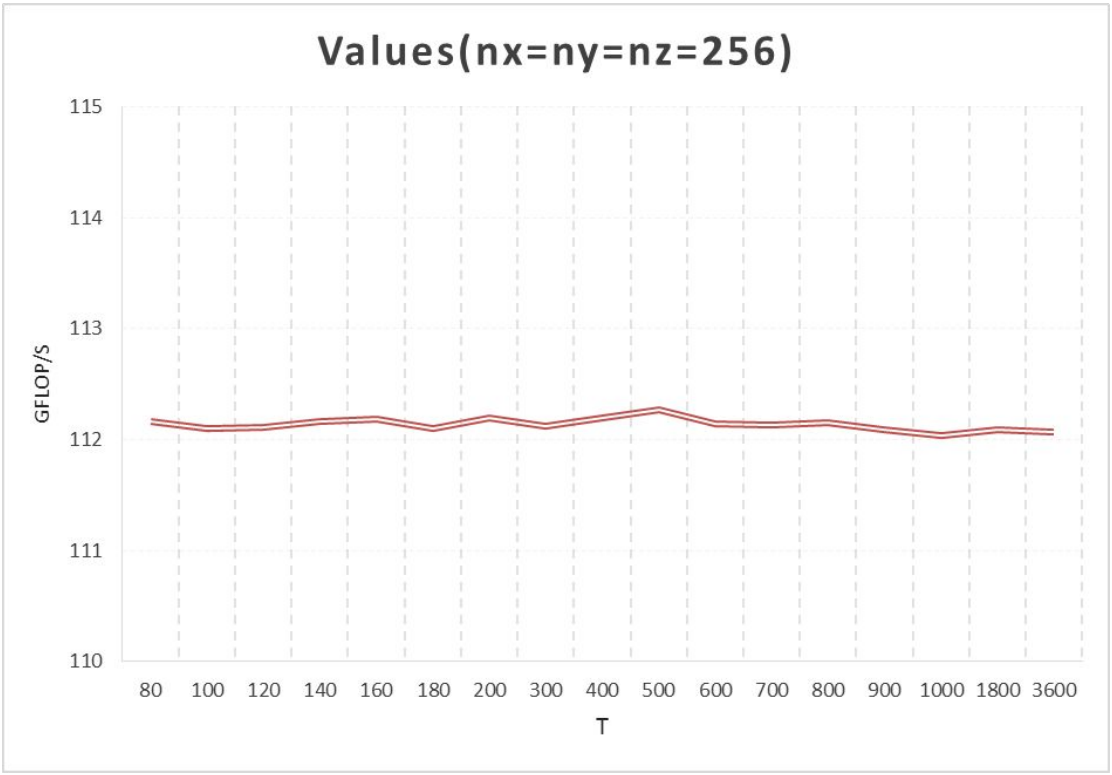


Figure 5.6: the values of parameter T

According to the test results, we found that the influence of parameter T on the results is not very obvious. But it met the regular we found before that result were better when the value of parameter T was larger. From the results, we can obtain the optimal value when T equals to 500, which was close to the result we get from testing platform (T=400). From other side, it proved the correctness of the regular in the selection of parameter T.

5.6.3 Test 3: Influence of OMP_NUM_THREADS on experimental results

Compared with the CPU testing platform, HPCG test with CUDA need to set the number of threads. This is because the GPU accelerator card can reduce communication overhead through OpenMP parallel and improve performance. Less number of threads cannot make full use of hardware resources while too much number of threads reduce performance because of the conflict caused by limited resources.

As the optimal parameters were determined in test 1 and test 2 (nx=ny=nz=256, T=500), we modified the OMP_NUM_THREADS environment variable. The test result was showed in Figure 5.7. From the figure, it can be concluded that HPCG can give the best results in the base of OMP_NUM_THREADS=8

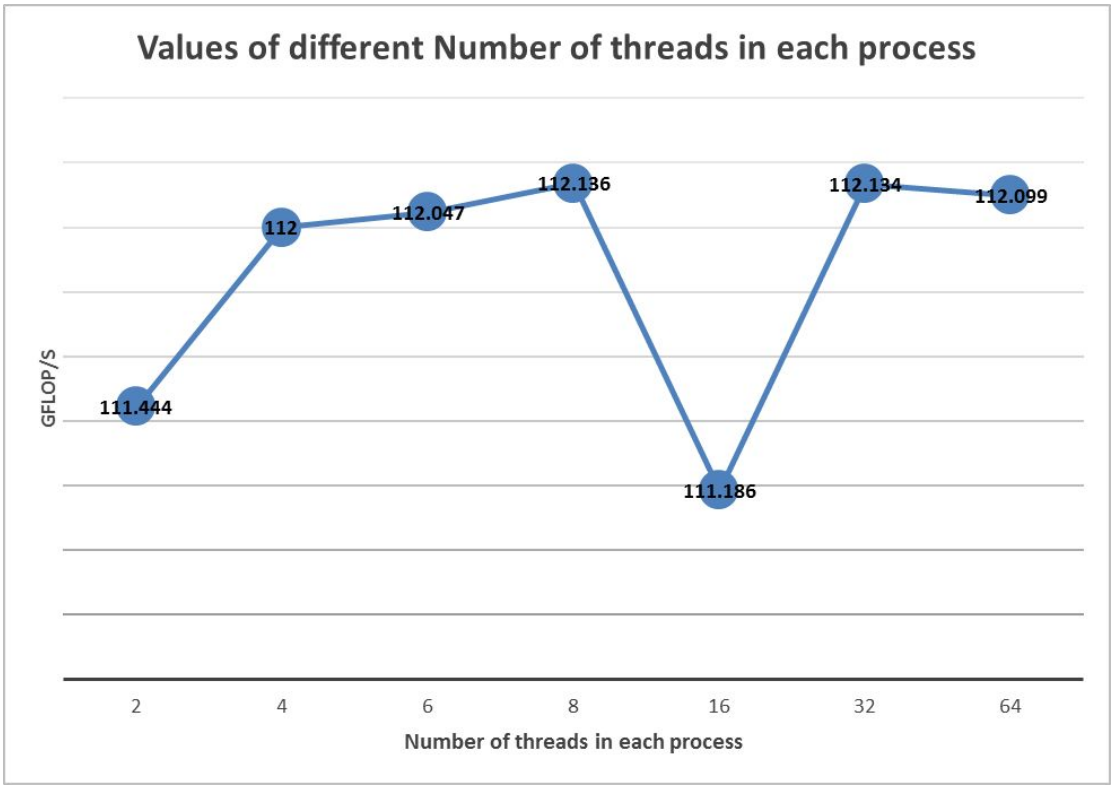


Figure 5.7 : Values of different Number of threads in each process

5.6.4 Conclusion

Through the HPCG test with CUDA, for Nvidia K40M, we could obtain the best result when the optimal parameters collocation is nx=ny=nz=256, T=500, OMP_NUM_THREADS=8. The best result was 112.136Gflops. This cluster has four accelerator cards and theoretical floating-point performance is 5720Gflops. So, the proportion was 1.96%, which is a good result.

The different between GPU accelerator card and CPU platform lies in the architecture and memory. GPU accelerated card comprises a plurality of processing units which share one block of GDDR memory welded on the PCI board so that reading speed is faster than the memory on the motherboard. At the same time the design of architecture of GPU is more suitable than CPU for CG operation. In theory, it could obtain a better result than what could be obtained on CPU platform. However, compared to CPU, in order to obtain high speed memory access, GPU accelerator card's memory is integrated in the PCI board which is lack of scalability. The CPU platform can expand the memory capacity according to the demand, especially in the face of this HPCG test which need to consider the collocation of CPU core and memory, reasonable collocation of memory and processor is highlighted.

Our testing results reflects that the memory of K40M is less for HPCG test, which did not play full performance in HPCG test. For a certain type of accelerator card, the card calculation capability memory size and core are fixed and cannot be changed, so you can not improve the memory size to improve test scores.**Optimized results of HPCG test were shown in Appendix A: HPCG(GPU).**

5.7 Test results on platform C (OUR-MIC Cluster)

Under the mode of OFFLOAD, the installation of HPCG test is a little trouble than others. According to the needs of specific compiler, corresponding to select the MPI library file, modify the HPCG path, MPI path and HPCG test parameters. After the success of the compiler, we set the number of processes as 1, 2, 8 to test the performance under offload mode. As shown in Figure 5.8.

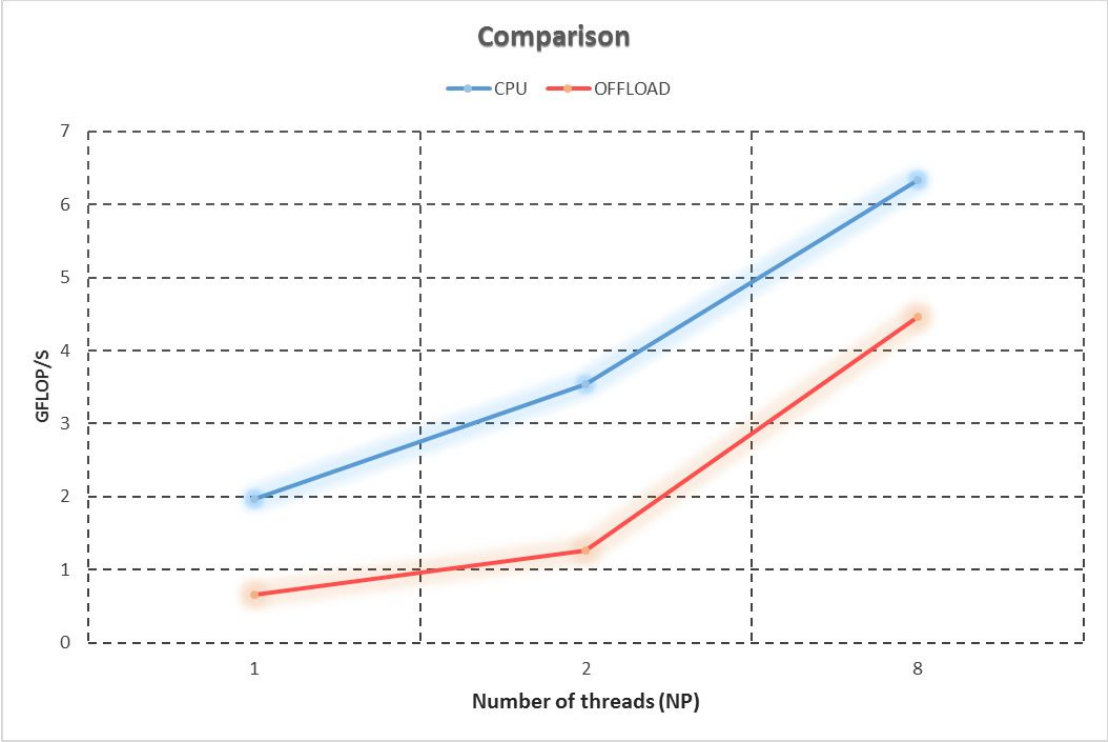


Figure 5.8: Performance comparison between CPU and OFFLOAD

Through the test, we found that the offload mode has no advantage for pure CPU platform. To the contrary, the performance fell. We speculated that the reasons for such problems is the problem caused by small scale. During the experiment, setting a large scale of the problem will lead to the test time more than benchmark time that we specified under memory constraints. Besides, with the limitation of the number of CPU cores, we could not open more processes to carry on symmetric offload, which also lead that performance did not meet expectations. **Optimized results of HPCG test were shown in Appendix A: HPCG(MIC).**

5.8 Comparison of test results on three platforms

After comparison, the results were as follows:

Table 5.5: Comparison table

	Testing platform A （CPU）	Testing platform B （GPU）	Testing platform C （MIC）
The theory of floating point arithmetic capability	1996.8Gflops	6027.2Gflops	5139.3
Power consumption	2532.5w	1465w	1725w
Power consumption ratio	0.7884Gflops/w	4.114Gflops/w	3.01Gflops/w
Best result	70.9577Gflops	112.136Gflops	4.4Gflops(offload mode)
Proportion	3.553%	1.96% (only GPU)	0.36% (offload mode)

By contrast, we found that HPCG test got the best result on CPU platform. That is to say, That is to say, the CPU platform has achieve the best balance between computational and memory core. And we believed that further increase the memory size of CPU platform can further increase the HPCG test value.

By comparison, the performance of GPU platform is not as good as what we expected. This is because GPU accelerated card carries smaller memory and GPU stream processor and memory collocation is not suitable, which is also an important problem GPU has. Due to the special architecture of GPU that lead to no expansion of GPU, the test scores were worse than the test scores on CPU platform.

MIC cards in offload mode did not meet the expected requirements. One of the reasons was the number of processes in offload mode was limited by the number of cores in CPU. Another reason was that offload mode need to consume a lot of time for communication. Due to the limitation of memory, the scale of the problem was not large enough. So that the performance of the MIC could not be completely released. Similar to GPU, it was the special structure that led to decline of scalability. Thus, the results did not meet our expectations.

5.9 Limitations of our platforms

Due to the repeated experiments and the use of the three platforms in the experiment, we have a sufficient understanding of the three platforms. The advantages and disadvantages of three testing platform are gradually reflected.

In HPCG test, the performance of CPU were well beyond our imagination. However, considering from the angle of overall competition, although CPU platform obtained highest scores, this is because what HPCG pay attention to is the collocation of core and memory. The high scalability of CPU can make us modify the hardware configuration to meet the requirement of the HPCG test as much as possible. When facing with other tests, especially for the tests whose requirement for the number of cores and memory is low, the CPU platform will not play a very good performance.

And its low power consumption ratio and low ability of floating point arithmetic will be prominent.

Although the GPU platform does not play a proper level, but it still get a good result. The GPU platform has the highest theoretical floating-point arithmetic ability, the power consumption ratio is also ranked first, and it also has strong performance. But, due to the hardware architecture of accelerator card and the large workload for code transplantation in CUDA environment, it is difficult for us to put our GPU platform as the main platform.

The performance of the MIC platform is relatively good. Subject to CPU core number, offload model can't play its due performance. And in HPCG test, it also face the problem that hardware accelerator card is fixed and can not be extended. But, it have better development in other applications. What we should think highly of is its high compatibility. Codes only need small modifications that they will be able to run on the mic. And it support the offload blend modes. Besides, it can make full use of cluster performance,not wasting any one watt of power. Therefore, the super competition platform we preferred should be the cluster platform based on CPU and MIC accelerator card.

5.10 Predict test results of proposals based on Inspur server NF5280M4

5.10.1 Prediction results of proposal A (INSPUR-CPU)

By the experience we obtained, we have predicted the optima test result in proposal A. Here is the performance comparison between different proposals .

Table 5.6 : Comparison between proposal A and platform A

	Proposal A（CPU）	Testing platform A（CPU）
The theory of floating point arithmetic capability	2496Gflops	1996.8Gflops
Power consumption	2825w	2532.5w
Power consumption ratio	0.8835Gflops/w	0.7884Gflops/w
The number of cores of a single node	24	16
Memory of a single node	64GB	32GB
Memory per core	2.667GB/core	2GB/core

By contrast, for the aspect of the theory of floating point arithmetic capability, proposal A is better than testing platform A. But the gap of memory per core between two platform is small. The prediction results are as follows:

Table 5.7: Prediction results of proposal A

	Testing platform A（CPU）		Proposal A（CPU）
The optimal dimension	16*16*16	**The prediction of optimal dimension**	16*16*16
The optimal time	400	**The prediction of optimal time**	400
The optimal result	3.553% 70.9577Gflops	**The prediction of optimal result**	4% 99.84Gflops

5.10.2 Prediction results of proposal B (INSPUR-GPU)

By the experience we obtained, we have predicted the optima test result in proposal B. Here is the performance comparison between different proposals .

Table 5.8: Comparison between proposal B and platform B

	Proposal B（GPU）	Testing platform B（GPU）
The theory of floating point arithmetic capability	9828Gflops	6027.2Gflops
Power consumption	2902w	1465w
Power consumption ratio	3.386Gflops/w	4.114Gflops/w
The number of GPU accelerator card	6	4
Single card memory	12GB	12GB

Only from the HPCG test with CUDA, proposal B only have two more accelerator card. The prediction results are as follows:

Table 5.9: Prediction results of proposal B

	Testing platform B（GPU）		Proposal B（GPU）
The optimal dimension	256*256*256	**The prediction of optimal dimension**	256*256*256
The optimal time	500	**The prediction of optimal time**	500
The optimal result	1.96% 112.136Gflops	**The prediction of optimal result**	2% 171.6Gflops/s

5.10.3 Prediction results of proposal C (INSPUR-MIC)

By the experience we obtained, we have predicted the optima test result in proposal C. Here is the performance comparison between different proposals .

Table 5.10: Comparison between proposal C and platform C

	Proposal C（MIC）	Testing platform C（MIC）
The theory of floating point arithmetic capability	7996.8Gflops	5139.2Gflops
Power consumption	2759.5w	1725w
Power consumption ratio	2.8979Gflops/w	2.979Gflops/w
The number of MIC accelerator card	6	4
Single card memory	16GB	16GB

From the experimental experience, the prediction results in offload mode were as follows:

Table 5.11: Prediction results of proposal C

	Testing platform C（MIC）		Proposal C（MIC）
The optimal dimension	24*24*24	**The prediction of optimal dimension**	24*24*24
The optimal time	60	**The prediction of optimal time**	60
The optimal result	0.36% 4.46Gflops	**The prediction of optimal result**	0.4% 28.992Gflops

According to the test results, the performance of the MIC platform is excellent. But it did not show good performance in HPCG test. This is because the MIC card operation procedures were limited by the symmetrical structure and memory size. We believe that the MIC card will have excellent performance in other applications.

6 MASNUM_WAM Test

6.1 Introduction

The MASNUM model is the third generation ocean numerical model developed in China. In recent years, the model has been widely used in China's marine disaster prevention, research and other aspects of the wave model. The predecessor of this model is LAGFD-WAM wave model and the MASNUM model has constant improvement and development in the spherical coordinate system under LAGFD-WAM model. MASNUM mode mainly use Fortran 70/90, including three main parts ,that is the initialization mode, the numerical simulation, the output of results.

In physics, the model includes wave propagation function propagat and the source function Implsch. The main calculation process of this model only consider the vertical direction. The adjacent calculation point and the calculation process in horizontal direction are not taken into

consideration. So, the subdivision in the grid of the MASNUM model from the horizontal direction is more appropriate. Through the statistic research on the process of wave breaking, it improves the adaptability of models in different sea conditions.

6.2 Installation

(1) The installation of NETCDF

According to the requirements of the model, the version of the netcdf-3.6.2 is recommended. Due to the version is too old that we cannot get this version, we choose the version of netcdf-3.6.3 which is the closest to netcdf-3.6.2 as grid data processing software. During the installation, the default of the netcdf-3.6.3 installation path placed Inc and lib base file in the public directory, which affected the program's reads. Therefore, it is important to set the installation path of the Inc and the Lib base in installation path of the netcdf during the installation..

(2) The installation of MASNUM_WAM

During the installation, the makefile of MASNUM_WAM need to be modified. Here are parameters in makefile:

Table 6.1: Parameters in makefile

Name	Meaning
F77	Select the FORTRAN compiler
FFLAGS	Optimization options of Fortran compiler
NETCDF_PATH	Installation path of Netcdf
LIBS	Installation path of lib base in Netcdf
INC_PATH	Installation path of include file in Netchd

6.3 Optimization analysis

We found that MASNUM_WAM compiled files do not add compiler options, so we selected the compiler options according to the characteristics of the MASNUM_WAM program. The options are as follows:

- -O3 : Using the optimization combination of the O3 level

- -xHost : Improve the priority of the operation to allocate more resources for the program
- -align array32byte : Choose byte aligned size. Considered that the double precision floating-point number was used in budget, we used 32 byte to improve memory access efficiency

- -no-prec-div : The division operation are changed into multiplied by the reciprocal operation, which can significantly enhance the efficiency of matrix operation..

6.4 Experimental environment

As the existing MASNUM_WAM code only has MPI parallel, this part of the experiment is only tested on platform A (OUR-CPU) given in chapter 4. See chapter 4 for details.

6.5 Exp1 test

6.5.1 Test results before optimization

After the compilation, we tested exp1. We performed the experiment by by modifying the exp1_run.csh script file. In the script, setting calculation period from January 1 to February 2 and setting the step length of each step is 7.5 minutes, we set the number of processes which meet the experimental requirements according to the size of the grid. Here are the results:

Table 6.2: results:

The number of processes	Time	Speed-up ratio
1	1513.5	1.00
2	735	2.06
4	349.5	4.33
8	202	7.49
16	110	13.76
24	77.5	19.53
32	62	24.41
64	53.5	28.29
80	52.5	28.83
96	57	26.55
104	49	30.89

Changes which test time changes with the number of processes were showed in Figure 6.1.

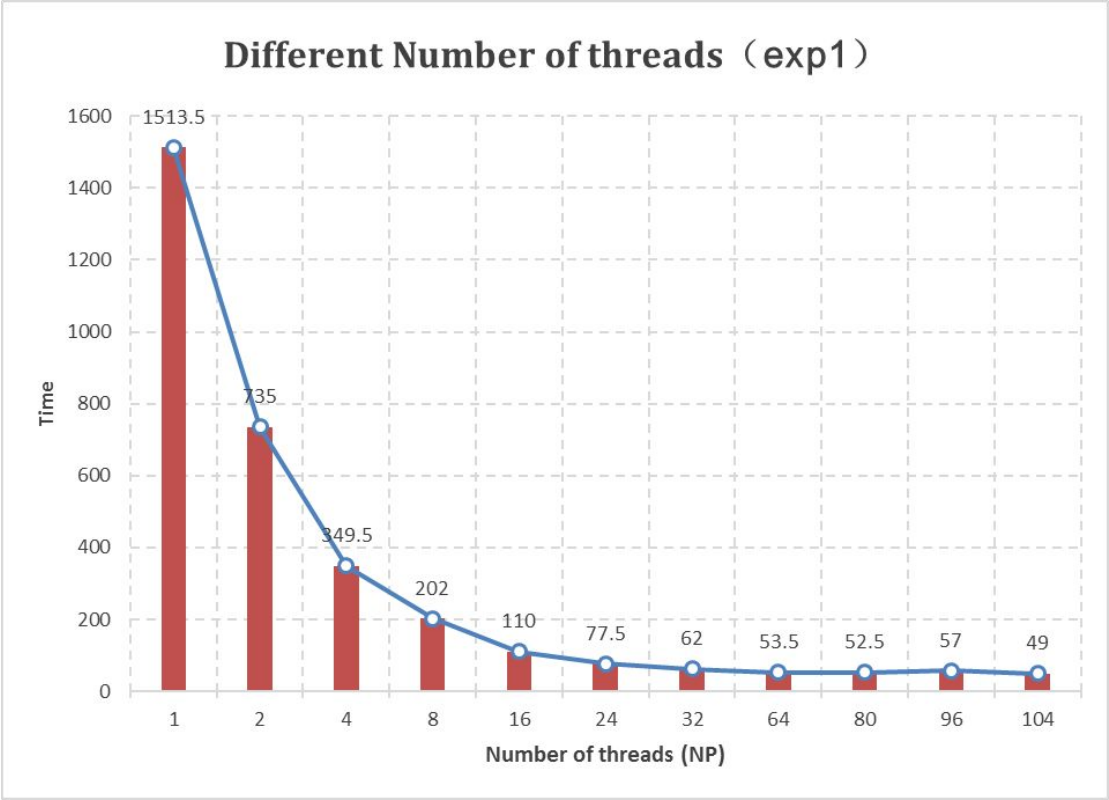


Figure 6.1 : Diagram between test time and the number of processes (NP)

Seen from the diagram, in few processes, with the increase of the number of the processes, the testing time reduce, but decrement also reduce. When the number of processes is more a certain number, with the increase of the number, the testing time increase. This is because the computation time and communication time are close in magnitude, although the increase of the number of processes can reduce testing time, the communication overhead also increases, resulting in the fact that continued to increase the number of processes is unable to obtain the proper optimization results.

The speed-up ratios of each process are as follows:

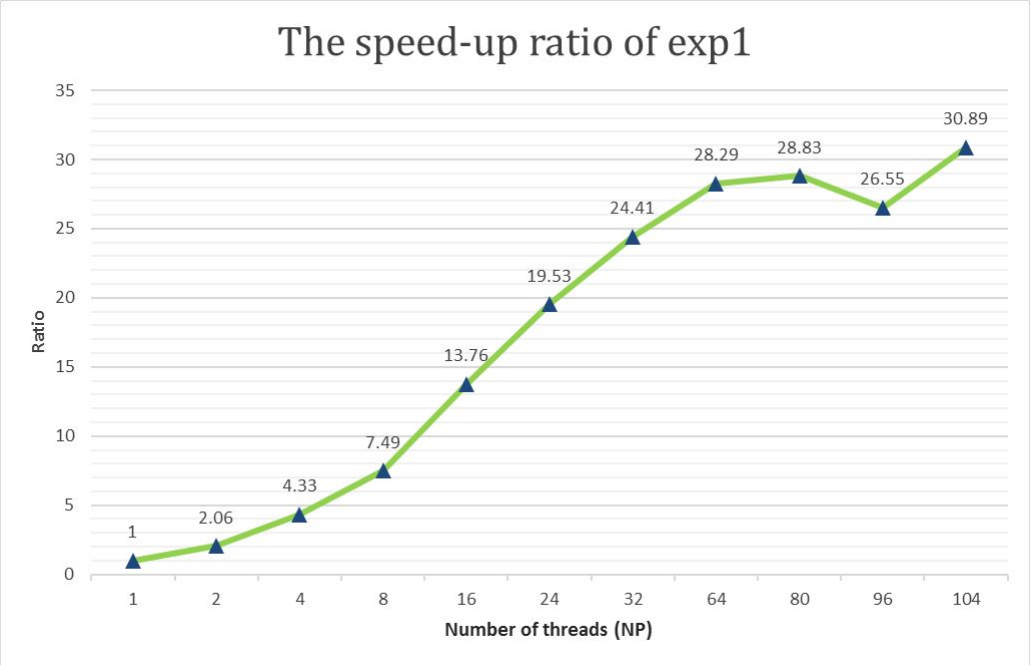


Figure 6.2 : The speed-up ratio

From the figure, it can be seen that with the increase of the number of processes, the speed-up ratio decrease, which caused by communication overhead.

Here is the enlarged view from 32 to 96 processes.

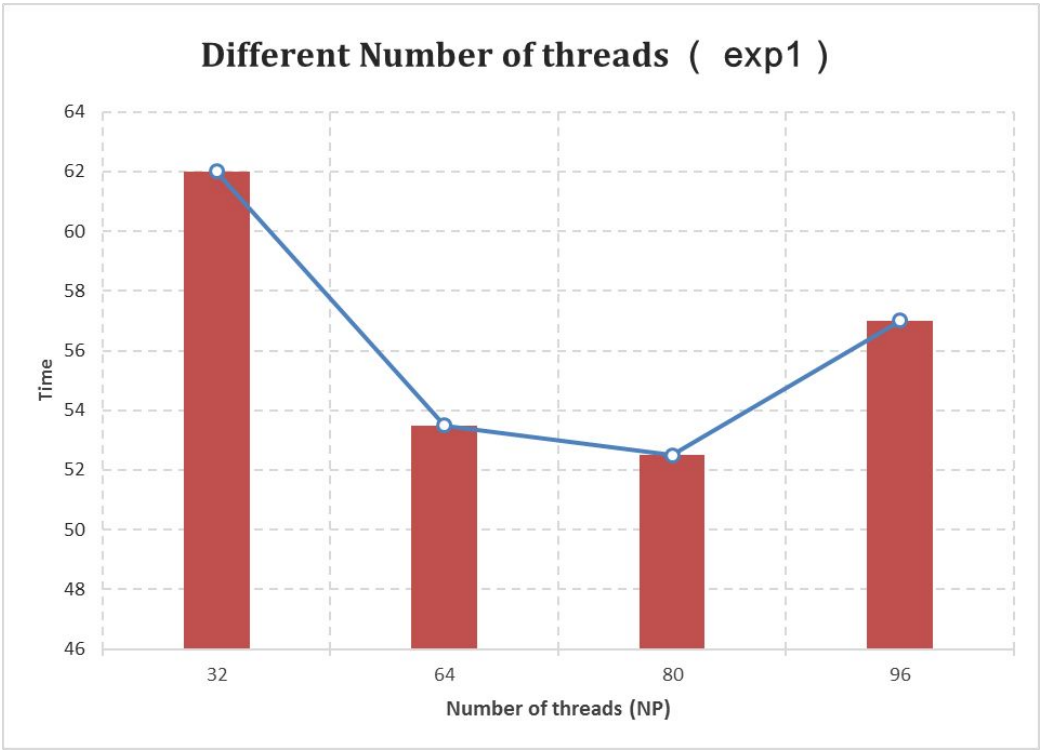


Figure 6.3: enlarged view

From the figure, we could obviously find that the curve like a tub, which illustrated that when the number of processes are above a certain number, the magnitude of the communication time is greater than the computation time. At this time, the optimization should be focused on communication cost.

6.5.2 Test results and analysis after optimization

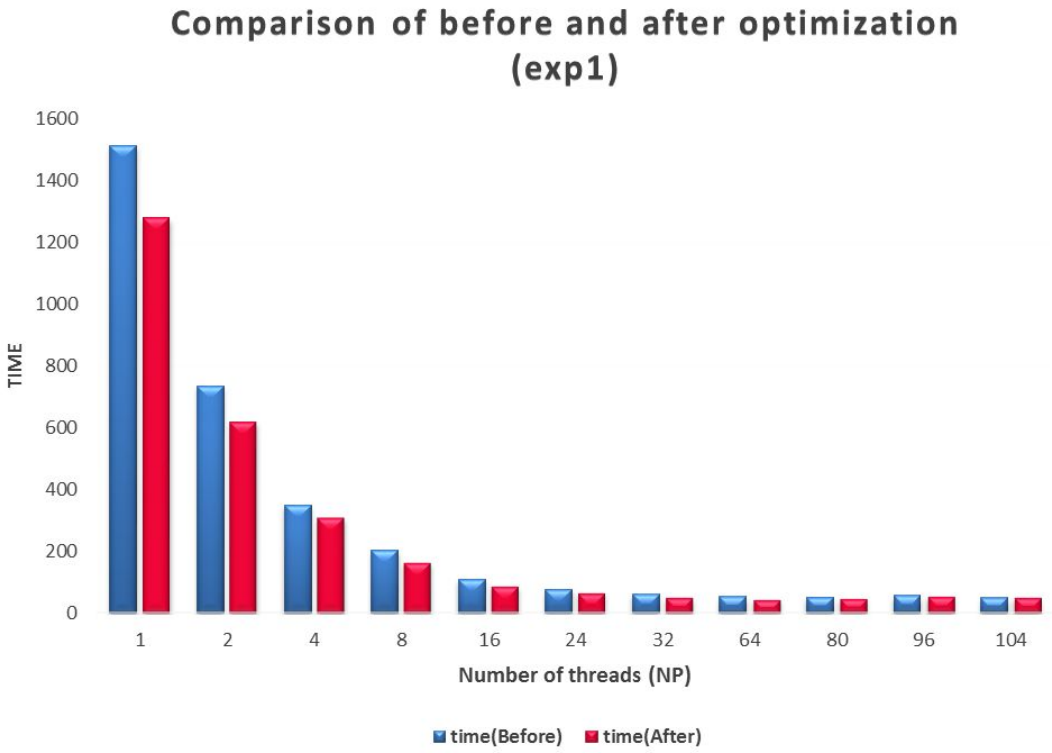


Figure 6.4: Comparison

According to the graph, in a few number of processes, the compiler optimization efficiency for ascension is stable and obvious. But with the increase of the number of processes, compiler optimization effect is not obvious. This is because with the increase in the number of processes, the computation time is small compared to the communication time. Thus, there is no comparison between the optimization efficiency and communication overhead, resulting in optimization effect is not obvious.

Here are the optimized data.(The shortest time is mark in red)

Table 6.3: Optimized data

The number of processes	Time before optimization	Time after optimization
1	1513.5	1279.5
2	735	620.5
4	349.5	312
8	202	166
16	110	90
24	77.5	65.5
32	62	52
64	58	49
80	53.5	**47**
96	52.5	50
104	57	57.5

6.5.3 Summary

According to the test, we found that with the increase of the number of processes, each step of the computation time decreased first and then increased. This is because when the number of processes is small, the computation time of each process is long while the communication time is short. Thus, with the increase of the number of processes, the computation time is reduced and the calculation efficiency increased. However, with the number of the processes continues to increase, the calculation task to each process becomes small. So, the computation time of each process become short while the communication time become long. The frequently communication affect the performance of the program so that the calculation efficiency decreased.

After the compiler optimization, when the number of the processes is small, the computational efficiency has significantly improved, the maximum of about 1.2 times. But the compiler optimization can not solve the problem of communication between nodes. So, in the case of a large number of processes, the effect of compiler optimization performance is not satisfactory.

6.6 Exp2 test

6.6.1 Test results before optimization

After exp1 test, according to the experience we obtained from exp1, we modified the exp2_run.csh script file and preform exp2 test. Here are benchmark results.

Table 6.4: Benchmark time

The number of processes	time	Speed-up ratio
2	209	1.00
4	100.3	2.08
8	52	4.02
16	31.2	6.70
24	24.9	8.39
32	21.8	9.59
64	22	9.5
80	24.4	8.57
96	27.1	7.71
104	28.8	7.26

Changes which test time changes with the number of processes were showed in Figure 6.5.

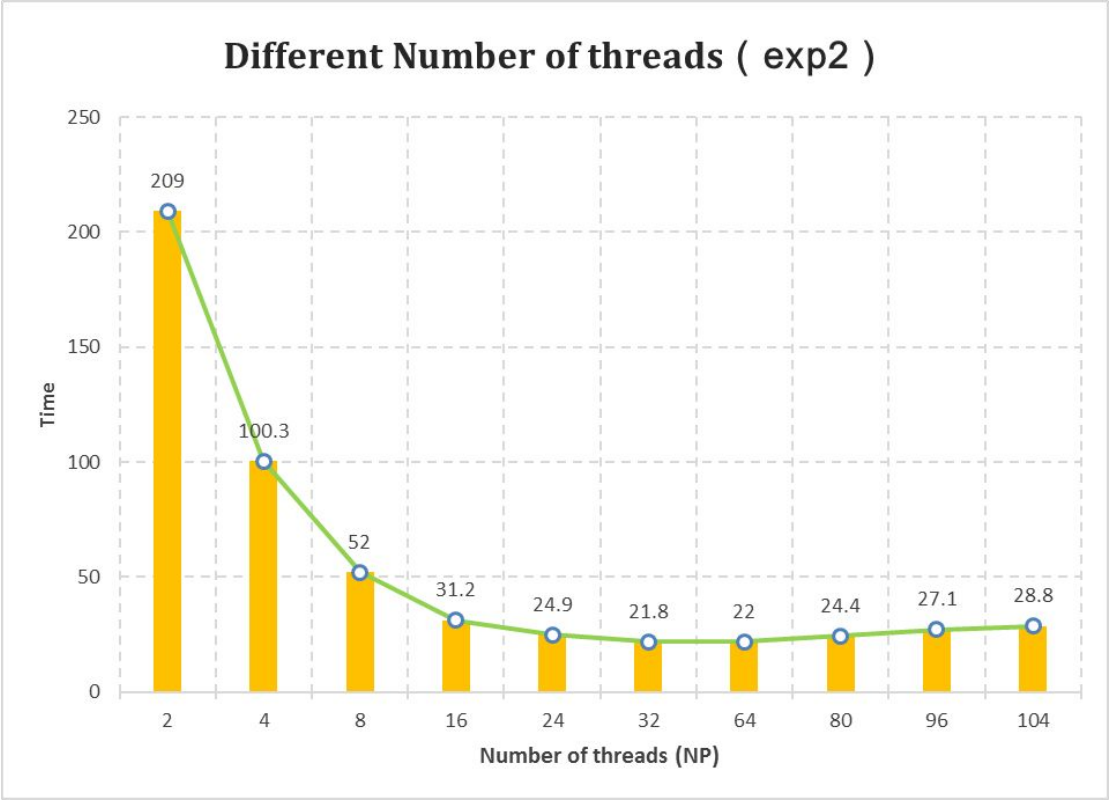


Figure 6.5 : Diagram between test time and the number of processes

The speed-up ratios of each process are as follows:

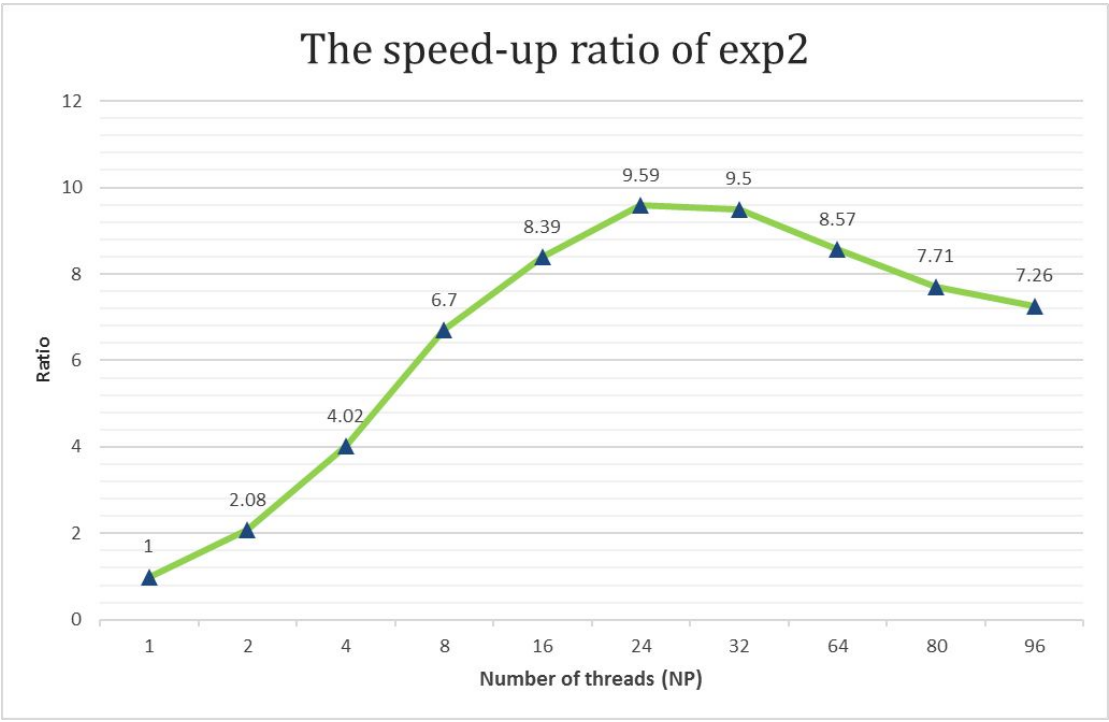


Figure 6.6: the speed-up ratio

Similar to exp1, the curve of the speed-up ratio of exp2 also like a tub, and more obviously. As the resolution of exp2 is low and the amount of calculation is smaller than exp1, the computation time and communication overhead is more comparable.

6.6.2 Test results after optimization

We take the same optimization options in exp1. The optimization results are as follows

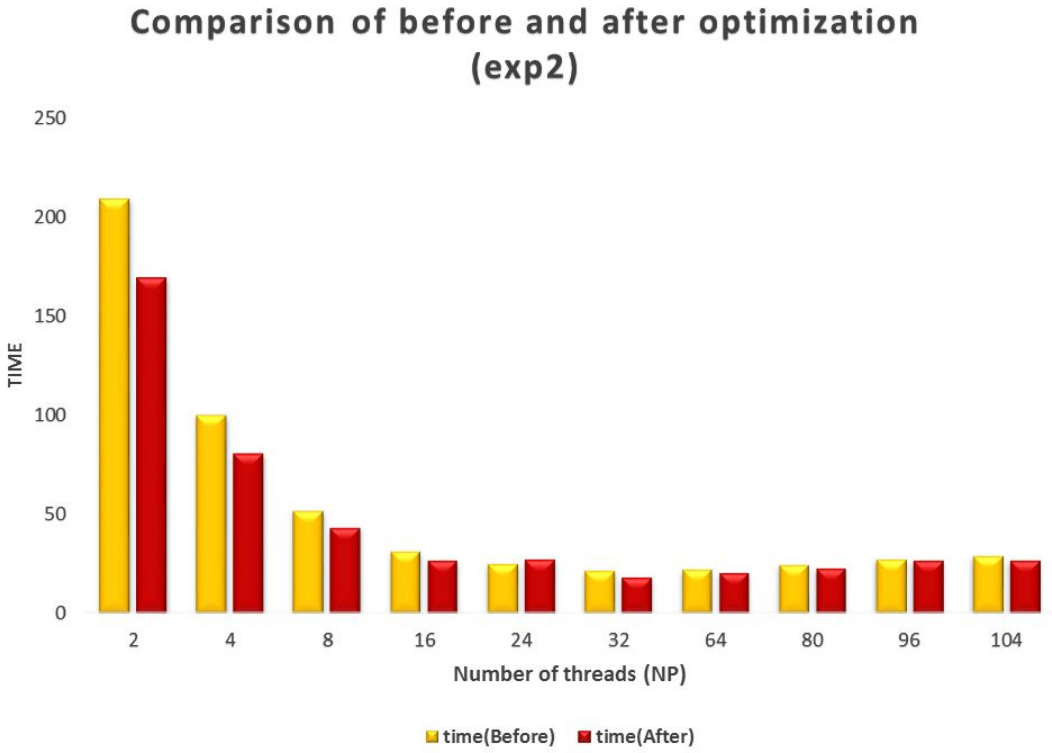


Figure 6.7 : Comparison

Similar to exp1, due to the impact of communication overhead, with the increase of the number of the processes, the communication overhead gradually occupy the dominant position, which no longer has the validity of optimization calculation time.

Here are the optimized data.(The shortest time is mark in red)

Table 6.5: Optimized data

The number of the processes	Time before optimization	Time after optimization
2	209	169.5
4	100.3	80.6
8	52	43.2
16	31.2	26.6
24	24.9	27.5
32	21.8	**18.4**
64	22	20.5
80	24.4	23
96	27.1	26.5
104	28.8	27

6.6.3 Summary

Similar to exp1, in exp2 test, with the increase of the number of progresses, each step of the computation time still showed a trend of decreased first and then increased. What is different is that the optimal number of threads in exp2 is less than exp1. The reason is that the grain size of calculation in exp2 is thick and resolution is relatively low. Due to the long span of time, the step count reduced. Thus, the calculation of each step in exp2 is smaller than exp1, about 13.8% of each step of exp1. Because of the small quantity of calculation and large communication overhead compared to the calculation, the improved performance which is according to the compiler optimization would not be significantly out, resulting in compiler optimization did not achieve the expected acceleration effect in optimal conditions.

6.7 Conclusion

According to the learning about the mode and two test above, we have a deeper understanding about MASNUM_WAM model. Through the experiment, exp1 and exp2 can be sped up about 28.8 and 19.4 times and acceleration curves all like a tub, which illustrated that we need consider the balance between computation time and communication overhead during the optimization process. Then, exp1 and exp2 can be sped up about 1.18 times and 1.07 times through compiler optimization and the acceleration curves were monotonically decreasing, which illustrated that computation time and communication overhead is comparable during the optimization process and the efficiency brought by compiler optimization was gradually not obvious. Using the two kinds of the optimization at the same time, exp1 and exp2 can be sped up about 32.2 times and 22.72 times. The effect of hybrid optimization is better than the single optimization method when magnitudes of computation calculation and communication overhead are close to each other. Besides, hybrid optimization can provide higher speed-up ratio. So, the choice of hybrid acceleration is the most effective means of optimization. The optimized results were listed below.

Table 6.6 : Optimized results

	Parallel	**Compiler optimization**	**The product of optimized performance**	**Hybrid optimization**
Exp1	28.8	1.18	33.984	32.2
Exp2	19.4	1.07	20.758	22.72

6.8 Validation

6.8.1 Validation of exp1

According to the requirements of the subject, we modify the makefile parameters in the exp1 folder under the validation folder and produce test program by commend “make”. The makefile document for validation procedures only need choose compiler and add the netcdf path. Then, test

program ‘compare_exp1’ was generated. Compared generated file “pac_ncep_wav_20090228.nc” with system’s standard file “pac_ncep_wav_20090228_standard”, the program shows “success”, that is our results are correct.The picture of validation consequence is given in Figure 6.8.

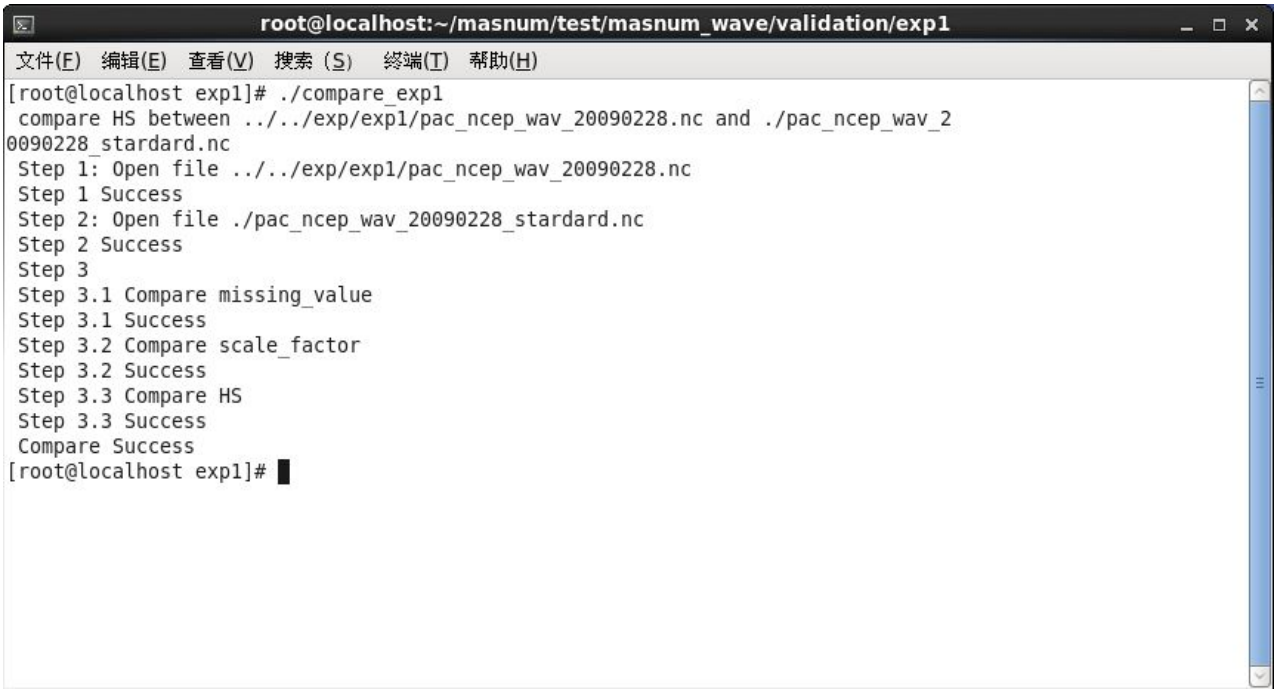


Figure 6.8 : the validation consequence of exp1

6.8.2 Validation of exp2

According to the requirements of the subject, we modify the makefile parameters in the exp2 folder under the validation folder and produce test program by commend “make”. The makefile document for validation procedures only need choose compiler and add the netcdf path. Then, test program ‘compare_exp2’ was generated. Compared “global_ncep_wav_20090630.nc” with “global_ncep_wav_20090630_standard”, the program shows “success”, that is our results are correct.The picture of validation consequence is given in Figure 6.9.

Figure 6.9 : the validation consequence of exp2

6.9 Our suggestions for further optimization

Based on our understanding of the model, the bottleneck of current MASNUM_WAM model is frequent file operations. So we put forward two suggestions for further optimization.

1) Avoid file access conflict
   On most cluster platforms, NFS mapping technique is used to share files between nodes. Because of this, in the current MASNUM_WAM model, multi processes read the same input file. Since this mode itself has frequent reading and writing operations, multi access to the same file will cause access conflict which leads to waiting. Also, it waste precious computation time. In our view, we can let each node has a copy of data, than MPI processes can read the file in local node. Thus, file access conflict can be avoided.

2) Carry file operation and calculation in parallel
   Divide the processes of each node into two group. In the first group, there is only own processes, which focuses on file reading and writing. The other processes belong to the second group, focus on computations. Than by making good use of the non-blocking communication of MPI, we can achieve the parallelization between file operations and computations.

Illustration of our suggestions is given in Figure 6.10

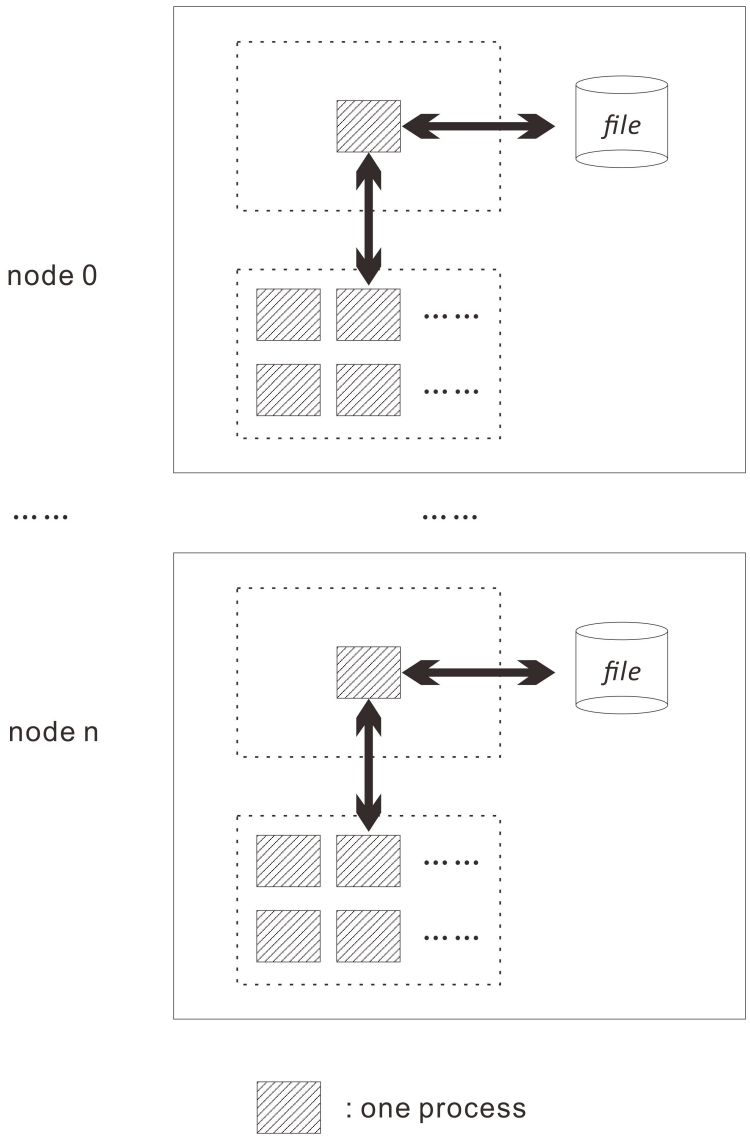


Figure 6.10 : Illustration of our suggestions

7 Optimization of the DNN program

7.1 Introduction

Deep Neural Network (DNN) is a deep learning algorithm, which modeled loosely after the human brain and are designed to recognize patterns ,it has been successfully applied in many research domains, such as speech recognition and image recognition.

Figure 7.1 demonstrates the structure of DNN. In Deep Neural Network, each layer of nodes is trains on a distinct set of features based on the previous layer's output. The further you advance into the neural net, the more complex the features your nodes can recognize, since they aggregate and recombine features from the previous layer.

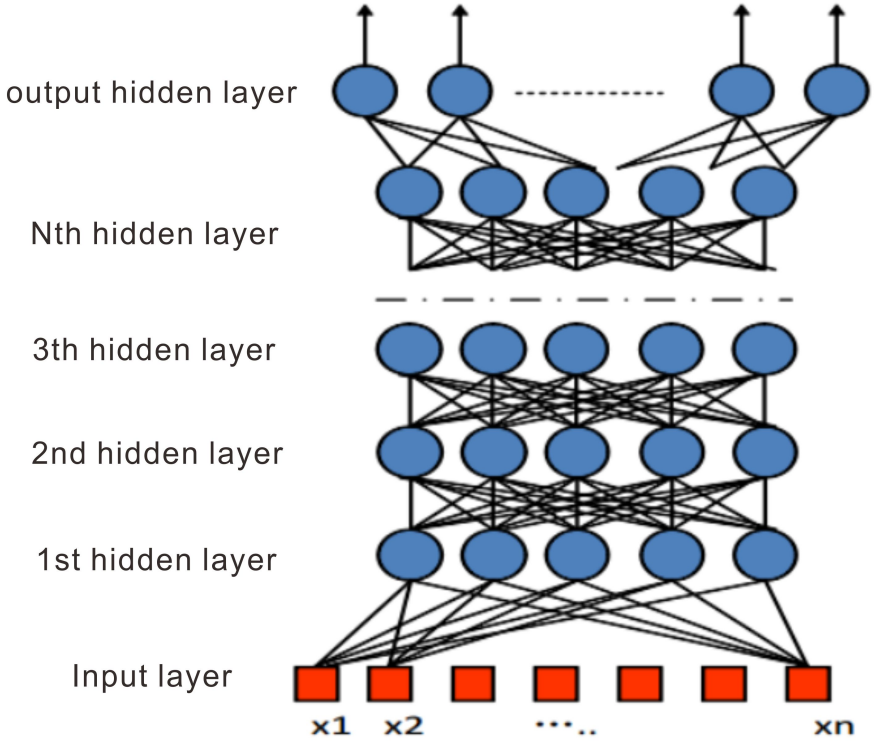


Figure 7.1: Structure of DNN

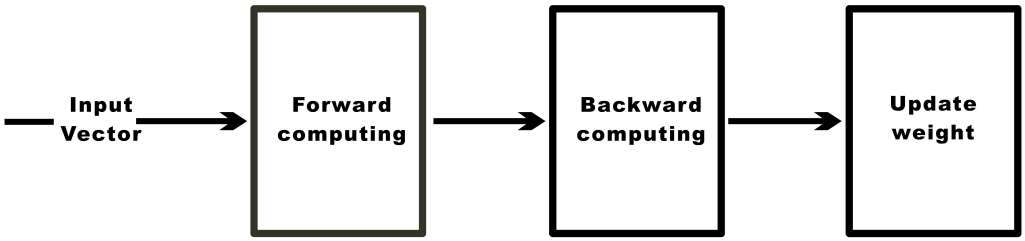


Figure 7.2: Process of DNN training

The calculation step of a real DNN training system for speech recognition is illustrated in Figure 7.2. The calculation is sequentially processed step by step. Within each step, data is processed through different layers. In a forward computing step, the process starts from the first layer and each following layer takes output of its previous layer as input. The computing ends when it reaches the last output layer of the set.On the contrary, a backward calculation is a reverse process of a forward one. It will calculate the error vector of the last output layer and go through all layers to the first one. The update weight process is similar as forward computing. It updates the weight matrix of each layer one by one till the last layer.

7.2 In-depth analysis of the DNN algorithm

7.2.1 Calculation Complexity of DNN

The DNN used in this test consists of eight layers, including an input layer, six hidden layer,and an output layer. The number of nodes of input layer is 429, middle layers is 2048, and the output layer is 8991. The algorithm includes three major parts: forward calculation, error back propagation, and weight update. In the process of training,there will be 3571 bunches go through these three parts of the training.Flow chart of provided program as the Figure 7.3.The pseudo code is shown with Figure 7.4.

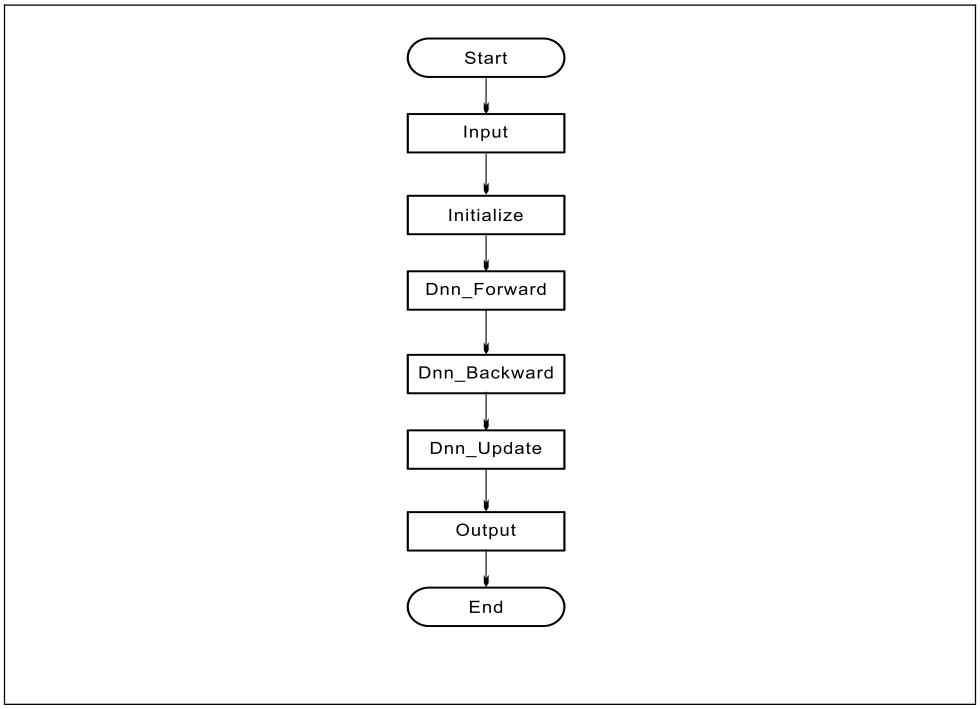


Figure 7.3 Flow chart of DNN algorithm

```
BEGIN
    input and initialize
    for i = 0 to 3571 do
        dnnForward(nodeArg)
            for i = 0 to 7 do
setmatY()       //Initialize the matrix Y
                cblas_sgemm()   //Matrix multiplication
                if (i == numL-1) then
                    softmaxZ(d_Y[i], d_Y[i], numN, numA[i]) //Matrix in_vec,out_vec calculation and update
                else then
                    sigmoidY(d_Y[i], numN, numA[i]) //Update the matrix Y

        dnnBackward(nodeArg)
            errorOutput()    //Matirx calculation and update
            for i = 5 to 0 do
                cblas_sgemm()    //Matrix multiplication
                errorTrans()      //Matrix E update

        dnnUpdate(nodeArg)
            for i = 0 to 7 do
                cblas_sgemm()    //Matrix multiplication
                updateW()    //Matrix W update
                updateB()    //Matrix B update

    output
END
```

Figure 7.4 Pseudo code of DNN algorithm

The pseudo code mentioned above briefly expressed the computation process of DNN algorithm. In the calculation process of forward, backward, and update, the main computation is matrix calculations. Main matrix operation of DNN algorithm are shown in Table 7.1. In original source code, these matrix calculations are realized by function cblas_segmm().

Table 7.1: Main matrix operation of DNN algorithm

Parts	**Operation**
Forward	$C = A * B + C$
Backward	$C = A * B^T$
Update	$C = -0.00075 * A^T * B$

During one train, the amount of multiplication operations of each function is listed in Table 7.2. The total Calculation Complexity of a train is 828.6841 G. No hard to see, function cblas_segmm

occupies most of the calculation.

Table 7.2: Amount of multiplication operations during a train

Parts	The calculation	Function	The calculation
forward	268.8639G	setmatY	0.0203G
		cblas_segmm	268.7890G
		softmaxZ	0.0428G
		sigmoidY	0.0117G
backward	262.9674G	erroroutput	0.0085G
		cblas_segmm	262.9238G
		errorTrans	0.035G
update	296.8527G	cblas_segmm	296.7890G
		updateW	0.0414G
		updateB	0.0222G

7.2.2 Analysis 1: hot-spot under different number of threads for the original program

During the process of hot-spot analysis, as it takes a long queue, hot test by VTUNE is not convenient in the remote server provided by the organizing committee. Therefore, we use our local server. Here are its hardware and software configuration.

Table 7.3: Hardware of our local server

Item Name	Configuration
CPU	intel Xeon E5-2630v3 * 2
Memory	Samsung 8G ECC Registered DDR4 2133Mhz * 8
Hard Drive	160G MLC SSD
Accelerator card	Intel Xeon Phi 7110p*2

Table 7.4: Software of our local server

Name	Version
Operating System	Redhat Enterprise Linux Server release 6.5
Function Library	i_mkl_2015.1.133
MPI	OpenMPI-1.6.5 IMPI-5.0.2.044
The Compiler	ifort-15.0.1 icc-15.0.1
Performace Analysis Software	Vtune_amplifier_e_2015.1.1.38310 inspur TEYE

The hot-spot analysis software we used is VTUNE. The screen-shots are shown below.

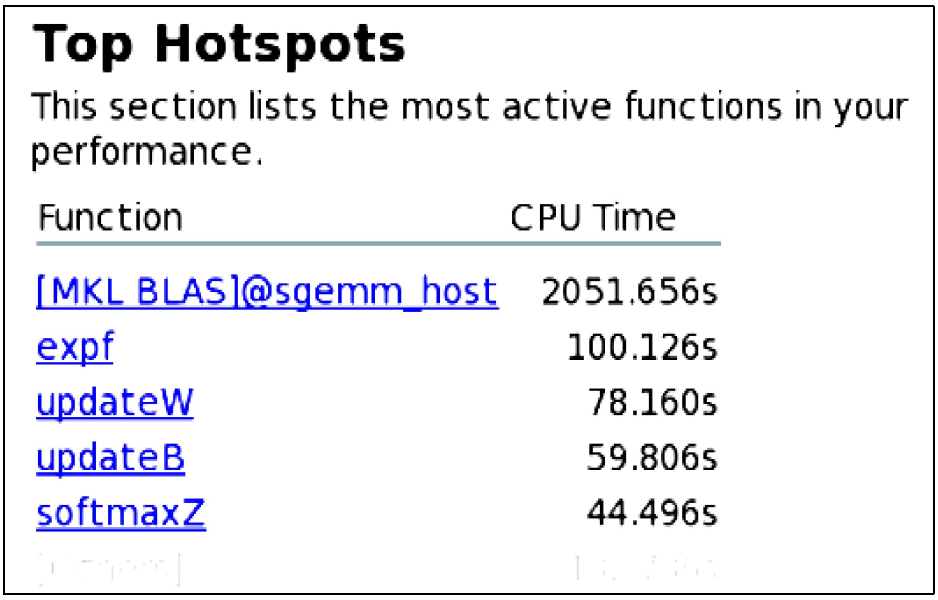


Figure 7.5 : Screen-shot of VTUNE(CPU time of hot function)

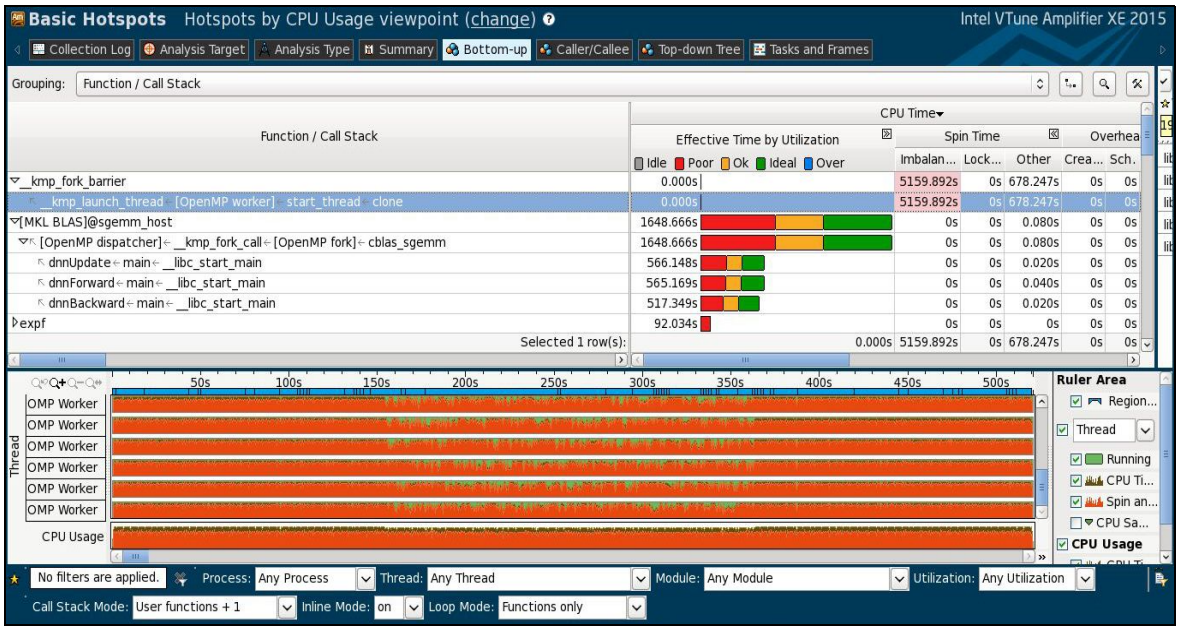


Figure 7.6 : Screen-shot of VTUNE(details of hot function)

The original program calls the cblas_segmm function in MKL Library to realize the main matrix computation. The library itself has done openMP parallel, the number of threads could be set by changing the environment variables MKL_NUM_THREADS , so as to reach a certain degree of acceleration.

Here, we investigate the hot-spot under different number of threads for the original program. The execution time of hot function is listed in Table 7.5.

Table 7.5 : Hot function and execution time under different number of threads

Number of threads	1	2	4	6	8	10	12	14	16
Cblas_sgemm (cpu time)	1020.64	1045.95	1093.04	1210.94	1250.69	1491.24	1443.09	1551.55	1670.66
kmp_fork_barrier (cpu time)	---	304.12	961.03	1679.25	2421.80	3108.82	4084.69	4965.63	5636.50
Expf (cpu time)	74.87	76.47	76.51	85.47	88.92	86.48	93.91	92.96	92.94
updateW (cpu time)	56.26	55.99	55.92	62.00	64.25	61.77	68.79	68.55	68.58
updateB (cpu time)	44.08	44.20	52.35	47.28	48.04	47.07	50.07	49.05	49.03
softmaxZ (cpu time)	36.852	---	---	---	---	---	---	---	---
Others (cpu time)	93.62	143.62	140.44	146.88	156.09	222.37	175.24	226.13	225.82
Total time (elapsed time)	1326.51	836.64	596.71	542.57	509.34	507.01	500.27	512.49	492.68
Speed-up ratio	1	1.58	2.22	2.44	2.60	2.61	2.65	2.59	2.69

According to Table 7.5, when it comes to single-threaded, Cblas_segmm () function occupies about 77% of the total CPU time. When increasing the number of threads, Cblas_segmm () is still the main hot-spot. Function updateW, updateB and softmaxZ are also one of the hot-spot functions.

In addition, the elapsed time of cblas_segmm, kmp_fork_barrier and total time are given in Figure 7.7. According to Figure 7.7, it can been seen that with the increase of the number of threads, although the elapsed time of cblas_segmm decreases, the thread blocking time increase unfortunately. Further more, there are functions(updateW, updateB softmaxZ, etc) which are not parallel in the original program. That’s why the total time does not change obviously when the number of threads is more than eight.

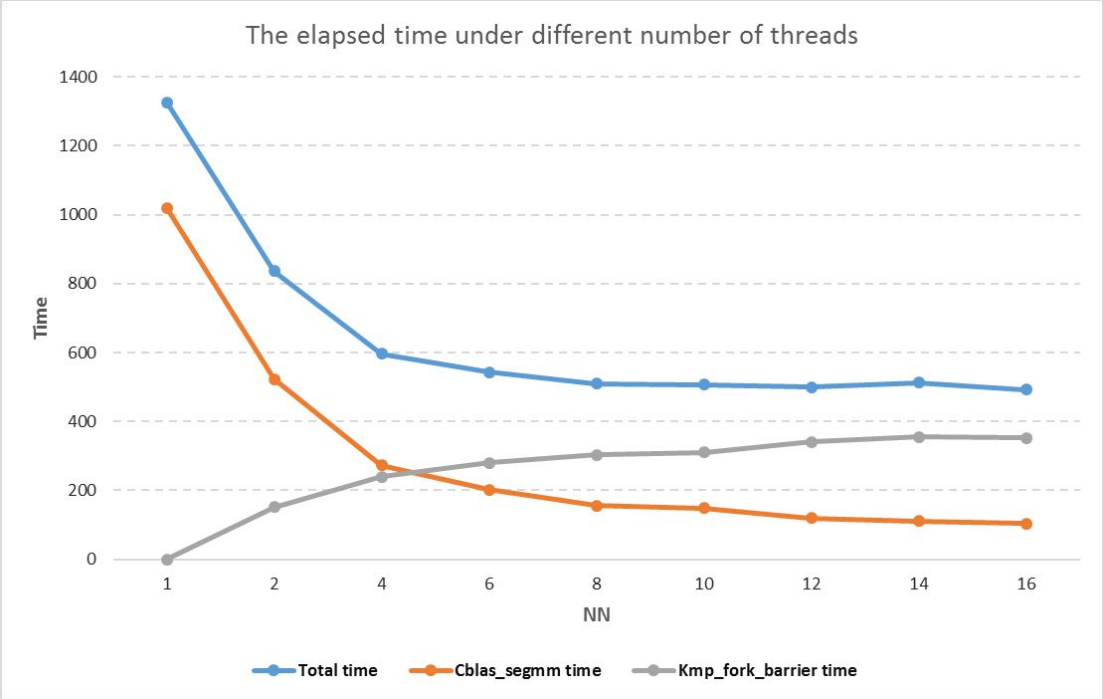


Figure 7.7: the elapsed time under different number of threads

7.2.3 Analysis 2: Whether rewrite function cblas_segmm

From subsection 7.2.2, we know the original program called cblas_segmm. This function has been paralleled at multi-thread level. We also found in subsection 7.2.2 , that the parallel result is not good when the number of threads is more than 8. If you want to further improve the parallel effect of cblas_segmm, it is necessary to rewrite this function.

The definition of cblas_segmm function is giving bellowing.

```
void cblas_sgemm(const enum CBLAS_ORDER Order, const enum CBLAS_TRANSPOSE TransA,
   const enum CBLAS_TRANSPOSE TransB, const int M, const int N,
   const int K, const float alpha, const float  *A,
                  const int lda, const float  *B, const int ldb,
                  const float beta, float  *C, const int ldc)
```

Figure 7.8: The definition of cblas_segmm function

Its function is computing

$$C = alpha * op(A) * op(B) + beta * C$$

We tried to realize cblas_segmm function through our own programming, named **our_segmm()**. In **our_segmm()**, block matrix multiplication is adopted. Compared with the general matrix multiplication algorithm, the block algorithm can increase the hit rate of Cache and reduce the probability of data read from the main memory, so as to speeding up the computation speed. See Table 6 for the comparison between general matrix multiplication and block matrix multiplication. It can be seen from Table 7.6 that, the efficiency of block matrix multiplication is about 2 to 3

times of the general matrix multiplication.

Table 7.6: comparison between general matrix multiplication and block matrix multiplication

Time(s)	Number of threads									
	1	2	4	6	8	10	12	14	16	32
general matrix multiplication	325.28	246.99	124.21	97.25	87.60	82.81	77.26	73.79	73.80	60.22
Block matrix multiplication	223.3	112.42	58.64	43.92	32.52	30.36	28.03	27.27	26.75	23.37

The source code of **our_segmm()** is given with Figure 7.9.

```
void our_segmm(float *X, float *Y, float *Z,int M, int N, int K, float alpha, float beta)
{
    int i,j,l;
        for(int jj = 0; jj < N; jj += BLOCK)       //BLOCK is the size of block
          for(int kk = 0; kk < K; kk += BLOCK)
              for(i = 0; i < M; i++)
                  for(j = jj; j < min(jj+BLOCK,N); j++)
                      for(l = kk; l < min(kk + BLOCK,K); l++)
                          Z[i*N + j] = X[i * K + l] * Y[l * N + j] * alpha + beta * Z[i *N + j];
}
```

Figure 7.9: Code of **our_segmm**

Now, we call **our_segmm** in parallel like Figure 7.10.

```
void matrix_omp(float *X, float *Y, float *Z,int M, int N, int K, float alpha, float beta)
{
    int n;    //threads num
    int nums[MAX];    //matrix rows of each thread
    int the_num[MAX];     //the starting position of each matrix

    nums[0] = M / n;
    the_num[0] = 0;
    for(int i =0 ; i < n; i++)
    {
        num[i] = M / n;
        the_num[i] = the_num[i-1] + num[i];
    }
    num[n-1] += M / n;
```

```
    float *tempA = X;
    float *tempC = Z;
    #pragma omp parallel for num_threads(n)
    for(int j = 0; j < n; j++)
    {
        tempA = X + the_num[i] * K;
        tempC = Z + the_num[i] * N;
        our_segmm(tempA, Y, tempC, M, N, K, alpha, beta);
    }
}
```

Figure 7.10: Call our_segmm() in parallel

Comparison of **our_segmm** and **cblas_segmm** is given in Table 7.7. Note that the time is the execution time of forward of a train data.

Table 7.7: Comparison of **our_segmm** and **cblas_segmm**

Time(s)	**Number of threads**								
	1	2	4	6	8	10	12	14	16
our_segmm	223.3	112.43	58.64	43.93	32.53	30.36	28.03	27.28	26.75
cblas_segmm	1.07	0.55	0.37	0.25	0.22	0.22	0.24	0.25	0.28

From Table 7.7, we know that the efficiency of cblas_segmm () function is much better than **our_segmm**. The possible reasons for high efficiency of MKL are as follows:

(1) Using the program locality principle and the characteristics of the storage system to minimize the time which it takes to access data. Compared to algorithm 2 - matrix_block, it did better.

(2) The MKL library aiming at the most frequently used core operation in program, such as small matrix multiplication and matrix copy, do the comprehensive development in the underlying according to the characteristics of the hardware targeted of different processor.

So it is adjudged wise to use cblas_segmm () in the following optimization.

7.2.4 Conclusion of analysis 1 and 2

By the above experimental results and analysis, we can draw the following conclusions:

(1) The original program calls the MKL Cblas_segmm which have parallel ability at the thread level. However, the parallel result is not good when the number of threads is more than 16. The shortest execution time is 492.68 s. Compared with a single thread, speed ratio is 2.695. In the case of the shortest execution time, that is when the threads number is 12, the hot-spot functions are Cblas_sgemm, kmp_fork_barrier, updateW and updateB. With the increase of the number of threads, thread blocking time of Cblas_segmm increases, which influence the parallel effect.

(2) The function updateW updateB, softmaxZ do not make any parallel processing in the original program.

(3) The efficiency of Cblas_segmm in MKL is nearly 200 times higher than our_segmm, so it is adjudged wise to use cblas_segmm () in the following optimization.

7.3 Optimization scheme

According to the experimental results and conclusions in 7.2, we tried to further optimize the program as the following three steps:

Step 1: Compiler optimization
Step 2: Parallelize function updateW, updateB and softmaxZ
Step 3: Run cblas_segmm function in parallel

7.3.1 Compiler optimization

1. The choice of compiler and compiler options
The use of compiler options can improve the running efficiency of the code and the efficiency of memory.

-O3 :
Optimization level of O3 is to open several low security options based on O2. The optimization option has enabled the vast majority of security in O2. The added optimization options are:

- -finline-functions : Allows the compiler to choose some simple functions to be opened at the place where they are called, relatively safe, suitable for server level CPU architecture whose level two cache is large ;

- -funswitch-loop : Put the variable which doesn't change its value out of loop body.a it can significantly reduce reduce the calculation of loop body.

- -fgcse-after-reload : In order to remove excess overflow, perform an additional elimination step after the heavy load. It can improve the memory efficiency.

The O3 option is common in compiler optimization for the reason that the security of O3 options

are high in optimization. Vector optimization is the optimization method which has been proved very effective, especially for matrix operation, vectorization has great advantage.
In the Makefile, the original compiler options are as follows:

Table 7.8: The original compiler options

Name	The value of the variable
GCC	?=g++
CCFLAGS	:= -g -Wall -fPIC

Here are the compiler options after modification:

Table 7.9: The compiler options after modification

Name	The value of the variable
GCC	?= icpc
CCFLAGS	:= -g -Wall -fPIC -vec-report=1 -openmp -O3

2. Vectorization

Intel compiler support vectorization. The vectorization is a method of using vector processing unit to batch computing. The loop calculation part can use the extension instruction set to speed up the calculation speed. In most case, we usually adopt parallel outer loop, the inner loop adopts the Vectorization. But if the outer loops are less likely to affect the degree of parallelism, we need to consider the trade-off between parallelism and vectorization. Having been analyzed above, as for updateB, updateW and other functions which have no internal data dependence and the outer loop data volume lowest is 429 while the maximum is 8991, the vectorization should get a satisfactory effect. The optimal solution as shown in the following Figure 7.11:

```
int setmatY(float *Y, float *B, int row, int col)
{
    int idx;
    for(int i=0; i<row; i++)
    {
        #pragma omp parallel for schedule(guided)
        for(int j=0; j<col; j++)
        {
            idx = i*col + j;
            Y[idx] = B[j];
        }
    }
    return 0;
}
```

Figure 7.11: Optimal solution of setmatY

From the results of experiment 1, the optimization vectorization we do here does not work, for the reason that the compiler optimization - O3 has already done it.

7.3.2 Parallelize function updateW, updateB and softmaxZ

UpdateW, updateB, softmaxZ functions are also one of the hot-spot functions. After analyzing the code of updateW, updateB, softmaxZ function, it can be found that the operation is assigned within the loop update operation without data dependence and data read-write conflict which is suitable for parallel processing at the thread and process level.

Separating the matrix to block by row, each thread assumes certain rows of the matrix and complete the update of the matrix together. When separating the matrix, the number of the matrix rows is not necessarily the multiples of the number of threads. Therefore, in the process of matrix segmentation, in order to achieve load balancing, we should choose the appropriate algorithm to separate the matrix.

Take an example of updateW, we given the original source code and parallelized source code with Figure 7.12 and Figure 7.13 respectively.

```
int updateW(float *W, float *Wdelta, int row, int col)
{
    int idx;
    for(int i=0; i<row; i++)
    {
        for(int j=0; j<col; j++)
        {
            idx = i*col + j;
            W[idx] += Wdelta[idx];
        }
    }
    return 0;
}
```

Figure 7.12: Original source of updateW

```
int updateW(float *W, float *Wdelta, int row, int col)
{
    int idx;
    #pragma omp parallel for schedule(guided)
    for(int i=0; i<row; i++)
    {
        for(int j=0; j<col; j++)
        {
            idx = i*col + j;
            W[idx] += Wdelta[idx];
        }
    }
    return 0;
}
```

Figure 7.13: Parallelized source code of updateW

7.3.3 Run cblas_segmm in parallel

Since it has been confirmed that function cblas_segmm in MKL is more efficient than the function written by us (see subsection 7.2.3 for detail), so we will not change this function (cblas_segmm), that is we will not do any parallelization of this function itself. But, we will separate the matrix into several little matrix and run cblas_segmm in parallel. Figure 7.14 is the sketch of our parallelization.

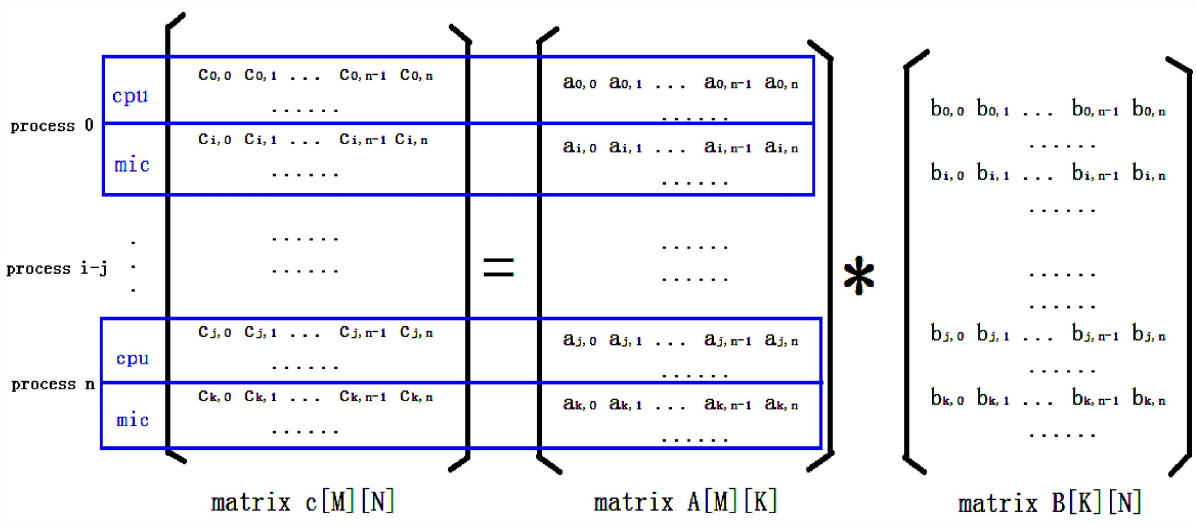


Figure 7.14: mic+mpi +openmp matrix segmentation way

First, we divide the whole matrix in (n+1) parts, where (n+1) is the number of processes. Each process only deal with the corresponding part . So cblas_segmm can be called independently in each process. Second, we divide the sub-matrix in each process into two parts , one for CPU and one for MIC. So cblas_segmm can be called independently in CPU and MIC now. After the completion of the calculation, the data would be returned back to CPU from MIC. Thus, form a complete matrix calculation results.**Kernel codes were shown in Appendix B: The optimal kernel codes**

7.4 Experiment results

7.4.1 Experiment environment

Table 7.10: Configuration of computing node of CPU+MIC cluster

Item	Name	Component Name	Model
Server	Inspur NF5280M4*1	CPU	Intel Xeon E5-2680v3 *2,2.5Ghz,12 cores
		Memory	16G *8,DDR3,2133Mhz
		Hard disk	1T SATA *1
		Accelerator card	Intel Xeon PHI-31S1P (57 cores,1.1GHz, 1003GFlops,8GB GDDR5 Memory) *1

7.4.2 Results of optimization step 1 : compiler optimization

First, we give the experiment results of optimization step 1, that is compiler optimization. In the Makefile, the original compiler options are as follow:

Table 7.11: the original compiler options

Name	The value of the variable
GCC	?=g++
CCFLAGS	:= -g -Wall -fPIC

Here are the variables after modification

Table 7.12: the variables after modification

Name	The value of the variable
GCC	?= icpc
CCFLAGS	:= -g -Wall -fPIC -vec-report=1 -openmp -O3

After modifying the Makefile in compiler and compiler options, its experimental results are shown in Table 7.13 :

Table 7.13: optimization results of step 1: Compiler optimization results (Number of threads is 24) (workload1)

Before compiler optimization (Time/s)	After compiler optimization (Time/s)
422.14	150.30

According to the experimental results, the compiler optimization can speed up about 2.8 times.

7.4.3 Results of optimization step 2 : Parallelization of function updateW, updateB and softmaxZ

Now on the base of optimization step1, we give the optimization results of step 2 with following table 7.14.

Table 7.14 optimization results after step 2: Parallelization of function updateW, updateB and softmaxZ (Number of threads is 24) (workload1)

Before (Time/s)	After (Time/s)
150.30	113.53

7.4.4 Results of optimization step 3 : Run cblas_segmm in parallel

On the base of optimization step1 and step2, the results after step 3 is given in Table 7.15.

Table 7.15:optimization results after step 3: Run cblas_segmm in parallel (workload1) (**Time/s**)

100% on CPU	**Half on CPU and half on MIC**	**100% on MIC**
113.53	159.54	277.02

From Table 14, we find the efficiency of step 3 is not good. The computation resource of MIC is not utilized at all. The results are not consistent with that on our local server. See Table 7.16 for the optimization results on our local server. See subsection 7.2.2 for the detail of our local server. The reason maybe that the mic card on remote Inspur server is overload or other unknown reasons.

Table 7.16:Optimization results after step 3 on our local server (workload1)

100% on CPU	**Half on CPU and half on MIC**	**100% on MIC**
248.41	207.36	305.82

7.4.5 Summary

The final results is given in Table 7.17.

Table 7.17: summary of our optimization

	Remote inspur server		**Our local server**	
	Workload1	Workload2	Workload1	Workload2
Before optimization	422.14	1543.14	512.45	1984.00
After step 1: Compiler optimization	150.30	461.47	264.61	799.34
After step 2: Parallelization of function updateW, updateB and softmaxZ	113.53	426.20	242.042	760.87
After step 3: Run cblas_segmm in parallel	130	446.99	220.21	699.13

7.5 Conclusion

As we were required to use only one MIC node in the preliminary round, we did not do parallel MPI. However, our code has been very suitable for parallel MPI. So, if we can enter the final, our program is expected to obtain better results in multiple nodes.

Appendices

Appendix A Optimized results of HPCG Test

HPCG results submission form(CPU)

HPCG	
HPCG version	**HPCG 3.0**
System Spec	**Dual Intel Xeon E5-2630V3@2.4Ghz,32GB memory**
Compute Nodes	**13**
OS	**Redhat Enterprise Linux Server release 6.5**
MPI	**IMPI-5.0.2.044**
Compiler	**icc-15.0.1**
Compiler Flags	**-O3, -ffast-math,-ftree-vectorize, -ftree-vectorizer-verbose=0**
Environment Variables:	**OMP_NUM_THREADS=1 PATH=/opt/intel/impi/5.0.2.044/bin64:/opt/intel/composer_xe_2015/bin**
Turbo (ON/OFF)	**ON**
	Results
Number of nodes	**13**
Number of cores	**208**
HPCG (base or optimized)	**optimized**
HPCG result	**70.9577Gflops**

HPCG results submission form(GPU)

HPCG	
HPCG version	**HPCG 3.0 Binary from NVIDIA Optimized for NVIDIA GPUs (bugfix)**
System Spec	**Dual Intel Xeon E5-2630V3@2.4Ghz,32GB memory,Dual Nvidia K40M**
Compute Nodes	**2**
OS	**Redhat Enterprise Linux Server release 6.5**
MPI	OpenMPI-1.6.5
Compiler	**GCC-4.4.7**
Compiler Flags	**-O3, -ffast-math,-ftree-vectorize, -ftree-vectorizer-verbose=0**
Environment Variables:	**OMP_NUM_THREADS=8 LD_LIBRARY_PATH=/usr/local/cuda-6.5/lib64:/opt/openmpi/lib**

	PATH=/opt/intel/composer_xe_2015/bin
Turbo (ON/OFF)	**ON**
	Results
Number of nodes	**2**
Number of cores	**32cores&4GPU**
HPCG (base or optimized)	**optimized**
HPCG result	**112.136Gflops**

HPCG results submission form(MIC)

HPCG	
HPCG version	**HPCG 3.0 for Intel XEON and Phi**
System Spec	**Dual Intel Xeon E5-2630V3@2.4Ghz,32GB memory,Dual Intel Xeon Phi 7110p**
Compute Nodes	**1**
OS	**Redhat Enterprise Linux Server release 6.5**
MPI	**IMPI-5.0.2.044**
Compiler	**icc-15.0.1**
Compiler Flags	**-O3, -ffast-math,-ftree-vectorize, -ftree-vectorizer-verbose=0**
Environment Variables:	**MIC_OMP_NUM_THREADS=60 MIC_LD_LIBRARY_PATH=./bin/lib/mic:$MIC_LD_LIBRARY_PATH PATH=/opt/intel/composer_xe_2015/bin**
Turbo (ON/OFF)	**ON**
	Results
Number of nodes	**1**
Number of cores	**32cores&2MIC**
HPCG (base or optimized)	**optimized**
HPCG result	**4.46Gflops**

Appendix B The optimal kernel codes

- dnnForword

```
#define rbl 1/2
__attribute__((target(mic:0))) float *dW0,*dW1,*dW2,*dW3,*dW4,*dW5,*dW6;
__attribute__((target(mic:0))) float* dY0,*dY1,*dY2,*dY3,*dY4,*dY5,*dY6;
extern "C" int dnnForward(NodeArg &nodeArg)
{
    float   *d_X = nodeArg.d_X;
```

```
    float **d_Y = nodeArg.d_Y;
    float **d_W = nodeArg.d_W;
    float **d_B = nodeArg.d_B;

    float one = 1.0f;
    float zero    = 0.0f;

    int numN = nodeArg.numN; //size of minibatch
    int numL = nodeArg.dnnLayerNum - 1; //layer nums
    int numD = nodeArg.dnnLayerArr[0]; //node nums of input layer
    int*numA = &(nodeArg.dnnLayerArr[1]); //node nums of hiden layer and output layer

    int a1r=(int)numN*rbl;//row of part1
    int a2r=numN-a1r;
    int ly[numL];    //length of d_E for offload
    int lw[numL];        //length of d_W
    lw[0]=numA[0]*numD;
    ly[0]=a1r*numA[0];
    setmatY(d_Y[0],d_B[0],numN,numA[0]);
    for(int i=1;i<numL;i++){
        lw[i]=numA[i-1]*numA[i];
        ly[i]=numA[i]*a1r;
        setmatY(d_Y[i],d_B[i],numN,numA[i]);
    }

    dY0=d_Y[0];
    dY1=d_Y[1];
    dY2=d_Y[2];
    dY3=d_Y[3];
    dY4=d_Y[4];
    dY5=d_Y[5];
    dY6=d_Y[6];

    float* in;
    dW0=d_W[0];
    dW1=d_W[1];
    dW2=d_W[2];
    dW3=d_W[3];
    dW4=d_W[4];
    dW5=d_W[5];
    dW6=d_W[6];

#pragma offload target(mic:0) in(d_X:length(numD*a1r)) in(numA:length(numL))\
                        in(dW0:length(lw[0]) alloc_if(1) free_if(0))\
                        in(dW1:length(lw[1]) alloc_if(1) free_if(0))\
                        in(dW2:length(lw[2]) alloc_if(1) free_if(0))\
```

```
                    in(dW3:length(lw[3]) alloc_if(1) free_if(0))\
                    in(dW4:length(lw[4]) alloc_if(1) free_if(0))\
                    in(dW5:length(lw[5]) alloc_if(1) free_if(0))\
                    in(dW6:length(lw[6]) alloc_if(1) free_if(0))\
                    in(dY0:length(ly[0]) alloc_if(1) free_if(0))\
                    in(dY1:length(ly[1]) alloc_if(1) free_if(0))\
                    in(dY2:length(ly[2]) alloc_if(1) free_if(0))\
                    in(dY3:length(ly[3]) alloc_if(1) free_if(0))\
                    in(dY4:length(ly[4]) alloc_if(1) free_if(0))\
                    in(dY5:length(ly[5]) alloc_if(1) free_if(0))\
                    inout(dY6:length(ly[6]) alloc_if(1) free_if(1))\
                    signal(in)
{
        cblas_sgemm(CblasRowMajor, CblasNoTrans, CblasNoTrans,a1r , numA[0], numD, \
                                        one, d_X, numD, dW0,    numA[0], one, dY0, numA[0]);
        sigmoidY(dY0, a1r, numA[0]);

        cblas_sgemm(CblasRowMajor, CblasNoTrans, CblasNoTrans,a1r , numA[1], numA[0], \
                                        one, dY0, numA[0], dW1, numA[1], one, dY1, numA[1]);
        sigmoidY(dY1, a1r, numA[1]);
        cblas_sgemm(CblasRowMajor, CblasNoTrans, CblasNoTrans,a1r , numA[2], numA[1], \
                                        one, dY1, numA[1], dW2, numA[2], one, dY2, numA[2]);
        sigmoidY(dY2, a1r, numA[2]);
        cblas_sgemm(CblasRowMajor, CblasNoTrans, CblasNoTrans,a1r , numA[3], numA[2], \
                                        one, dY2, numA[2], dW3, numA[3], one, dY3, numA[3]);
        sigmoidY(dY3, a1r, numA[3]);
        cblas_sgemm(CblasRowMajor, CblasNoTrans, CblasNoTrans,a1r , numA[4], numA[3], \
                                        one, dY3, numA[3], dW4, numA[4], one, dY4, numA[4]);
        sigmoidY(dY4, a1r, numA[4]);
        cblas_sgemm(CblasRowMajor, CblasNoTrans, CblasNoTrans,a1r,numA[5], numA[4], \
                                        one, dY4, numA[4], dW5, numA[5], one, dY5, numA[5]);
        sigmoidY(dY5, a1r, numA[5]);
        cblas_sgemm(CblasRowMajor, CblasNoTrans, CblasNoTrans,a1r , numA[6], numA[5], \
                                                one, dY5, numA[5], dW6, numA[6], one, dY6,
numA[6]);

}

    for (int i = 0; i < numL; i++) {

        if (i == 0){
            cblas_sgemm(CblasRowMajor, CblasNoTrans, CblasNoTrans, a2r, numA[i], numD, \
                            one, d_X+a1r*numD, numD, d_W[i],    numA[i], one, d_Y[i]+ly[i], numA[i]);
        } else {
            cblas_sgemm(CblasRowMajor, CblasNoTrans, CblasNoTrans, a2r, numA[i], numA[i-1], \
                            one, d_Y[i-1]+ly[i-1], numA[i-1], d_W[i], numA[i], one, d_Y[i]+ly[i], numA[i]);
```

```
            }
            if (i == numL-1) { //softmax on output layer
                #pragma offload_wait target(mic:0) wait(in)
                softmaxZ(d_Y[i], d_Y[i], numN, numA[i]);
            } else { //sigmod on hiden layers
                sigmoidY(d_Y[i]+ly[i],a2r, numA[i]);
            }
        }

        return 0;
}
```

- dnnBackword

```
extern "C" int dnnBackward(NodeArg &nodeArg)
{
        float **d_Y = nodeArg.d_Y;
        float **d_W = nodeArg.d_W;
        float **d_E = nodeArg.d_E;
        int       *d_T = nodeArg.d_T;

        float one = 1.0f;
        float zero    = 0.0f;

        int numN = nodeArg.numN; //size of minibatch
        int numL = nodeArg.dnnLayerNum - 1; //layer nums
        int numD = nodeArg.dnnLayerArr[0]; //node nums of input layer
        int*numA = &(nodeArg.dnnLayerArr[1]); //node nums of hiden layer and output layer

        errorOutput(d_E[numL-1], d_Y[numL-1], d_T, numN, numA[numL-1]);

        float *dE6=d_E[6];
        float *dE5=d_E[5];
        float *dE4=d_E[4];
        float *dE3=d_E[3];
        float *dE2=d_E[2];
        float *dE1=d_E[1];
        float *dE0=d_E[0];
        float* in1;
        int a1r=(int)numN*rbl;//row of part1
        int a2r=numN-a1r;
        int le[numL];    //length of d_E for offload
        int ly[numL];    //length of d_Y for offload
//      int lw[numL];       //length of d_W
```

```
//    lw[0]=numA[0]*numD;
      ly[0]=a1r*numA[0];
      le[0]=a1r*numA[0];
      for(int i=1;i<numL;i++){
//          lw[i]=numA[i-1]*numA[i];
            ly[i]=numA[i]*a1r;
            le[i]=numA[i]*a1r;
      }

#pragma offload target(mic:0) in(dE6:length(le[6])) in(numA:length(numL))\
                        out(dE5:length(le[5]))\
                        out(dE4:length(le[4]))\
                        out(dE3:length(le[3]))\
                        out(dE2:length(le[2]))\
                        out(dE1:length(le[1]))\
                        out(dE0:length(le[0]))\
                        nocopy(dW1)\
                        nocopy(dW2)\
                        nocopy(dW3)\
                        nocopy(dW4)\
                        nocopy(dW5)\
                        nocopy(dW6)\
                        nocopy(dY0)\
                        nocopy(dY1)\
                        nocopy(dY2)\
                        nocopy(dY3)\
                        nocopy(dY4)\
                        nocopy(dY5)\
                        signal(in1)
{

            cblas_sgemm(CblasRowMajor, CblasNoTrans, CblasTrans, a1r,numA[5], numA[6], \
                              one, dE6, numA[6], dW6, numA[6], zero, dE5,numA[5]);
            errorTrans(dE5, dY5,a1r, numA[5]);
            cblas_sgemm(CblasRowMajor, CblasNoTrans, CblasTrans, a1r,numA[4], numA[5], \
                                          one, dE5, numA[5], dW5, numA[5], zero, dE4,numA[4]);
            errorTrans(dE4, dY4,a1r, numA[4]);
            cblas_sgemm(CblasRowMajor, CblasNoTrans, CblasTrans, a1r,numA[3], numA[4], \
                                                  one, dE4, numA[4], dW4, numA[4], zero, dE3,numA[3]);
            errorTrans(dE3, dY3,a1r, numA[3]);
            cblas_sgemm(CblasRowMajor, CblasNoTrans, CblasTrans, a1r,numA[2], numA[3], \
                                                  one, dE3, numA[3], dW3, numA[3], zero, dE2,numA[2]);
            errorTrans(dE2, dY2,a1r, numA[2]);
            cblas_sgemm(CblasRowMajor, CblasNoTrans, CblasTrans, a1r,numA[1], numA[2], \
                                    one, dE2, numA[2], dW2, numA[2], zero, dE1,numA[1]);
            errorTrans(dE1, dY1,a1r, numA[1]);
```

```
            cblas_sgemm(CblasRowMajor, CblasNoTrans, CblasTrans, a1r,numA[0], numA[1], \
                                                    one, dE1, numA[1], dW1, numA[1], zero, dE0,numA[0]);
            errorTrans(dE0, dY0,a1r, numA[0]);

}
      for (int i = numL-2; i>=0; i--) {
                  cblas_sgemm(CblasRowMajor, CblasNoTrans, CblasTrans,a2r, numA[i], numA[i+1], \
                              one, d_E[i+1]+le[i+1], numA[i+1], d_W[i+1], numA[i+1], zero, d_E[i]+le[i],
numA[i]);
            errorTrans(d_E[i]+le[i], d_Y[i]+ly[i], a2r, numA[i]);
      }

      #pragma offload_wait target(mic:0) wait(in1)
      return 0;
}
```

Appendix C Key Achievements

a. Research:

任务分配均衡的双参数 CFAR 舰船检测并行算法

张临杰 1，张杰 2，张晰 2，郎海涛 3

1.中国海洋大学数学科学学院，山东 青岛 266100；

2.国家海洋局第一海洋研究所，山东 青岛 266061；

3.北京化工大学应用物理系，北京 100029

摘要： 双参数恒虚警率（Constant False Alarm Rate，CFAR）是舰船目标检测中最为常用的算法之一。近年来随着 SAR 分辨率的提高，SAR 图像幅宽增大，并且在检测时希望尽量保持舰船轮廓以便后续的舰船目标识别。在此背景下双参数 CFAR 算法本身虽然仍能满足目标检测需求，但是算法运行时间过长，不利于信息的及时处理。传统的 MPI 并行化解决方案在分配检测任务给各进程时，没有考虑因陆地掩膜，几何校正等预处理所导致的图像中待检测点分布不均。针对这一问题，本文给出了一个改进的 MPI 并行化解决方案。较传统的 MPI 并行化解决方案，该方案能够较为均衡地为各个进程分配检测任务。在实验用集群计算机上的实验结果表明，改进后并行标准效率提高约 43%。此外为应对机载 SAR 实时舰船目标检测的需求，在多核 PC 机上也进行了实验。结果表明本文所给并行化解决方案在多核 PC 机上也能够有效地缩短检测时间，对实现机载 SAR 实时舰船目标检测也有积极意义。

关键词： 舰船检测,双参数 CFAR,SAR 图像,MPI 并行化,检测窗口尺寸

中图分类号： TP751　　**文献标志码：** A

1　引　言

舰船是专属经济区监测的主要对象之一，发展海面舰船目标检测技术具有重要意义。由于合成孔径雷达(Synthetic Aperture Radar，SAR)具有全天候、全天时、大范围等诸多优点，在舰船检测方面已被证明是一种有效的手段，一直以来受到广泛的关注。

双参数恒虚警率（Constant False Alarm Rate，CFAR）是一种自适应门限检测，能够适应背景杂波的变化，是常用的舰船目标检测方法之一（El-Darymli，2013）。然而近年来由于 SAR 传感器技术的进步，SAR 分辨率得到很大提高，已能达到亚米级。随着分辨率的提高，SAR 图像幅宽大幅增加，长、宽均超过一万个像素点的情况已经比较常见；舰船目标不再以点目标的形式出现，而是以硬目标的形式出现，目标轮廓较为清晰，因此在检测的同时还要求尽可能地保持目标轮廓；此外对小型舰船的检测也提出了要求。这些原因导致在使用双参数 CFAR 进行检测时，计算时间过长，不利于信息的及时处理。利用并行处理技术是缩短检测时间的有效解决方法。

目前主流的并行处理技术有基于 openMP、基于 MPI 和基于 GPU 三种，分别适用于多核计算机，集群计算机和带 GPU 计算卡的计算机。通过调查以往文献我们发现，由于 SAR 成像中所处理的数据量大，因此以上并行技术在 SAR 图像成像中的应用已被引起广泛重视，例如李景山等（2014）利用 MPI 并行处理技术，在集群计算机上实现了的 SAR 环境一号 C 卫星 SAR 图像全分辨率快速实时处理系统；赵敬亮等（2010）和孟大地等（2013）利用基于 GPU 的并行处理技术，研究了 SAR 实时成像处理；Park 等(2012)利用 openMP 技术在配备了众核加速卡(Intel R Xeon Phi TM coprocessors)的单节点计算机上，实现了秒以内 3K×3K 的图像成像。而一直以来由于 SAR 成像后图像尺寸较小，因此对基于 SAR 图像的目标识别算法的并行化研究文献较少。但近年来由于前面所提到 SAR 分辨率提高的原因，对基于 SAR 图像的目标识别算法的并

收稿日期：

基金项目：国家自然科学基金 编号：11371333)；山东省自然科学基金(编号：ZR2013FQ026)；中央高校基本科研业务费专项(编号：201362033)；海洋公益性行业科研专项经费项目（编号：201505002）

第一作者简介：张临杰(1975—)，女，博士，现从事基于 SAR 图像的舰船目标识别的理论和应用研究，E-mail：zhanglinjie@ouc.edu.cn

通信作者简介：张杰（1963— ），男，教授，博士生导师，从事于海洋遥感学研究，E-mail：zhangjie@fio.org.cn

行化研究也逐渐引起重视，例如林旭等（2013）利用 openMP 并行技术对双参数 CFAR 算法进行了并行化，在四核计算机上将检测时间缩短了约 75%。基于 MPI 和 GPU 的舰船目标检测算法的并行化研究目前在国内仍存在很大的空白。

考虑到集群计算机目前已较为普遍，且具有不受单个节点内存和计算能力的限制，因此本文采用 MPI 并行技术对双参数 CFAR 进行并行化。传统的 MPI 并行方案按图像尺寸平均分割检测任务。然而我们在执行实际检测任务时，经常要对待检测的 SAR 图像先做陆地掩膜，几何校正等预处理。这些预处理会导致处理后的 SAR 图像中待检测点分布不均。众所周知，在未进行 MPI 并行时，这种待检测点的分布不均并不影响检测效率。然而在并行时则会导致各进程所负责的检测任务不均衡，有的进程完成的快，有的进程完成的慢。而并行检测时间是由最慢的那个进程所决定的。

针对这一问题，本文给出了一个改进的 MPI 并行化解决方案。与传统的解决方案相比，该方案考虑了预处理所导致的待检测点分布不均，按照待检测点个数将检测任务较为均衡地分配给各个进程，从而缩短整体检测时间。实验结果表明，该方案无论是在集群计算机，还是在多核 PC 机上的都能够有效地缩短检测时间，提高加速比和标准效率，是一个行之有效的并行解决方案。

2　双参数 CFAR 检测算法

双参数 CFAR 检测使用三个窗口：目标窗口 T，保护窗口 P，和背景窗口 B，分别对应图 1 中的斜纹区域，空白区域和横线区域。目标窗口内是待检测的点，海面背景信息从背景窗口获得，保护窗口是为了确保舰船目标部分不会被包含在背景窗口中。目标窗口内的点被检测为目标的判断标准是

$$\frac{\mu_t-\mu_b}{\sigma_b} > t \Leftrightarrow \text{目标}$$

其中 μ_t 为目标窗口内的均值，μ_b 为背景窗口均值，σ_b 为背景窗口内的标准偏差，t 控制虚警率，也称为标称化因子（一般取 2 或 3），上式中不等式左边也被称为检测比率。在对整个 SAR 图像进行检测时，上面三个窗口按照一定的步长（一般与目标窗口尺寸取相同值）在图像中滑动，逐一判断目标窗口内的点是否为目标点（种劲松等，2006）。

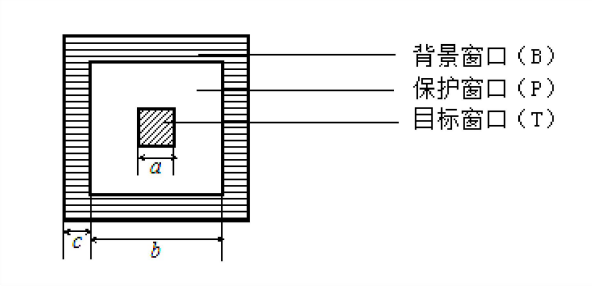


图 1　双参数 CFAR 检测窗口（B:背景窗口 P:保护窗口 T:目标窗口）

Fig.1 The detection window of two-parameter CFAR（B:background window P:guard window T:target window）

由上面简述可知，在双参数 CFAR 检测中需要指定三个窗口的尺寸(见图 1 中 *a, b, c*)和标称化因子 t。一般来说，保护窗口的尺寸要足够大，以免舰船目标的一部分被包含在背景窗口中,从而增加背景均值和方差,使得检测比率减小；同样背景窗口的尺寸不宜过大，以免包含临近目标，减小检测比率；目标窗口尺寸越小越能够降低漏检的概率，但由于目标窗口要遍历整个图像,因此计算量也随之大大增加。

在分辨率较低的 SAR 图像中，舰船目标表现为点目标（参见图 2），仅占若干个像素点，轮廓不清晰。当使用双参数 CFAR 检测时，一般取目标窗口尺寸(即图 1 中 *a* 值)为小船目标船长的 1-2 倍，保护窗口尺寸(即图 1 中 *b* 值)为大船目标的 2 倍，背景窗口尺寸(即图 1 中 *c* 值)通常取 3 以上的整数（种劲松等，2006）。

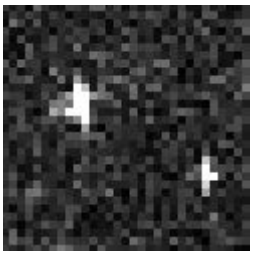


图 2　分辨率为 25 米时的舰船目标

Fig.2 Ship targets when the resolution is 25m

值得注意的是，上面窗口尺寸选取策略适用于分辨率较低，目标以点目标形式出现的情况。当分辨率较高时，舰船目标在图像中以硬目标的形式出现，若仍采用上述选取方案，一是容易引起图 3 所示的漏检；二是无法很好地保留舰船轮廓。而保留舰船轮廓，也是高分辨率舰船检测的实际需求之一。

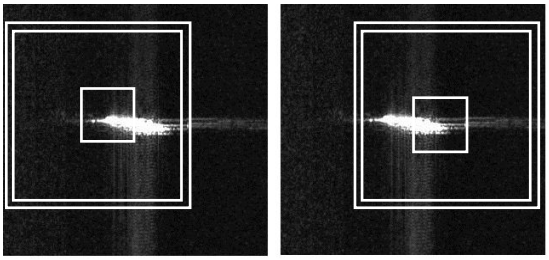
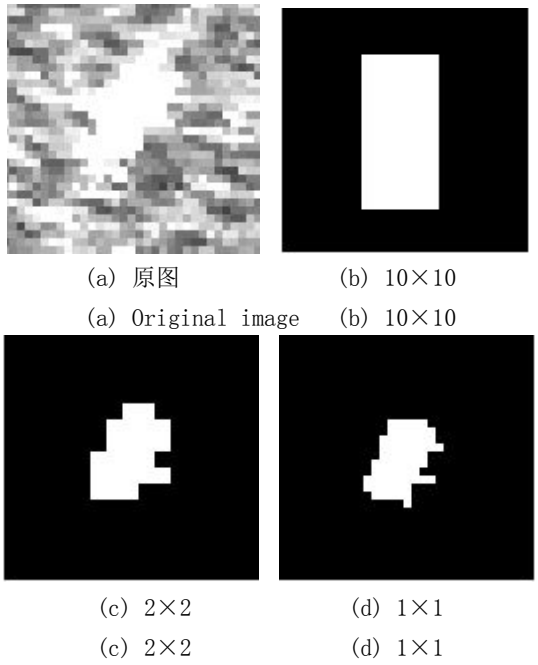


图 3 舰船目标以面目标形式出现时的漏检例

Fig.3 Undetected targets in high resolution

舰船目标轮廓保持情况直接受目标窗口尺寸的影响，具体例参见图 4。由此可知保持舰船轮廓的最直接的解决方案就是不再依据小船船长设定目标窗口尺寸，而是尽可能小地设置目标窗口尺寸（最小值为 1）。而减小目标窗口尺寸又带来了计算量的大幅增加。

综上所述，将双参数 CFAR 检测算法应用于高分辨率的目标检测时，有两个原因导致计算量增大：一是图像本身尺寸增大直接导致计算量增大；二是为防止漏检，以及很好地保留舰船轮廓，而减小目标窗口尺寸所引起的计算量增大。在对一幅大尺寸高分辨率图像进行检测时，检测时间长达几分钟或者更长，妨碍了舰船目标实时检测的实现。为缩短检测时间，下面我们考虑对双参数 CFAR 检测算法进行并行化处理。



(a) 原图　　(b) 10×10

(a) Original image　　(b) 10×10

(c) 2×2　　(d) 1×1

(c) 2×2　　(d) 1×1

图 4 不同目标窗口尺寸时的舰船轮廓比较

Fig.4 Ship contour in different sizes of target window

3 传统的 MPI 并行化解决方案

一般来说，对双参数 CFAR 检测算法进行 MPI 并行化时，将 SAR 图像平分为 n 份，每个进程负责其中一份的检测，最后再将所有检测结果汇总。分割方案一般有水平，垂直和网格分割三种。由于双参数 CFAR 检测算法的目标判断标准仅依据图 1 所示的三个局部窗口内的信息，因此无论哪种分割都不影响最终检测结果。而本文所使用的 SAR 图像数据是按行存放，因此我们采用水平分割方式。另外需要注意的是，各进程在读取所负责的子图像数据时，应在子图的上下两个方向各多读取一个冗余区域的数据，以确保所负责子图的上下两端边界部分的检测能够正常进行（参见图 5）。冗余区域的宽度应与图 1 中目标窗口左上角与背景窗口左上角的垂直距离相等。注意第 1 个进程只有下方的冗余区域，最后一个进程只有上方的冗余区域。

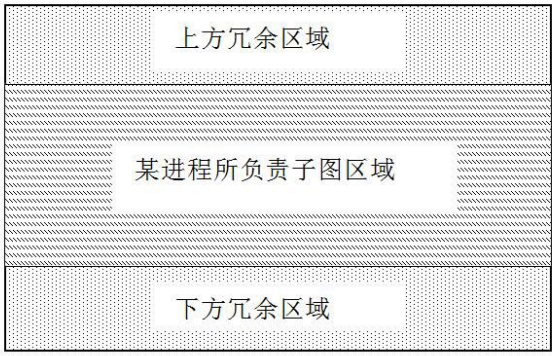


图 5 某进程读取图像数据示意图

Fig.5 Part of image to be imported for a process

并行步骤详述如表 1 所示。

表 1 传统的 MPI 并行步骤

Table 1　Traditional MPI parallel steps

	进程 0	其他进程
步骤 1	计算本进程所负责子图像范围区间	同进程 0
步骤 2	从文件读入本进程所需子图像数据	同进程 0
步骤 3	对本进程所负责子图进行双参数 CFAR 检测	同进程 0
步骤 4	收集其他进程的检测结果并汇总	将检测结果发送给进程 0

下面我们以一个实际的 SAR 图像为例（参见图 6），说明上面传统并行方案所存在的问题。该图像是一个经过了几何校正和陆地掩膜后的星载 16

位 SAR 图像，大小为 12477×14569，分辨率约为 11.83 米。检测窗口尺寸，即图 1 中的 *a, b, c* 值分别设为 2, 30 和 3。实验在有 3 个节点的集群计算机上进行。每个节点配置如下：操作系统 SUSE Linux Enterprise Server 10，4G 内存，2 个 4 核 AMD Opteron(tm) Processor 6128 800MHz 处理器。

未进行 MPI 并行化时总检测时间为 489.21 秒，其中 CFAR 算法运行时间为 488.35 秒，其余时间为读写文件所用时间。下面我们将图像平分为 8 份，每个进程（Proccessor）负责一份。各进程所负责的行范围区间见图 6，并行结果见表 2 与图 7。

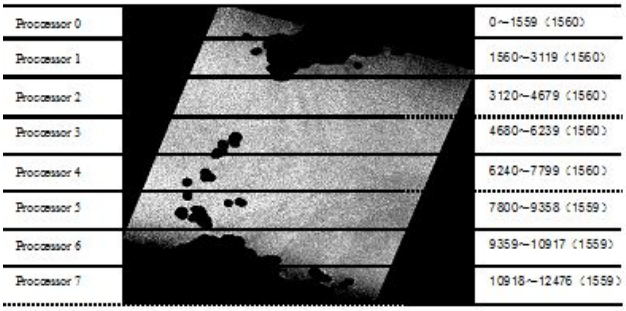


图 6 水平平均分割图像结果

Fig.6 Divide image horizontally in average

表 2 改进前各进程计算时间详细

Table 2 Detection time, waiting time and total time of each process before improvement

进程号	CFAR 检测时间	通信等待时间	总时间
0	20.00	73.83	93.97
1	50.91	0.60	51.60
2	92.38	1.17	93.66
3	89.25	0.14	89.51
4	89.94	1.13	91.08
5	85.14	0.84	86.09
6	56.67	0.17	56.96
7	16.76	4.17	21.04

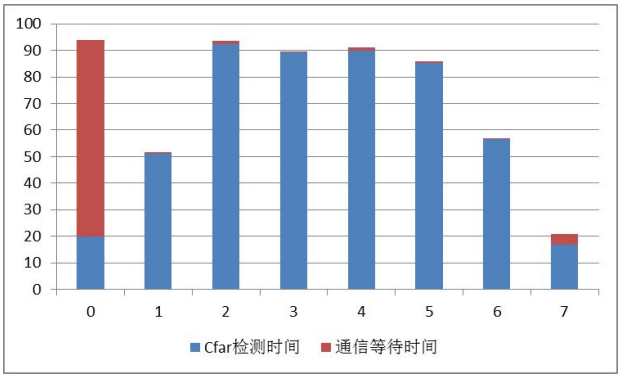


图 7 改进前各进程检测时间与等待时间

Fig.7 Detection time and waiting time of each process before improvement

由上面并行结果可以看出，经过几何校正和陆地掩膜后的 SAR 图像待检测点分布不均匀，若将图像平均分配给各进程，将导致各进程检测时间严重不均衡。例如进程 0 的 CFAR 检测时间仅 20.00 秒，然而进程 4 的检测时间却长达 89.94 秒，相差 4 倍以上。进程 0 在完成自己所负责子图的检测后，必须等待所有进程完成检测，才能收集汇总检测结果并输出，等待时间高达 73.83 秒。加速比 $Sp=t1/tp$（$t1$ 为非并行时的运行时间，tp 是在有 p 个进程时的运行时间）仅为 5.20，标准效率 $Ep=Sp/p$ 等于 0.65。

4 改进的 MPI 并行化解决方案

为避免上面各进程负载不均衡问题，本文所给出的并行化解决方案，将根据待检测点个数分割图像。分割方案详细如下：

（1）统计整幅图像的待检测点总数 n；

（2）计算每个进程所负责的检测点个数 m，（$m=n/p$，其中 p 为进程数）；

（3）重新自上而下逐行遍历图像，根据 m 值逐一确定每个进程所负责的行区间范围。

需要说明的是，为简化算法的实现，进程间任务分割的最小单位为行，因此每个进程所负责的检测点数并不严格等于 m，但与 m 值最多相差不到 1 行像素点数，即图像的列数；各进程在读取所负责的子图像数据时，仍和传统的解决方案一样，在所负责子图的上下两个方向各读取一个冗余区域的数据，以确保所负责子图的上下两端边界部分的检测能够正常进行；本实验中的待检测点即为非零点，在实际应用中也可以根据经验设定一个非零的全局阈值，或是使用其它算法计算出一个全局阈值，然后根据该阈值判断是否为待检测点；与传统的解决方案相比，我们仅改变了任务分配方式，没有改变双参数 CFAR 的检测机制，因此对检测结果没有影响，这一点在随后的实验中也得到了证实。最后我们给出并行方案的详细步骤流程如表 3 所示。

表 3 改进的 MPI 并行步骤

Table 3 Improved MPI parallel steps

进程 0	其他进程

步骤 1	读入整个图像，统计待检测点总数，确定各个进程所负责图像范围区间，并将分割方案发送给其他进程	从进程 0 接收图像分割方案，明确本进程所负责图像范围区间
步骤 2	从文件读入本进程所需子图像数据	同进程 0
步骤 3	对本进程所负责子图进行双参数 CFAR 检测	同进程 0
步骤 4	收集其他进程的检测结果并汇总	将检测结果发送给进程 0

在步骤 1 中我们让进程 0 负责统计待检测点总数，也可以将统计任务分配给每个进程并发执行。但是考虑到多个进程同时读取一个文件会大大降低文件读取速度，并且在内存满足需求的情况下，读取文件一部分和读取整个文件的时间差别不明显，因此本文对步骤 1 没有并发执行。

下面仍以图 6 所示图像为例进行试验，进程数仍指定为 8，分割结果见图 8，并行结果见表 4 与图 9。

由检测结果可以看出，与传统的并行方案相比，各进程的 CFAR 检测时间接近均衡，加速比 7.45，标准效率 0.93，都有了很大提高。虽然在第一步统计待检测点总数时，需要读入整个 SAR 数据文件，有一定的时间花费（约 3 秒），但是从总计算时间的缩短情况来看，这里的计算花销是值得的。

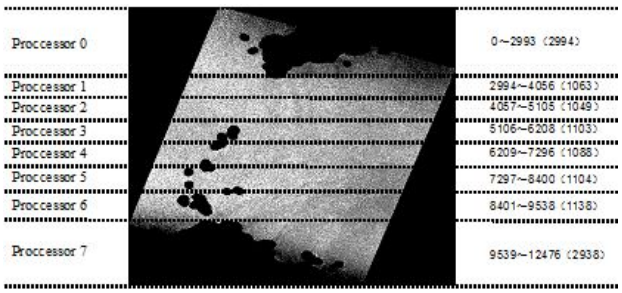


图 8 根据待检测点个数分割图像结果

Fig.8 Divide image in terms of number of points to be detected

表 4 改进后各进程计算时间详细

Table 4 Details of each process after improvement

进程号	CFAR 检测时间	通信等待时间	总时间
0	61.02	1.19	65.65
1	59.19	2.00	64.51
2	59.01	2.28	64.63
3	59.24	2.16	64.75
4	59.17	2.43	64.93
5	59.22	2.53	65.07
6	59.43	2.50	65.25
7	60.58	1.45	65.47

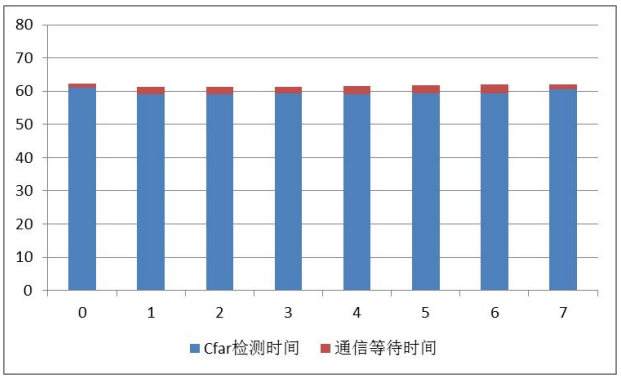


图 9 改进后各进程的检测时间与等待时间

Fig.9 Detection time and waiting time of each process after improvement

改进前后的并行效率比较见表 5。由表 5 可以看出，与传统的并行方案相比，本文所给并行化解决方案，有效地提高了并行效率。

表 5 改进前后并行效率比较

Table5 Comparison of the parallel efficiency before and after improvement

	总时间	加速比	标准效率
非并行时	489.21	---	---
改进前	93.97	5.20	0.65
改进后	65.65	7.45	0.93

检测结果与传统并行方案检测结果相同，共检测出疑似目标 583 处，检测结果见图 10。由于本实验图像尺寸远超屏幕分辨率，因此我们仅展示了整体检测结果的一部分。所展示局部左上角在整幅检测结果图中的坐标为（8860，10715），大小为 300×300。图 10 还给出了其中一个疑似目标的放大图，可以看出舰船轮廓被较好地保留了下来。

另外需要说明的是，在执行舰船检测任务时，还须在 CFAR 检测结果的基础上，利用形态学的方法，对疑似目标进行进一步地筛选，剔除虚警。然而此部分超出本文所论范围，因此略过。

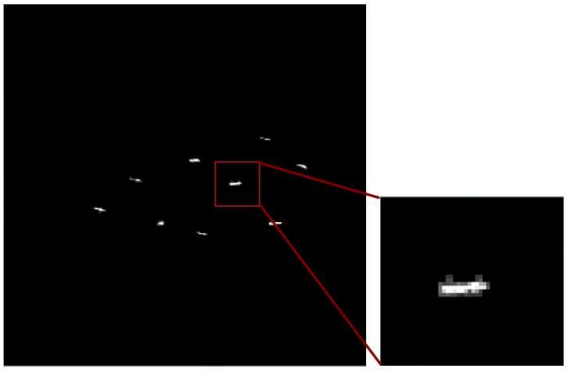


图 10 检测结果局部

Fig.10 A part of detection result

最后图 11 给出不同进程数时的加速比。从图中可以观察到，随着进程数的增加，加速比呈增加趋势。但是增加趋势有所变缓，即标准效率有所下降。原因有两个：一是随着进程数的增加，并行方案中的步骤 3 的执行时间减少，在总计算时间中所占比例下降。而未被并发执行的步骤 1 的执行时间不因进程数的增加而改变，因而在总计算时间中所占比例增大，影响了标准效率的提高；二是并行方案中的步骤 2，即从图像数据文件读入本进程所需数据，当多个进程同时对同一个数据文件进行读取时，会在一定程度上影响各进程的读取速度。由以上分析可知，当进程数增加到一定程度，加速比将不再随进程数的增加而增大。

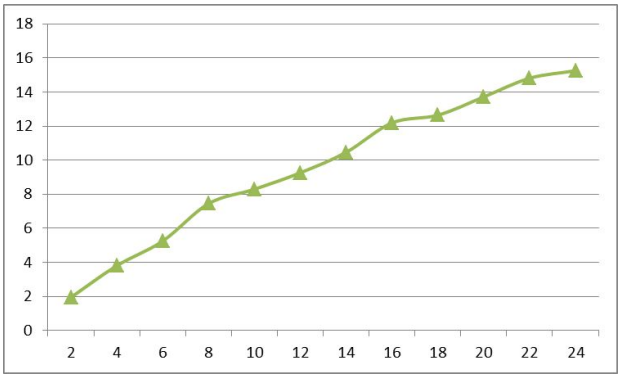


图 11 不同进程数时的加速比

Fig.11 The speed-up ratio with different number of processes

5 多核 PC 机上的并行效果

众所周知 MPI 并行程序一般在集群计算机上执行。而有时在执行检测任务时无法使用集群计算机。例如在搭载机载 SAR 的飞行器上进行目标检测时，受飞行器空间限制无法搭载集群计算机，只能使用性能较好的 PC 机。而目前绝大多数 PC 机都配备了多核处理器，甚至配备多个多核处理器。为考察上面 MPI 并行程序在多核 PC 机上的效果，我们在多核 PC 机上也进行了试验。试验用机是 32 位 Windows XP 操作系统，4G 内存，2 个 4 核 Quad-core AMD opteron(tm) processor 2378 792MHz 处理器。

非并行时总检测时间为 355.21 秒。通过任务管理器我们观察到，在程序运行的不同阶段，各核使用情况大体分为三种，参见图 12(a-c)。无论是哪一种情况，CPU 使用率都基本保持在 13%左右，即在同一时刻仅 1 个核的资源被使用。执行上面 MPI 并行程序，并指定进程数为 8 时总检测时间缩短为 50.22 秒，加速比 7.07，标准效率 0.88。通过任务管理器，可以观察到 8 个核都被充分使用，CPU 使用率达到 100%，参见图 12(d)。

(a)

(b)

(c)

(d)

图 12 Windows 任务管理器截图

Fig.12 The screen shot of Windows task manager

6 结 论

传统的双参数 CFAR 的 MPI 的并行化解决方

案，是将待检测图像按照图像的尺寸平均分配给各个进程。然而在实际执行目标检测任务时，往往会在检测前对 SAR 图像先进行陆地掩膜，几何校正等预处理，使得处理后的 SAR 图像中待检测点分布不均，从而导致进程间任务分配不均衡，影响并行效率的发挥。

针对这一问题，本文所给解决方案依照待检测点个数为各个进程均衡分配检测任务。实验结果表明，该方案无论是在集群计算机上，还是多核 PC 机上都能有效地缩短检测时间，提高并行效率。尤其是在多核计算机上缩短检测时间对实现机载 SAR 舰船目标实时检测具有积极意义。

另外，本文所给出的并行方案不仅适用于对经过陆地掩膜或几何校正后的 SAR 图像进行检测，而且对一些复合检测算法（如 Ji 等（2010）提出的先用 Kfar 算法计算出一个全局阈值，然后再对大于该阈值的点进行双参数 CFAR 检测）也同样适用。

在执行任务的飞机上搭载配备 GPU 或 Intel 众核加速卡的非集群计算机，利用 GPU 和 openMP 并行技术进一步缩短检测时间，实现机载 SAR 舰船目标实时检测，是下一步的重要研究方向。

参考文献(References)

El-Darymli K, McGuire P, Power D, Moloney C. 2013. Target detection in synthetic aperture radar imagery: a state-of-the-art survey[J]. Journal of Applied Remote Sensing, 7(1): 079998-079998.

Ji Y G, Zhang J, Meng J M, Zhang X. 2010. A new CFAR ship target detection method in SAR imagery[J]. Acta Oceanologica Sinica, 29(1): 12-16

Li J S, Wen S Y, Wang J, Zhong L H, Zhang W Y. 2014. Design and Implementation of a Real-time Processing System of Full Resolution Quick-look Image of HJ-1 Environmental Satellite C SAR Based on High Performance Cluster[J]. Journal of Radars, 3(3): 332-333 (李景山, 温双燕, 王建, 仲利华, 张问一. 2014. 基于高性能机群的环境一号 C 卫星 SAR 图像全分辨率快视实时处理系统设计与实现. 雷达学报, 3(3): 332-333)

Lin X, Hong J, Sun X, Yan Y. 2013. Fast ship detection method based on ScanSAR image［J］. Journal of University of Chinese Academy of Sciences, 30(6): 793-799 (林旭,洪峻,孙显,鄢懿. 2013. ScanSAR 图像舰船目标快速检测方法[J]. 中国科学院大学学报,2013,30(6):793-799)

Meng D D, Hu Y X, Shi T, Sun R, Li X B． 2013. Airborne SAR Real-time Imaging Algorithm Design and Implementation with CUDA on NVIDIA GPU[J], Journal of Radars, 2(4):481-491 (孟大地, 胡玉新, 石涛, 孙蕊, 李晓波. 2013. 基于 NVIDIA GPU 的机载 SAR 实时成像处理算法 CUDA 设计与实现[J]. 雷达学报, 2(4): 481-491)

Park J, Tang P T P , Smelyanskiy M, Kim D, Benson T．2012. Efficient Backprojection-based Synthetic Aperture Radar Computation with Many-core Processors[C]. Conference on High Performance Networking and Computing, Nov. 2012 - Nov. 2012, Salt Lake City, Utah, 1-11

Zhao J L, Lei W M, Huang Y H. 2010. Real-time Parallel Imaging Processing of SAR Based on GPU [J], Information Research, 36(22):28-30 (赵敬亮,雷万明,黄银和, 2010. 基于 GPU 的 SAR 实时并行成像处理,信息化研究， 36(22): 28-30)

Zhong J S, Oouyang Y, Zhu M H. 2006. Ocean Target Detection in Synthetic Aperture Radar[M]. Beijing: Ocean Publishing Press: 47-49 (种劲松，欧阳越，朱敏慧．2006. 合成孔径雷达图像海洋目标检测 [M]．北京: 海洋出版社: 47-49)

Task Distribution Balancing for Parallel Two-parameter CFAR Ship Detection

ZHANG Linjie[1], ZHANG Jie[2], ZHANG Xi[2], LANG HaiTao[3]

1. College of Mathematical Science, Ocean University of China, Qingdao Shandong 266100, China

2. The First Institute of Oceanography, SOA, Qingdaog, 266061, China

3. Physics & Electronics Department, Beijing University of Chemical Technology, Beijing, 100029, China

Abstract: [Objective] Ship detection is of great significance for both military and civilian applications. Synthetic aperture radar(SAR) with all-day, all-weather, ultra-long-range characteristics, has been used

widely. Two-parameter constant false alarm rate(CFAR) method is one of the most popular methods for target detection. It is an adaptive threshold detection scheme, works efficiently when the background clutter is unevenly distributed. However, in recent years, the resolution of SAR images is improved greatly due to the rapid development of the SAR sensor. With the improvement of resolution, the size of SAR images increased sharply, and the ship targets no longer appear as point targets but in the form of hard targets. The contour of targets becomes more clear too. When we make use of the two-parameter CFAR to detect ship target with good contour, much more computation time is needed. MPI parallelization is a workable solution to shorten the computation time of two-parameter CFAR with MPI parallel technique.

The traditional MPI parallelization divides the SAR image horizontally/vertically in average. However in the practical application, preprocessing as land mask, geometric correction and etc was performed before detection. These preprocessing could cause uneven distribution of points to be detected. This uneven distribution would lead to the unbalance of tasks between the parallel processes. So the efficiency of MPI parallelization would be influenced greatly. The objective of this paper is to emit the negative influence caused by the uneven distribution.

[Method] In this paper, we give an improved MPI parallel solution of two-parameter CFAR ship detection method, in which the SAR image is divided in term of number of points to be detected. Partitioning strategy is given as follows. First, calculate the total points number to be detected. Secondly, compute the rough number of responsible points for each process. Thirdly, decide the responsible rows of image for each process.

In this way, the whole detection task is divided among processes in balance. Details of the improved parallel algorithm is given below.

Step1: The first process computes partitioning strategy and send it to the other processes.

Step2: Each process imports its responsible part of image.

Step3: Each process performs two-parameter CFAR detection on its responsible part of image.

Step4: The first process gathers detection results from the other processes.

[Result] Numerical experiment was carried out on a cluster computer. When the number of processes is 8, the speed-up of the improved parallel algorithm is 7.45, which is much better than that of the normal parallel algorithm. The efficiency of parallelization increased about 43%. Similar experiment was also carried out on a multi-core computer, and similar result was obtained.

[Conclusion] Experimental results show that the improved parallel solution can shorten the detection time and improve the parallel efficiency well on cluster computer or multi-core computer. This study is of positive significance for real-time ship detection based on airborne SAR image. Further research is needed to shorten the detection time more by making use of GPU or Intel MIC architecture.

Key words: Ship detection, CFAR, SAR images, MPI parallelization, size of detection window

CLC number: TP751 **Document code:** A

b. Teaching:

The picture of the certificate of RDMA Programming Competition was shown below.