Southeast University

Proposal For ASC 20-21

Yifan Huang
Haorui Li
Ruicheng Gao
Chaoyang Wang
Jiangtao Wang

8th January, 2021

Contents

1 Brief Background Description of Supercomputing Activities

1.1 Hardware and Software Platforms

Today, the data deluge is impacting scientists in both research fields and industrial applications in a very profound way. In order to address this challenge, we establish a high performance computational (HPC) cluster in 2016.

The HPC cluster is equipped with more than 300 cores, 3.3 TB memory and 144 TB storage with Lustre (parallel) file system. The peak performance of the whole HPC cluster is 13 TFlops. Our HPC cluster consists of a number of batch nodes and a small number of special purpose nodes.

For the batch nodes we differentiate between so-called thin nodes, fat nodes, and GPU nodes. Ten thin nodes constitute the majority of the available batch nodes. GPU nodes use GPUs to accelerate the computations. Fat nodes have more physical cores (96) and larger memory (2 TB) than the thin nodes, which is especially suitable for analyzing large scale of sequencing data. Special node is so-called service node. The service node provides login service and administrates the whole HPC cluster. Every computing node has a Mellanox 56 Gb/s InfiniBand adapter providing 4 $\times$ FDR (Fourteen Data Rate) resulting in 56 Gbit/s inter-node bandwidth, with an inter-island latency of 3 µs. All nodes within HPC cluster connected by 1000 Mb interconnect. Each node runs under the same operating system (a Linux distribution compatible with Red Hat Enterprise Linux). The HPC cluster also set up with a parallel computational environment.

The detailed information of the hardware and software configuration of our HPC cluster is listed in Figure 1 and Figure 2, respectively.

Item	Name	Configuration	Number
Login and management node	Inspur NF5280M4	CPU: Intel Xeon E5-2620v3 x 2, 2.4Ghz, 6 cores	1
		Memory: 8G, DDR4, 2133Mhz	
		Hard disk: 1.5 TB SATA	
Compute node	Inspur NX5440M4	CPU: Intel Xeon E5-2650v3 x 2, 2.3Ghz, 10 cores	10
		Memory: 16G, DDR4, 2133Mhz	
		Hard disk: 300G SSD	
GPU node	Inspur NF5280M4	CPU: Intel Xeon E5-2620v3 x 2, 2.4Ghz, 6 cores	1
		Memory: 8G, DDR4, 2133Mhz	
		Nvidia V100 GPU	
		Hard disk: 1.5 TB SATA	
8-route fat node	Inspur TS860	CPU: Intel Xeon E7-8850v2 x 8, 2.3Ghz, 12 cores	1
		Memory: 16G, DDR4, 2133Mhz X 128	
		Nvidia V100 GPU	
		Hard disk: 2.4 TB SAS	
Storage node	4U data node	CPU: Intel Xeon E5-2620v2 x 2, 2.1Ghz, 6 cores	2
		4TB SATA X 18	
Connection	FDR	InfiniBand Mellanox ConnectX-3 HCA card, single port QSFP, FDR IB	1
	GbE switch	10/100/1000Mb/s, 24 ports Ethernet switch	
	Gigabit CAT6 cables	CAT6 copper cable, blue, 3m	
	InfiniBand cable	InfiniBand FDR optical fiber cable, QSFP port, cooperating with the InfiniBand switch for use	

Figure 1: Summary of hardware configuration of our HPC cluster

Item	Name	Configuration
Operating system	Linux	Centos 7
Cluster management software	Inspur	Cluster Engine v3.3
Application development environment	parallel computational environment	MPICH
		MVAPICH2
		OpenMPI
		OpenAcc
		Intel compiler, support C/C++ Fortran
		GNU compiler, support C/C++ Fortran
		CUDA
		Mathematical library (MKL, ACML, BLAS, LAPACK, Scala pack, FFTW)
	Genomics	BWA, Bowtie, Top Hat, GATK

Figure 2: Summary of software configuration of our HPC cluster

1.2 Courses, Training, and Interest Groups

Our supercomputing-related trainings are carried out by the joint education program between Institute of Life Sciences and Chien-Shiung Wu College (Honors) in Southeast University.

1.2.1 Institute of Life Sciences

Institute of Life Sciences is established as a direct department of Southeast University to commit to a high level of scientific research. Institute of Life Sciences contains "Key Laboratory of Developmental Genes and Human Diseases, Ministry of Education", Biology doctoral research Centre, first level subject master's degree of Biology and second level subject doctoral degree of Genetics. The Genetics subject is especially appraised as Professor job-setting subjects and the "Tenth Five-Year", "Eleventh Five-Year" key disciplines of Colleges and Universities in Jiangsu Province by Cheung Kong Scholars of Ministry of Education.

Now the Key Laboratory of Developmental Genes and Human Diseases under Ministry of Education has abundant resource of faculty, including a professor specially invited by Cheung Kong Scholars Program of Ministry of Education, a 973 Chief Scientist, four laureates of China National Funds for Distinguished Young Scientists , 2 awardees of the New Century National Hundred, Thousand and Ten Thousand Talent Project, two professors who receive subsidies from the state council, two of the first batch of young scientific and technological leading talents of Jiangsu Province 333 High-level Personnel Training Project, four excellent youth teachers of Blue Project. In recent years, it obtained construction funds up to more than 20 million RMB from 985 Program and 211 Program.

The laboratory has four stable research fields, which includes the study of development related gene function, the molecular mechanism and therapy of nerve development related diseases. It gained three National Natural Science Funds for Distinguished Young Scholar, more than 20 National Natural Science Foundation of China, a National 973 Foundation including 3 programs, a fund from NIH, a Science Scholarship for The Excellent Youth Scholars of Ministry of Education of China and several provincial research funding. Institute of Life Sciences mainly focuses on high-level scientific research and publishes a number of outstanding scientific papers in life sciences and medical field.

1.2.2 Chien-Shiung Wu College (Honors)

Chien-Shiung Wu College (Honors), named after the world-renowned physicist and also alumna of Southeast University, Madam Chien-Shiung Wu, is the honors college of Southeast University, which provides elite education for outstanding students.The future major of the students cover all engineering majors in Southeast University. Chien-Shiung Wu College provides complete 4-year customize-tailored program including intensive fundamental curriculum in first two years, which is tougher and more challenging. The college provides more training on autonomous study, research-led study, integrated practice, teamwork, communication, together with more emphasis on leadership, global view and international experiences.

Our HPC cluster provides us with the best practical environment for students. We mainly train the students with basic principle of HPC and practical skills. We are teaching the students with a series of courses of Linux, Shell, OpenACC, MPI, CUDA and so on, by which the students can fast grasp the working principle of parallel computing. Subsequent practice on HPC enables the students to increase their capability to resolve practical problems in research.

We encourage the students to do parallel computing in their projects. Especially, at present, the majority of bioinformatic software and programs are designed in a way of sequential program, which usually requires more running time. Parallelizing these sequential programs will enhance analytic efficiency, which is an important orientation

we spend efforts on. The students modify the previous program code and achieve parallelization for many sequential program.

For the excellent students who have strong interests in supercomputing, we establish an interest group and provide them with more training programs. In order to broaden the student horizon, we invite the experts from Tianhe-2 to introduce the latest advance of supercomputing technologies. Even we just start supercomputing training, we believe supercomputing is becoming the most powerful tool to accelerate scientific research and has a broad industrial application. Learning supercomputing will enable the students to have more competitive capability for their future either in research or in industry.

1.3 Research and Applications

Our HPC cluster is the main equipment of our Bioinformatics Systems Biology Platform, which is a core facility that provides bioinformatic support to the Institute of Life Sciences, Southeast University. Our research covers a wide range of biological research fields requiring cutting edge computational techniques such as genomics, proteomics, structure biology and theoretical biology. Our purpose is to assist researchers in the processing, organisation and analysis of biological data, providing insight and aiding scientific discovery for academic partners and industrial collaborators by using HPC.

Data on biological systems is being generated at an unprecedented speed by new high-throughput molecular profiling techniques. Consequently, we are rapidly accumulating information about all aspects of cell function, such as DNA and protein sequences, gene expression levels and its regulations, epigenetic modifications, post-translational modifications, protein-protein interactions, metabolic pathways, protein complexes etc. No doubt, in Big Data Era, bioinformatics and systems biology are rapidly extending their applications in comprehensive biomedical research fields.

Today medicine is now undergoing a transformation of the nature of healthcare from reactive to preventive. The change is rooted in new sciences including computer science. This change will be catalyzed by supercomputing that will trigger the emergence of precision medicine – a medicine that focuses on the integrated diagnosis, treatment and prevention of disease in individual patients. And big data of healthcare, which has revolutionary changed concept of health for everyone. In the not far future, everyone will be benefit from big data of healthcare. Southeast University and her collaborators are participating in big data healthcare project in Jiangbei Industry Zone in Nanjing. Supercomputing is necessary for deep learning for such big data, such as genomic and other biomedical data from million population in Nanjing, or Jiangsu province, even big area. Therefore, there are many opportunities for basic and applied research when supercomputing is applied in Southeast University.

A central focus of our research and application is to use supercomputing and bioinformatic tools to interpret the information produced by such technologies and

identify biologically and medical significance in research, medical practice and healthcare management. These fields, we are working on parallelizing the previous sequential program and empowering our analytic capability to face up to the challenge from the era of Big Data and Precision Medicine.

1.4 Key Achievements on Supercomputing Research

The most exciting achievements on supercomputing is that Southeast University team achieve the first class prize in ASC2017 and ASC2018. Through competition with huge amount of team worldwide, we finally entered in final competition. We achieve a big honor and our University also propaganda our achievements. Since then, more students want to know and learn supercomputering.

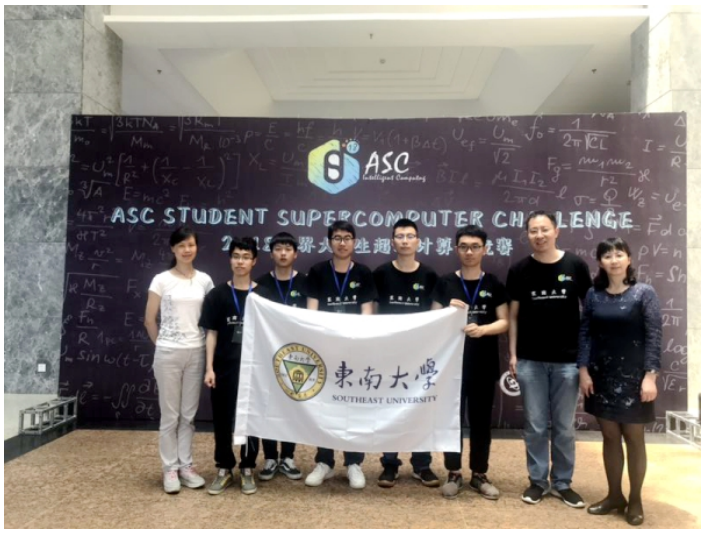


Figure 3: Wining first class prize in ASC 18

Another achievement in research is that we are are carrying two projects in collaboration with BGI: We are using deep learning to analyze omics data such as whole genome sequencing data, whole genome association study data, clinical data, metagenomic data, daily diet information and daily sport activity from thousand individuals to understand the most important reason of obesity for individual. Based on this better understanding, everyone can make a personalized design for better control his or her weight to achieve a better healthy condition. In other project, we are combing genomic data with portrait photo and using deep learning to reveal the relationship between genomic DNA sequence with appearance (facial features). Based on training the model, we hope to use genomic information from a given individual to build up his or her appearance.

Through these interesting and useful projects, the students can obtain more practical skills in supercomputering and deep learning.

2 Introduction of the Team

2.1 Brief description of the Team Setup

Our supercomputing-related trainings are carried out by Chien-Shiung Wu College (Honors) college, School of Artificial Intelligence, School of Computer Science and School of Information in Southeast University.

Our HPC cluster provides us with the best practical environment for students. We mainly train the students with basic principle of HPC and practical skills. We are teaching the students with a series of courses of Linux, Shell, OpenACC, MPI, CUDA and so on, by which the students can fast grasp the working principle of parallel computing. Subsequent practice on HPC enables the students to increase their capability to resolve practical problems in research. We encourage the students to do parallel computing in their projects. Especially, at present, the majority of bioinformatic software and programs are designed in a way of sequential program, which usually requires more running time. Parallelizing these sequential programs will enhance analytic efficiency, which is an important orientation we spend efforts on. The students modify the previous program code and achieve parallelization for many sequential program. For the excellent students who have strong interests in supercomputing, we establish an interest group and provide them with more training programs. Even we just start supercomputing training, we believe supercomputing is becoming the most powerful tool to accelerate scientific research and has a broad industrial application. Learning supercomputing will enable the students to have more competitive capability for their future either in research or in industry.

2.2 Introduction of Team Members

Team advisor:Jian Li

Professor in School of Life Sciences and Technologies, Southeast University, male, 43 years.

I am in charge of bioinformatics platform and HPC. Prior to joining Southeast University in 2014, I was post-doctoral fellow and, later, principle investigator in Department of Biomedicine, Aarhus University (Denmark) from 2008 to 2014. I received bachelor degree of medicine from School of Clinical Medicine, Southeast University in 2001 and PhD

at Human Genetics Institute, University of Aarhus in 2008. My research fields include Bioinformatics Systems of Biology and Genomics.

For the Human Genome Project (HGP), scientists from six countries spent 3 billion US dollars and more than ten years to complete sequencing one human genome. Nowadays, one can sequence a human genome within 24 hours with a cost of less than 1000 US dollars. It means, we are accumulating more and more genomic data at an unprecedented speed. Consequently, we have to evolve new capability to handle such huge data. Supercomputer is a good tool to achieve such goal. In this context, we decided to purchase a high-performance computational cluster to assist our research. No doubt, in Big Data Era, bioinformatics and genomics assisted by supercomputing are rapidly extending their applications in almost every biomedical research field. I am developing and applying bioinformatics tools with HPC to advance our understanding of the mechanisms of cancer and autism. To obtain novel clue from study such complex diseases, usually we need to analyze a large scale of data from a large cohort of patients generated by high throughput technologies. Supercomputer makes analyze such huge data possible. Parallel computational programs further make such analysis faster and more effective. Now, we are using our HPC to carry out the following projects: development of novel algorithms to understand tumorigenesis of skin tumors; meta-analysis of autism associated genes; deepening understands of the human genome; gene networks in cancers; novel molecule drug development.

Captain:Yifan Huang

I am Huang Yifan, an undergraduate student of Computer Science and Engineering College, Southeast University, major in Artificial Intelligence. I am now interested in NLP, an important area in Artificial Intelligence. I'm now conducting a Knowledge Graph based project called Entity Linking, which aims to link entities to knowledge graph.

From my perspective, HPC is a significant element in future development of AI and deep learning and I would like to do some research into combine them together.

Member:Haorui Li

A junior undergraduate student of Chien-Shiung Wu College(Honors), Southeast University, male, 22 years, majoring in computer science.

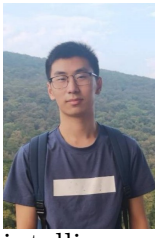
I am not a so-called "traditional" computer-majored student because what I've learned is more like an algorithm engineer, which can be seen in my personal achievement as follows. I've got the Outstanding Winner in Computer Design Competition for Chinese College Students(4C) in

Jiangsu area using a high precision parallel computing weather forecast system; the Meritorious Winner in MCM 2020 where I am in charge of programming, and finally, give correct results by a dynamic programming system for route planning.

During the extra-curricular time, I am in charge of my University's auto-reply robot for searching school information databases which brings me precious experience on load balance, remote server, and highly concurrent because it has to work great under a good number of visit.

The wide range of project experience makes me fit in ASC competition easily. From my perspective, ASC is not only a Computer Challenge but a comprehensive Challenge of many subjects.

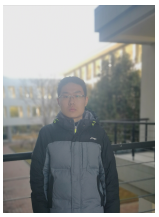
Member:Chengrui Gao



Gao Chengrui, a student in the school of computer science, Southeast University, major in artificial intelligence, male, 19 years old.

I am interested in computer vision and high performance computing. I have participated in a object detection and object tracking project in SEU and got excellent final evaluation. Nowadays, I research on the application of label distribution learning, especially for object counting. I think that high performance computing is necessary in artificial intelligence, astronomy and other basic science categories. With better utilization of high-performance computers, the research efficiency in these fields could be improved.

Member:Chaoyang Wang



I am Wang Chaoyang, male. I am 20 years old, majoring in information engineering. In September 2018, I entered the southeast university.

I have studied computer organization and structure, microcomputer interface program, c + +, digital image processing, has won the H prize in the Mathematical Contest In Modeling.

In my opinion, supercomputer is a competition related to performance optimization and big data computing. Both of these aspects are of great interest to me. In my opinion, big data computing and processing applications are the hot spots for future development. I hope to continue the research on big data and computing clusters in my graduate study.

Member:Jiangtao Wang

Studying at Chien-Shiung Wu College (Honors)Southeast University, majoring in cyber security, aged 20.

At present, I have learned some knowledge about the basic computer, C, data structure, digital logic circuit, computer architecture, and have a strong interest in computers. In the network security major, for cryptology, channel coding theory has a certain interest, a deep understanding of the importance of computational power for the decryption process, therefore, for the super-calculation has a strong interest.

In my opinion, the significance of the computer field lies not only in the wide application of computers, but also in the possibility of computer architecture in a new world, our future fields are by no means a problem of using computers, and more importantly, the development of computers may open new doors for more fields. Academician Zhang Jiping said that the computer field may be the birth of a new mathematics. I think more than that, new science may be born in the computer world. Hope to see and participate in the development of computers and supercomputing.

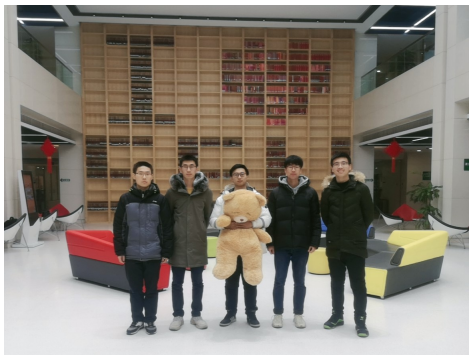


Figure 4: Group photo

3 Design of HPC system

We take the theoretical floating-point operation peak of the computing cluster as the objective function and the power limit as our constraint condition. We hope to get the peak of theoretical floating-point operation under the power constraint of 3000W.

Below we present cluster design on software and hardware configuration and interconnection. On top of that, we analyzed the energy consumption, performance evaluation, and the pros and cons of the cluster.

3.1 Design Description

Our proposed system is based on the Inspur NF5280M5 server following instructions and recommendations. The components and power consumption estimation listed in the table below.

3.1.1 Hardware Design

Item	Name	Configuration	Energy	Quantity
Server	Inspur NF5280M5	CPU Xeon Gold 6132 X2, 2.6GHz, 14 cores (140W x2) Memory: 16G x 12, DDR4, 2666MHz (7.5W)	806.5W	5
		Disk: 1TB SATA x 1 (10W)		
		Accelerator: NVIDIA Tesla Pcle V100 GPU x2 (250W x2)		
HCA card	FDR	Infiniband Mellanox ConnectX®-3 HCA card, single port QSFP, FDR IB (9W)		
Switch	GbE Switch	1000Mb/s，24 ports Ethernet switch	30W	1
	FDR-IB Switch	SwitchX FDR InfiniBand switch, 36 QSFP port	130W	1
Cable	Gigabit CAT6 cables	CAT6 copper cable, blue, 3m	0W	10
	InfiniBand cable	Infiniband FDR optical fiber cable, QSFP port, cooperating with the Infiniband switch for use	0W	5

Figure 5: Hardware configuration of designed HPC system

3.1.2 Software Design

Item	Configuration
Operating System	CentOS 7
Compiler	Intel Compiler XE 2020 gcc 4.8.5
MPI Environment	Open MPI 1.10.2
Math Library	Intel MKL 2020 CuBLAS
GPU Library	CUDA 10.1
Debugger	GNU/Intel IDB Nsight (for Nvidia GPU)

Figure 6: Software configuration of designed HPC system

3.1.3 System Layout

The system layout is as below, black line indicates Gigabit Ethernet running at 1Gbps (for each node one port is used, the other is either as alternative or can be binding for quick speed), green line indicated FDR-InfiniBand at 100Gbps for parallel applications' best performance. The end users and system administrators can connect to the nodes with Ethernet connection.

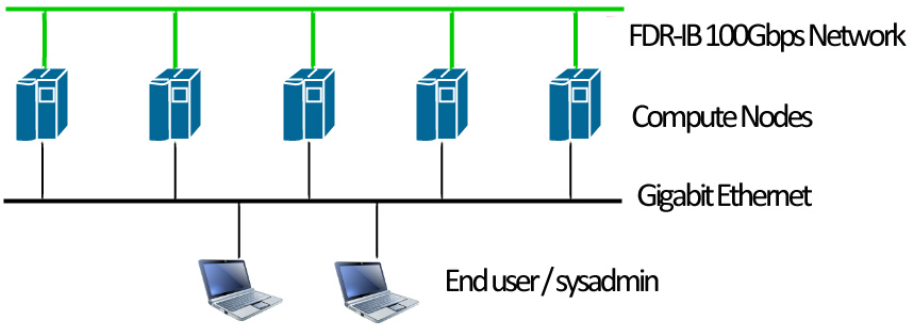


Figure 7: System Layout

3.2 Performance Optimization

Within the power limitation of 3000 Watts, there are usually 4 nodes each with two V100 GPU that can be used. Our strategy is that by reducing the frequency of V100 GPU, the ratio of performance and consumption rises. So, we can add one extra node with two V100 GPU. It can bring an improvement of performance.

Although our theoretical power consumption is nearly 4200w. However, according to the previous experience, the computing performance of the computing cluster cannot be given full play. The actual power consumption of this design will be less than 3000W.

We notice that the CPU is used much more frequently than the GPU in actual computing. Therefore, we assume to increase the number of CPU, but there is a big difference between the floating-point computing power of CPU and GPU.

Item	Peak Performance/TFLOPS	Efficiency/(GFLOPS/W)
CPUx2(280W)	32×2.6GHz×14×2=2.328	8.31
GPUx2(500W)	7×2=14	28

Table 1: Efficiency of CPU and GPU

The advantage of our strategy is clearly that it has greater efficiency than regular systems. However, since we reduce the max frequency of GPU, it can't have the best performance, which shows a low cost-performance ratio.

So we will try the following strategies 1) frequency-reduced GPU only, 2) CPU+GPU to find out the best performance within limit of 3000W.

3.3 Analysis and Discussion

The advantage of our HPC system are as follows:

i According to our existing experience, the actual power consumption of this computing cluster is below 3000w.But the theoretical floating-point operation peak of the cluster can reach 81.64TFLOPS. This is the cluster design method in which our objective function achieves the maximum under the existing conditions.

ii We use node 1 as the management node, but it still acts as both a compute node and a storage node. Data and files are transferred between nodes through NFS. This design maximizes the computational performance of the cluster.

iii Although GPU has good computing performance, some software runs only on CPU. The cluster design takes into account the requirements of CPU and GPU, and the overall performance is superior.

Beside of the advantages, there are several disadvantages:

i We only take the peak value of floating-point operation as the optimization target, but ignore the actual scheduling of GPU and CPU in the process of specific problems. This will be further advanced in our follow-up study.

ii We have two GPU in each node. The performance of two GPU is lower than expected due to some problems in task scheduling as well as other sides.

4 HPL and HPCG

4.1 Environment Description

In this part, We will give a clear description of our hardware and software configuration, including CPU and memory configuration, operating system, math library, etc.

4.1.1 Hardware

Item	Configuration
CPU	Intel(R) Xeon(R) Gold 6240 CPU @ 2.60GHz*2
Memory	128G

Table 2: Hardware configuration

4.1.2 Software

Item	Configuration
OS	CentOS 7.8
Compiler	Intel® Parallel Studio XE 2020
Math Library	Intel® Math Kernel Library 2021.1
MPI	Intel® MPI Library 2021.1
HPL	HPL-2.3
HPCG	HPCG-3.1

Table 3: Software configuration

4.2 HPL Optimization

In this part, We are going to explain the process of HPL optimization from several aspects.

4.2.1 HPL Algorithm

Since our target is to optimize HPL performance, it is crucial to dig in the specific algorithm of HPL, with the help of which we are able to have a deeper understanding of the optimization process and of course, get a better performance.

HPL is a software package that solves a (random) dense linear system $\boldsymbol{Ax} = b$ in double precision arithmetic on distributed-memory computers. It can thus be regarded as a portable as well as freely available implementation of the High Performance Computing Linpack Benchmark.

Firstly, HPL compute the LU factorization of matrix $\boldsymbol{A}$:

$$\boldsymbol{A} = \begin{pmatrix} A \end{pmatrix}_{11} A_{12} A_{21} A_{22} = \begin{pmatrix} L \end{pmatrix}_{11} 0 L_{21} L_{22} \cdot \begin{pmatrix} U \end{pmatrix}_{11} U_{12} 0 U_{22} = \boldsymbol{LU} \tag{1}$$

among which, A_{ij} and L_{ij} are already known, U_{ij} are unknown$(i, j = 1, 2)$. From the equation above we can obtain:

$$A_{11} = L_{11} U_{11} \tag{2}$$

L_{11} and U_{11} can be computed through row partial pivoting. Thus L_{21} and U_{12} can be computed through the equation below:

$$L_{21} = A_{21} U_{11}^{-1} U_{12} = A_{12} L_{11}^{-1} \tag{3}$$

Beside, we can also obtain:

$$A_{22} - L_{21} U_{12} = L_{22} U_{22} \tag{4}$$

As can be seen from the equations above, the calculate of HPL is matrix related computation to a large scale, as a result of which, the optimization process is mainly based on the improvement of matrix computation.

4.2.2 Configuration Parameters

The configuration parameters in hpl.dat is of great significance to the performance result. Therefore, We carried out a great amount of experiments to find the most suitable parameters according to my hardware configuration.

Partition size NB To improve the overall performance, HPL partitions the matrix and then solve it. Therefore, the size of partition matrix plays an important role in the optimization of performance. Considering of this, We carried out several experiments to find the best partition size.

We did a search first——by a range from 2^3 to 2^{10}. To avoid the influence of other parameters, I set them as fixed values. I set problem size N as 113511, by which HPL is about to use 75% of my system's whole memory. The corresponding result can be seen from figure 8.

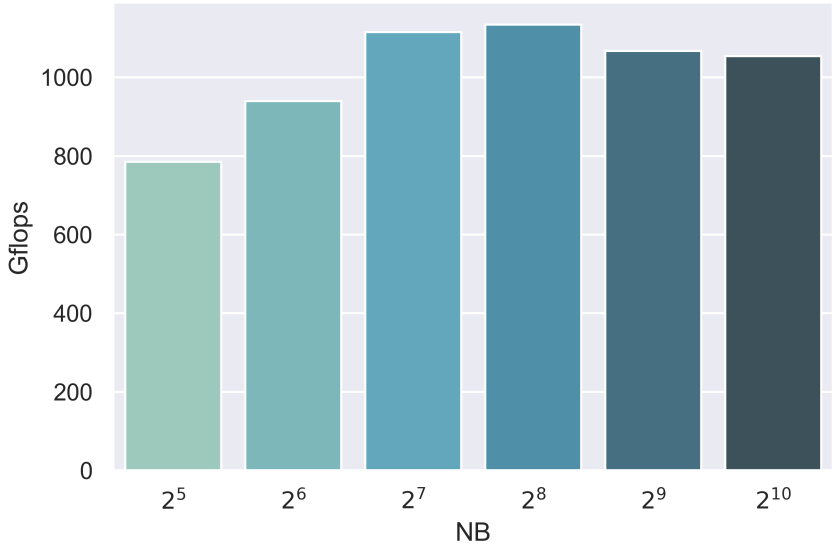


Figure 8: Performance of different NB

It is noticeable that the Gflops increased dramatically since 2^3 to 2^7, after which, it reached a peak at 2^8. Then, there is a steady decrease till 2^{10}. Therefore, we selected $NB = 256$ as the partition size.

Problem size N After deciding the partition size NB, we put emphasis on the problem size. In order to find out the best performance of my system, the largest problem size fitting in memory is what we should aim for.

The amount of memory used by HPL is essentially the size of the coefficient matrix. we definitely need to leave some memory for the OS as well as for other things. As a rule of thumb, 75% of the total amount of memory is a good guess. If the problem size picked is too large, swapping will occur, and the performance will drop; if the size picked is too small, HPL is not able to take advantage of our cluster's whole memory, leading a shrunk performance.

To find the most appropriate size N, we calculated the rough size according to the equation below, which is derived from the rule of thumb.

$$N = \sqrt{\frac{\alpha \cdot \text{TotalMemorySize}}{\text{sizeof(double)}}} \tag{5}$$

TotalMemorySize is the total memory of our cluster, which is 128 GB. The size of double equals to 8 bytes. α represents the ratio HPL used of total memory.

Considering the rule, I took $\alpha = 0.70, 0.75, 0.80$ perspective, the corresponding sizes are 109663, 113511 and 117243. To compare the difference of these sizes, I product the following experiment with different NB sizes.

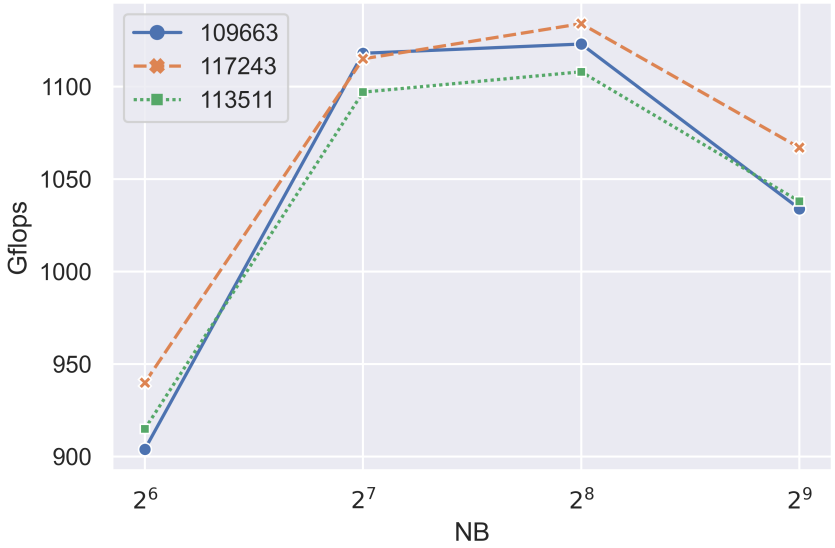


Figure 9: Performance of different N

As the figure shows, size 113511 has the best performance, which proves the rule mentioned before. Although size 109663 exceeds 113511 at a NB size of 2^7, 113511 still performs best from a general view. Therefore, we can decide 113511 as the most appropriate size N.

Process grids $P \times Q$ Referring to some existing researches, the process grid depends on the physical interconnection network. In other words, P and Q should be approximately equal, with Q slightly larger than P. However, If a simple Ethernet network is used, there is only one wire through which all the messages are exchanged. On such a network, the performance of HPL is strongly limited and very flat process grids are likely to be the best choices.

To test and verify these rules, we also conducted an elaborate experiment, with fixed N of 113511 and NB of 256. Considering our cluster has 36 logical CPU, we do the test separately on 5 different P×Q: 3×12, 4×9, 6×6, 9×4 and 12×3.

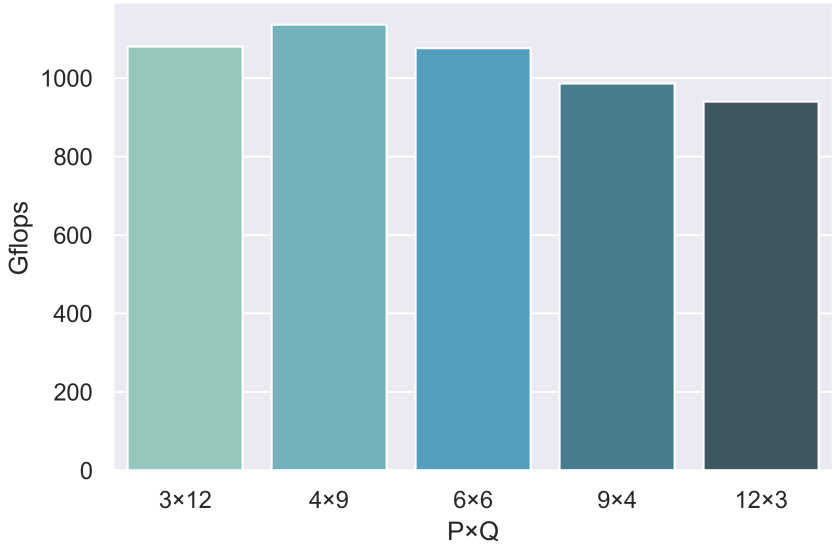


Figure 10: Performance of different P×Q

This experiment shows that the rules are valid: P×Q=4×9 has the best performance, followed by P×Q=6×6. With a smaller P and larger Q, the HPL's performance go through a slight decrease. Therefore, we choose P×Q=4×9 as the final process grids.

4.3 HPCG Optimization

HPCG, which stands for High Performance Conjugate Gradient, is a stand-alone code that measures the performance of basic operations. It's driven by multigrid preconditioned CG algorithm that exercises the key kernels on a nested set of coarse grids. Below we will introduce the optimization of HPCG based on experiments from two aspects.

4.3.1 Testing Time

HPCG can be run in just a few minutes from start to finish. However, official runs must be at least 1800 seconds (30 minutes) as reported in the output file.

To achieve a balance between validity and efficiency, We took 1800s as the run time of HPCG, which is able to get a valid result with acceptable time consumption.

4.3.2 Problem Size

A valid run must also execute a problem size that is large enough so that data arrays accessed in the CG iteration loop do not fit in the cache of the device in a way that

would be unrealistic in a real application setting. Presently this restriction means that the problem size should be large enough to occupy a significant fraction of "main memory", at least 1/4 of the total.

Based on this rule, We choose several problem sizes, trying to make out which can lead to best performance.

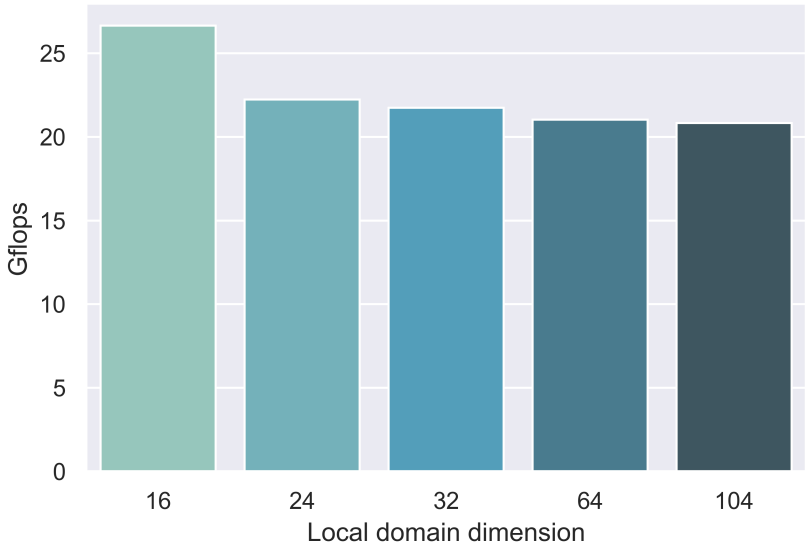


Figure 11: Performance of different problem sizes

The parameter local domain dimension specified by user in hpcg.dat predicts the problem size. The default local domain dimension is 104×104×104, which leads to the worst performance according to the figure above. Higher performance is observed when small problem size is specified. However, values under 16 will be defaulted to 16 (for a 16x16x16 mesh). Therefore, we choose 16 as the local domain dimension.

4.4 Performance Estimation

4.4.1 Theoretical Peak Performance

In order to assess how efficiently HPL and HPCG is running on a given computer, it is useful to know the theoretical peak performance of the computer. By comparing the measured performance of the program, We will then know how many percentage of the theoretical peak are achieved.

The theoretical peak performance can be calculated as followed:

$$\text{peak flops} = \text{processors} \times \text{cores} \times \text{clock speed} \times (2 \times \text{FMA units}) \times \frac{\text{vector size}}{64} \quad (6)$$

where processors is the number of processors that constitute a parallel computer, cores per processor is the number of cores per multi-core processor, clock speed is usually measured in GHz, FMA units is the number of FMA units per core, and the last term is the number of double-precision operands held in each vector register.

According to this equation, the theoretical peak performance of our cluster can be calculated as 1497.6 Gflops.

4.4.2 HPL Performance

Item	Value
N	113511(75% of total memory)
NB	256
P×Q	4×9
Maximum Gflops	1136.16 Gflops
Efficiency	76.87%

Table 4: HPL estimation

4.4.3 HPCG Performance

Item	Value
Testing time	1800s
Local domain dimension	16×16×16
Maximum Gflops	22.092 Gflops
Efficiency	1.48%

Table 5: HPCG estimation

5 Language Exam Challenge

Substantial progress has been made in training context-aware language models. The fact that Google's BERT model and Allen Institute's ELMo are occupying majority of SQuAD 2.0 leaderboard reaffirms their effectiveness as an embedding generator.

In this task, we try to implement and fine-train a model based on BERT to deal with the Language Exam Challenge, which is a very challenging task. After conducting tons of experiments and taking distributed training into consideration, we finally get an accuracy of 85.76% over evaluation dataset.

5.1 Hardware and Software Platform

5.1.1 Hardware Configuration

Item	Configuration
CPU	Intel(R) Xeon(R) Gold 6240 CPU @ 2.60GHz*2
GPU	Tesla V100 SXM2 32GB*8
Memory	512GB

Table 6: Hardware configuration

5.1.2 Software Configuration

Item	Configuration
System	CentOS 7.8
CUDA Toolkit	10.1
cuDNN	7.6.2
Nvidia Accelerator Driver	418.87.00
Python	3.6
Pytorch	1.2.0

Table 7: Software configuration

5.2 Analysis and Approach

This task drives us to teach machine understanding human language documents, to be more specific, doing cloze tests. Cloze tests[7], which means there is a passage and several blanks, each blank corresponds to four similar options, our target is to choose the correct answer to fill in blank from options. Clearly, the big idea of this task is a NLP task. This challenge requires a model with deep language understanding and wide attention span.

To fulfil this Language Exam, we decide to start with a deep neural network. We are going to not only test different models for the best one, but also take distributed training into consideration to obtain a better result. Our code is totally based on Pytorch.

5.3 Distributed Training

Nowadays, deep neural network models contain millions of parameters and need tons of data, which demand high efficiency computation and valid optimization strategy. To

improve the training performance of our model, we adopted some parallelism strategies based on the designed HPC cluster.

5.3.1 Data Parallelism

Data Parallelism is when we split the mini-batch of samples into multiple smaller mini-batches and run the computation for each of the smaller mini-batches in parallel.

In this task, data parallelism is implemented using torch.nn.DataParallel and torch.nn.parallel.DistributedDataParallel. The core code of data parallelism is shown below:

```
os.environ["CUDA_VISIBLE_DEVICES"] = "0,1,2,3,4,5,6,7"
......
if args.local_rank != -1:
    model = torch.nn.parallel.DistributedDataParallel(
    model,
    device_ids=[args.local_rank],
    output_device=args.local_rank)
elif n_gpu > 1:
    model = torch.nn.DataParallel(model, device_ids=[0])
......
```

Since we have 8 GPUs to train this model, the visible GPU IDs should be set to "0,1,2,3,4,5,6,7". The parameter "local_rank" represents whether perform training distributively on GPUs.

This method parallelizes the application of the given module by splitting the input across the specified devices by chunking in the batch dimension. The module is replicated on each machine and each device, and each such replica handles a portion of the input. During the backwards pass, gradients from each node are averaged.

5.3.2 Model Parallelism

Model parallel is widely-used in distributed training techniques. Data parallelism is able to train a neural network on multiple GPUs; this feature replicates the same model to all GPUs, where each GPU consumes a different partition of the input data. Although it can significantly accelerate the training process, it does not work for some use cases where the model is too large to fit into a single GPU.

The high-level idea of model parallel is to place different sub-networks of a model onto different devices, and implement the forward method accordingly to move intermediate outputs across devices.

However, since our GPUs has enough memory size for the target model, we didn't adopted model parallelism in this task

5.4 Experiments and Results

5.4.1 Model Selection

To obtain an accuracy as high as possible, we have to select a suitable model at first. Therefore we modified several deep neural network models for this specific task of language exam, including LSTM, TCN, BERT and ELMo. After reproducing these models, we tested them on the evaluation dataset to choose the best one.

LSTM To test the performance of RNN-based supervised models, we train a bidirectional LSTM to predict the missing words in passages. LSTM[3] is network with loops, allowing information to persist. It can not only process single data points, but also entire sequences of data.

TCN TCN[1] is identified to be a suitable RNN-type structures for tasks including language modeling and music generation. We implemented two parallel TCNs where one of which takes reversed input word vectors. The decoder consists of multiple linear layers.

BERT BERT[2] makes use of Transformer, an attention mechanism that learns contextual relations between words (or sub-words) in a text. In its vanilla form, Transformer includes two separate mechanisms — an encoder that reads the text input and a decoder that produces a prediction for the task. In our adapted BERT model, each word is masked as a 3-length token, then a decoder with a few fully-connected linear layers is used, which turns the 768-length embedding vector into a 30522-length vector. The last layer is a softmax layer which computes the probability for each of the options, and the model chooses the one with the highest probability. For this task, we tried both BERT-base and BERT-large models. Since BERT is such a huge model to train the whole parameters, we downloaded a pre-trained version of it and fine-tune the model with dataset offered by ASC.

ELMo The adaption of ELMo[6] model is much difficult than before cause ELMo uses character-level encoding of words. Thus, we modified the preprocessor and the tokenizer for previous models to supply input to ELMo.

After reproducing these representative models, we trained them on the train dataset and test on evaluation dataset, the overall results are as below:

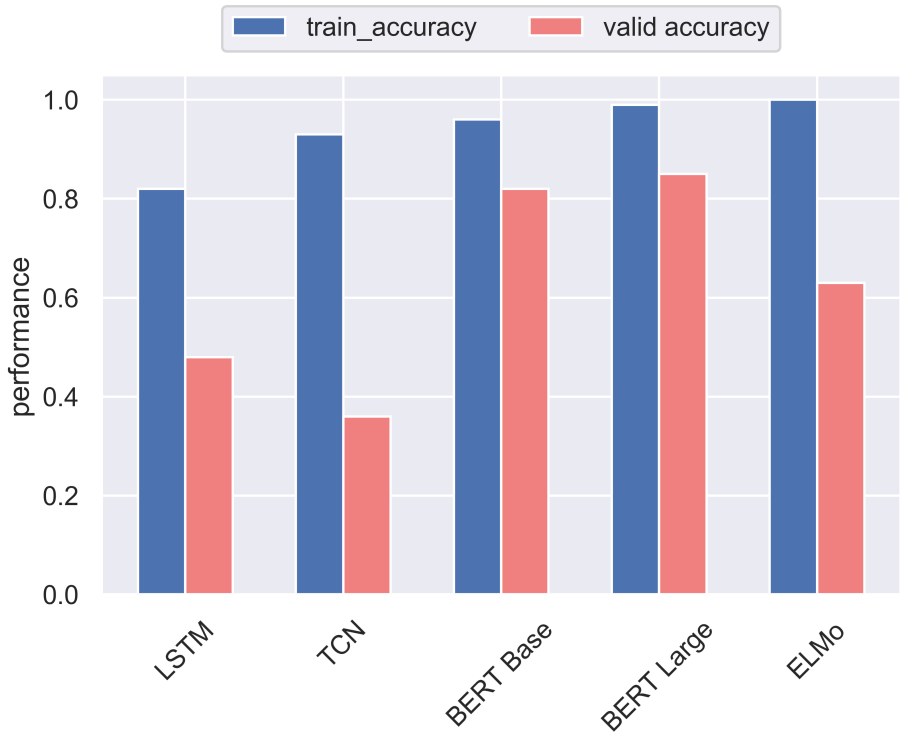


Figure 12: Comparison of different models

As can be seen from the figure, models' performance on training dataset are satisfying. However, when it comes to evaluation dataset, things get different. The BERT-base and BERT-large both perform well, with LSTM and TCN getting less than 50% accuracy. ELMo, though the training accuracy is nearly perfect, the evaluation accuracy is poor.

In summary consideration, we selected BERT-large model as the final choice for this challenge, considering its excellent performance on training and evaluation set.

5.4.2 Batch Size

The batch size defines the number of samples that will be propagated through the network.Choosing a batch size that is too small will introduce a high degree of variance within each batch as it is unlikely that a small sample is a good representation of the whole entire dataset. Conversely, if a batch size is too large, it may not fit in memory of the compute instance used for training and it will have the tendency to overfit the data.

Thus, selecting a well-performed batch size is a challenging work. Taking our overall GPU memory into consideration, we choose three different batch sizes for comparison:

16, 32 and 64. We trained three models for 10 epochs with these batch sizes, and the result can be seen from the figures below.

(a) Training accuracy (b) Validation accuracy

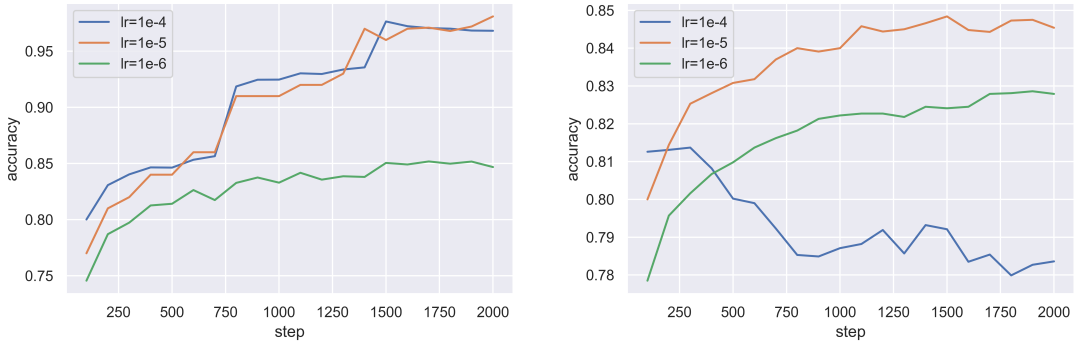
Figure 13: Comparison between different batch sizes

As the figures show, batch size of 32 and 64 almost have the same performance in training set and validation set. However, the batch size of 16 fails to meet our expectation. Although it reach a peak of 98.51% in training set, the performance in validation set is much lower than others.

Therefore, to achieve a balance between memory consumed and performance, we finally selected a batch size of 32 to train our model.

5.4.3 Learning Rate

The learning rate is a hyper-parameter that controls how much to change the model in response to the estimated error each time the model weights are updated. Choosing the learning rate is challenging as a value too small may result in a long training process that could get stuck, whereas a value too large may result in learning a sub-optimal set of weights too fast or an unstable training process.



(a) Training accuracy (b) Validation accuracy

Figure 14: Comparison between different learning rates

As indicated in the figure, a small learning rate of 1×10^{-6} leads to a slow convergent speed in both training set and evaluation set. A large learning rate of 1×10^{-4} performs well in training set, but the accuracy in evaluation set keep going down, which is definitely out of consideration. As a result, we selected a learning rate of 1×10^{-5} for faster convergence and better performance. Finally, we obtained the accuracy of 85.76% over evaluation set.

5.4.4 Data Augmentation

The prediction accuracy of the Supervised Deep Learning models is largely reliant on the amount and the diversity of data available during training. The relation between deep learning models and amount of training data required is analogous to that of the relation between rocket engines (deep learning models) and the huge amount of fuel (huge amounts of data) required for the rocket to complete its mission (success of the deep learning model).

In this task, we are provided with nearly 4000 passages for training. We generate an enhanced dataset aimed for higher accuracy through replacing options (3 incorrect options) by nearest neighbours according to GLoVE[5]. Articles and option indices kept the same. This dataset should be more difficult for the machine to choose the correct answer, cause some incorrect options in the origin dataset are easy to distinguish from the correct one.

GLoVE is an unsupervised learning algorithm for obtaining vector representations of words. Training is performed on aggregated global word-word co-occurrence statistics from a corpus, and the resulting representations showcase interesting linear substructures of the word vector space. Thus, a GLoVE model has the ability to represent words via a n-dimension vectors. In this section, we took advantage of a 300-d pre-trained GLoVE model trained on Wikipedia 2014.

Unfortunately, this elaborate data augmentation didn't make a significance difference, it merely increased the evaluation accuracy by about 0.6 percent. From our perspective, this unsatisfied result may be caused by the nearest neighbours of verbs—There are lots of verbs in options list, but a majority of their generated nearest neighbours are their different tenses. Take "like" as an example, the top-3 nearest neighbours are "liked", "liking" and "love". As a result, the different tenses of one verb are not able to offer useful information.

In the future we are going to explore more efficient ways to perform data augmentation.

5.5 Conclusion and Discussion

We can obtain a high accuracy of BERT to its training method. As discussed before, BERT was trained by recovering masked tokens, which is exactly the cloze test, albeit using a longer text. In fact, forcing BERT model to select 1 out of 4 options only simplified the problem. In contrast, ELMo was trained for predicting the next word, which is not perfectly suited for cloze test.

Lastly, the distributed training plays a important role in our approach. which indicates the significance of HPC. Data parallelism and model parallelism offer us an opportunity to train such a huge model in several GPUs better and quicker. We firmly believe that the combination of HPC and deep learning is going to change the world in a large scale.

6 The QuEST Challenge

6.1 Hardware and Software Platform

6.1.1 Hardware Configuration

Item	Configuration
CPU(Main Frequancy)	Intel Xeon Skylake 6133(2.5GHz)
vCPU	12core
Memory	24GB DDR4 2666Mhz
HardDisk	50G SATA $\times$ 1
GPU	GN10X 2XLARGE40

Table 8: Hardware Configuration

6.1.2 Software Configuration

Item	Configuration	Version
Operation System	Ubuntu	16.04.1 LTS x86
Compiler	CMake	3.5.1
Compiler	GNU	7.4.0
Compiler Directive	OpenMP	4.5
Math Library	Intel MKL	11.1.0.080
Simulator	QuEST	2.1.0

Table 9: Software configuration

6.2 Testing Methods and basic performance

The test method is as follows:

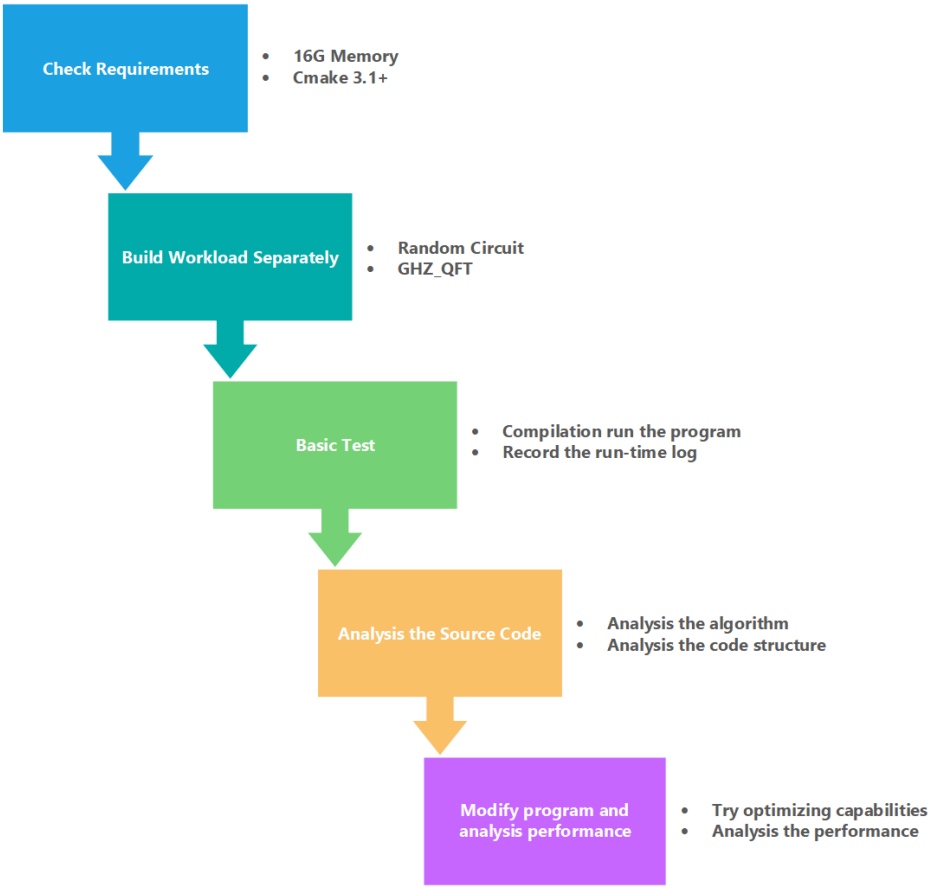


Figure 15: Test and Evaluation Process

At first, we run both .c files in our operating system, and they are random.c and GHZ_QFT.c. The procedures are as figure 16 and 17. At this point the hardware environment is 12 core.

At this chapter, all test methods are similar with the method provided by ASC competition organizing committee, which is submitted in Command line file(*.sh).

It took 333.0482 seconds to calculate the result of the first project named "random circuit":

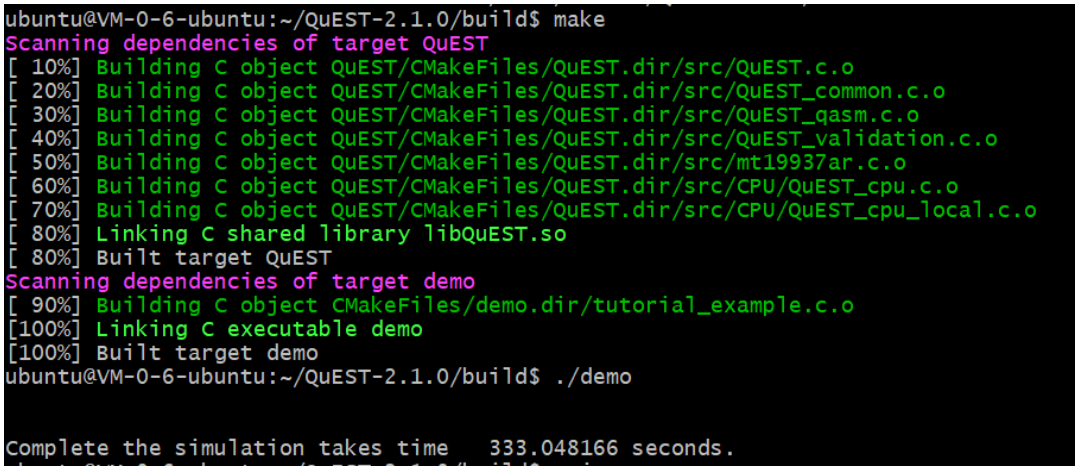


Figure 16: Random Circuit

For the second workload named "GHZ_QFT", it takes 353.1252 seconds to calculate the result.

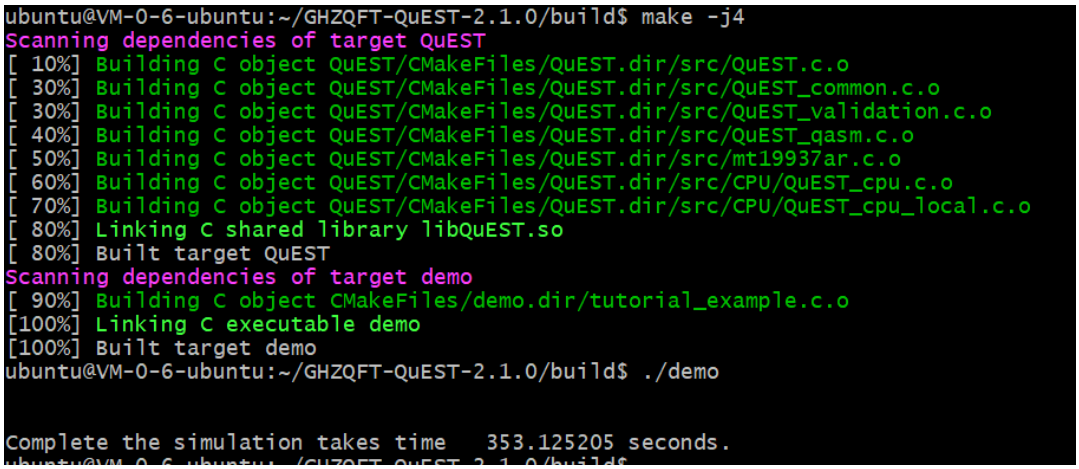


Figure 17: GHZ_QFT

The results of the calculations are found to be identical to the theoretical values

in the annex submitted. While the program running, it takes 16626.833MB CPU Memory.

6.2.1 Problem and Solution Analysis From QuEST's Source Code

After completing the simulation of quantum circuits of 30 qubits by using the provided quantum random circuit (random.c) and the quantum Fourier transform circuits (GHZ_QFT.c), in order to improve the performance and choose parallel strategy, we analysis QuEST's source code and find several optimizable points.

First analysis code for quantum random circuit. The compiling process is complex, so start from the dependence of random.c, which is as follow:

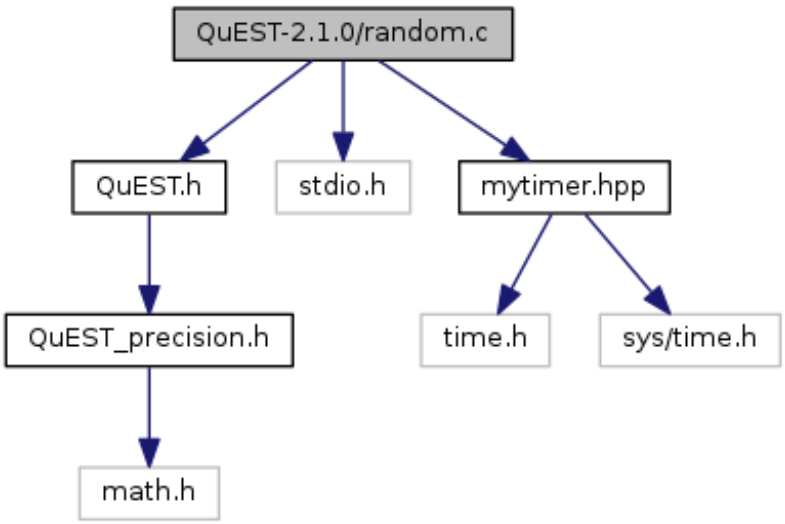


Figure 18: Dependence structure of random.c

According to the requirements from ASC competition organizing committee, we are not allowed to modify random.c, so it is meanful to look into QuEST.h, QuEST_percision.h and so on.

Though the dependency relationship of the above two c language files are complex, which can be seen at appendix A, after analysing the compiling process, we find several loop and nested loop which can be parallel.

First sample is in QuEST_common.c:

Code Listing 1: Loop in QuEST_common.c

```
1 //-------Line 98-101-------
2 void shiftIndices(int* indices, int numIndices, int shift)
      {
3     for (int j=0; j < numIndices; j++)
```

```
4           indices[j] += shift;
5   }
6   }
```

According to QuEST's algorithm[4], change this loop into parallel will not affect results.

And In compiling process, this header file is reused many times, which can be inferred from the following figure:

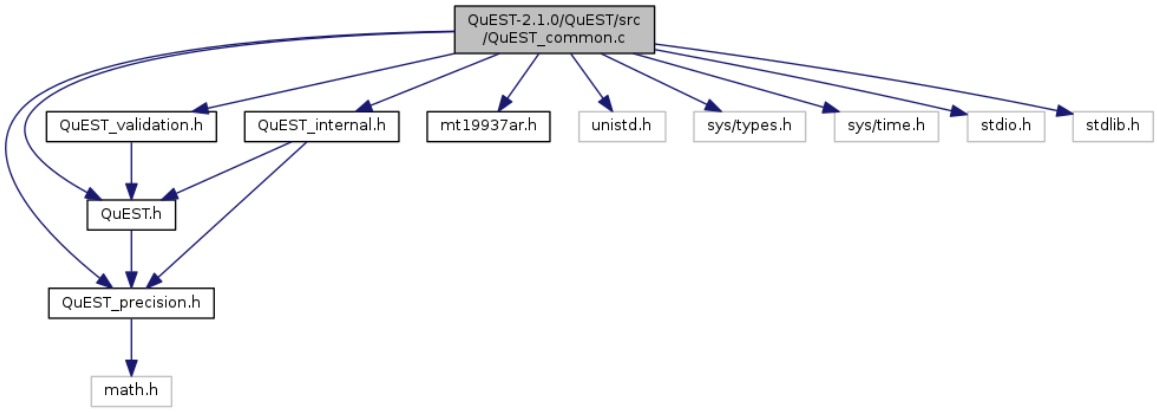


Figure 19: Dependence structure of QuEST_common.c

So implement parallel computing here will increase the performance and not affect test result.

Though in python file, there exists similar nested loop, however, according to QuEST's algorithm, only several loop can be modified into parallel computing. The analysis are as follows.

In utilities/QuESTTest/QuESTCore.py, there exists about five nested loop, but only two of them can be modified without affecting test results. For example:

```
#Class:TestResults, Function:compareStates,line:324-336
 if a.isDensityMatrix and b.isDensityMatrix:
    for row in range(a.numQubitsRepresented):
        for col in range(b.numQubitsRepresented):
            aState = getDensityAmp(a,row,col)
            bState = getDensityAmp(b,row,col)
            if not self.compareComplex(aState,bState,tol):
                return False
else:
    for state in range(getNumAmps(a)):
        aState = getAmp(a,state)
        bState = getAmp(b,state)
        if not self.compareComplex(aState,bState,tol):
            return False
```

Here we can only modify the inner loop, because if row in a.numQubitsRepresented is parallel computed.

Another example is

```
#Class:TestResults, Function:_run_test,line:380-468
for test in testType[0]:
    if test in "Mm":
        expectState = [None]*Qubits.numQubitsRepresented
        success = True
        for qubit in range(Qubits.numQubitsRepresented):
            ... ...
        if not success:
            for qubit in range(Qubits.numQubitsRepresented):
                ... ...
    elif test in "Ss":
        ... ...
        if not success: # Print resultant state vectors
            if not Qubits.isDensityMatrix:
                ... ...
            else:
                for row in range(2**Qubits.numQubitsRepresented):
                    for col in range(2**Qubits.numQubitsRepresented):
                        a = getDensityAmp(Qubits, row, col)
                        b = getDensityAmp(expectState, row, col)
                        self.log('{} {}'.format(a, b))
```

Though there exists many nested loop, but most of commands are written in order to record log, we are supposed to use thread pool to accelerate the computation to avoid out-of-order output.

After analysis on the second workload named "GHZ_QFT", we find GHZ_QFT.c depends on the source codes from QuEST we cited before, so trategies for these files will work for both workloads.

6.2.2 GPU Acceleration and Performance Estimation

In the first project, I use the GPU to accelerate the computation. To compile for GPU, use

Code Listing 2: Bash command to use GPU

```
$ cmake -DGPUACCELERATED=1 -DGPU_COMPUTE_CAPABILITY=70 ..
```

Where COMPUTE_CAPABILITY is the compute cabability of GPU. This can be looked up at the NVIDIA website, for my hardware it is 70.

We complete the simulation for both workloads, and get the right output file which are the exact same as of probs.dat and stateVector.dat, given for reference.

The workloads with random.c takes time 18.361457 seconds, and the other with GHZ_QFT.c takes time 13.974194 seconds, they are both shown in the .log file.

The result shows that with a GPU accelerate, we achieve a very massive speedup on simulation. However, the simulation need a large number memory on GPU, which is the limit of GPU acceleration.

6.2.3 Performance Optimization Methods and Estimations

In order to to implement OpenMP parallel computing, ddd the following code to the file'cmakelist.txt'.

Code Listing 3: Use OpenMP in CMakeList

```
#------OPENMP---------
FIND_PACKAGE( OpenMP REQUIRED)
if(OPENMP_FOUND)
        message("OPENMP␣FOUND")
        set(CMAKE_C_FLAGS "${CMAKE_C_FLAGS}␣${
           OpenMP_C_FLAGS}")
        set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS}␣${
           OpenMP_CXX_FLAGS}")
        set(CMAKE_EXE_LINKER_FLAGS "${
           CMAKE_EXE_LINKER_FLAGS}␣${
           OpenMP_EXE_LINKER_FLAGS}")
endif()
```

After configure OpenMP, it is feasible for us to change .c files into parallel computation. We add the following code in QuEST_common.c's nested loop:

Code Listing 4: Use OpenMP to solve nested loop in c files

```
#pragma omp parallel
```

And for python files, change the loops where will not affect test results using multiprocessing, and use multiprocessing.dummy to creat thread pool as follows:

```
#Class:TestResults, Function:compareStates,line:324-336
from multiprocessing.dummy import Pool
... ...
if a.isDensityMatrix and b.isDensityMatrix:
    for row in range(a.numQubitsRepresented):
        #for col in range(b.numQubitsRepresented):
        bitems=b.numQubitsRepresented
        pool=ThreadPool()
        pool.map(quickgetDensityAmp,a,bitems)
```

As is analysed above, both workloads depend on files depend on these source code, so after modification we re-run two workload to estimate the result.

After modified, it takes 329.989167 seconds for the workload "random circuit" to calculate the results, and the output is same as probs.dat and stateVector.dat given for references.

For the second workload named "GHZ_QFT", it takes 354.532118 seconds to calculate the results, and the output is same as probs.dat and stateVector.dat given for references just as the workload before.

Both workload's run time reduce 2%, though many nested loop arr changed into parallel computing. Our team think the reason of small reduction is the time-consuming communication in RAM, which counteracts the enhancement from parallel computing.

7 The PRESTO Challenge

7.1 Hardware and Software Platform

7.1.1 Hardware Configuration

We use one CPU node to compute the PRESTO Challenge. Our hardware configuration is shown in Table 10. CPU is Intel Xeon Skylake 6146 with 3.2GHz main frequemcy. The number of cores is 32. RAM is 128GB. Intranet bandwith is 8Gbps.

Item	Configuration
CPU(Main Frequancy)	Intel Xeon Skylake 6146(3.2GHz)
vCPU	32core
RAM	128GB DDR4 2666Mhz
Intranet bandwidth	8Gbps
Hard disk	50G SATA × 1

Table 10: Hardware Configuration

7.1.2 Software Configuration

Our software configuration is shown in Table 11. We use CentOS 7(operation system), gcc/g++/gfortran 4.8.5(compiler), PRESTO 3.0.1, and python 3.7.2.

Item	Configuration
Operation System	CentOS 7.8
Compiler(gcc)	4.8.5
Compiler(g++)	4.8.5
Compiler(gfortran)	4.8.5
Python	3.7.2
Numpy	1.19.4
Scipy	1.6.0
Mpi4py	3.0.3
Astropy	4.2
Presto	3.0.1
Tempo	13.101
Glib	2.0
Cfitsio	3.49
Pgplot	5.2
OpenMpi	1.4.1

Table 11: Software configuration

7.2 PRESTO Algorithm

The procedure of PRESTO algorithm contains about 13 steps: Examine data format; Search for RFI. Traverse all signals, determine which intervals are interference and make a mark. Generate a mask file; Make a topocentric, DM=0 time series; FFT the time series. Transform time domain signal to frequency domain by fast fourier transform; Identify "birdies" to zap in searches; Make a zaplist; Make de-dispersion plan; Divide broadband signals into many subbands; Process subbands and splice these results back into a broadband signal, which can decrease computational complexity; de-disperse; Search the data for periodic signals. Traverse signals, find periodic signals; Search the data for single pulses; Sift through the candidates; Fold the best candidates; Start timing the new pulsar.

We pay attention to the several most time consuming steps, including de-dispersion, FFT, searching for periodic signals and folding. De-dispersion plan divides the entire de-dispersion task into many subtasks, which can be computed parallel. FFT, searching and folding all need to traverse many files and process them serially. These steps can be parallel computed according to our section 7.4.

7.3 Testing Method

We test our parallel script on the given two datasets, GBT_Lband_PSR.fil and Dec+1554_arcdrift+23.4-M12_0194.fil. We compare our search result with the given

pipeline.py, making sure that they are the same. And we record the running time of our parallel script. Comparing our running time with benchmark, the running time of the given pipeline.py, we can evaluate the efficiency improvement of our parallel method.

The test command line is shown below, which calls the run.sh in our submission file. The my_pipeline.py is our parallel python script for presto.

```
#test on GBT_Lband_PSR.fil
bash run.sh GBT_Lband_PSR.fil
#test Dec+1554_arcdrift+23.4-M12_0194.fil
bash run.sh Dec+1554_arcdrift+23.4-M12_0194.fil
#content of run.sh
rm -rf subbands
(time python ./my_pipeline.py ${1} ${2}) > log.
   pulsar_search 2>&1
```

The final search result should be compared with standard pulsar catalogue, like https://www.atnf.csiro.au/research/pulsar/psrcat/ provided by Australian National Observatory. Match the parameters of candidates in cands.txt with pulsars in pulsar catelogue and judge whether they are pulsars by the bias between parameters. If the bias is low enough, we can decide that our search result are real pulsars.

7.4 Parallel Strategy

Our parallel strategy is data parallel. We utilize parallel computing to avoid serial loops. For loop code blocks, if the computation of every iteration doesn't affect each other, we can speed them by distributing all tasks to multiple processes and computing them parallel.

We notice that there are obvious serial loops in the computation procedure of presto, which can be optimized by parallel computing. In presto pipeline, there are about 13 steps, such as examining data format, de-dispersion and so on, while some of them are not effective and necessary. According to the pipeline.py provided by ASC20-21 Preliminary Round Notification, the required steps are examining data format, making de-dispersion plan, de-dispersion, searching for periodic signals, sifting through the candidates and folding the best candidates. We find that there are serial loops in steps of de-dispersion, fft, searching and folding, as shown in following example code. Taking fft for example, a loop is applied to traverse all datfiles and fft every single datfile is independent from each other. Therefore, we can fft every datfile parallel using multiple processes.

```
#the fft step in the given pipeline.py
for df in datfiles:
    fftcmd = "realfft %s" % df
```

```
    print(fftcmd)
    output = getoutput(fftcmd)
    logfile.write(output)

#the search step in the given pipeline.py
for fftf in fftfiles:
    searchcmd = "accelsearch -zmax %d %s"  % (zmax, fftf)
    print(searchcmd)
    output = getoutput(searchcmd)
    logfile.write(output)
```

We use packages, mpi4py and mulitiprocessing, to implement parallel computing. Mpi4py is a standard message passing interface package for python and the effectiveness of message-passing has been proved. So mpi4py is a good choice for parallel computing in our python script. What's more, the multiprocessing package provides convenient API to produce processes for parallel computing and it utilizes subprocesses instead of threads to bypass the global interpreter lock effectively, which is also suitable for parallel computing in python. Baesd on these two packages, we make some computational steps parallel, including de-dispersion, fft, searching and folding.

For de-dispersion step, we use message-passing to implement data parallel computing. The ddplan.py in presto provides de-dispersion scheme, which is to divide broadband signal to several subbands and guide us to compute de-dispersion on subbands. We learn that subbands are totally divided and independent which are suitable for a data parallel strategy. Therefore, instead of computing subtasks in a serial loop, we collect all subbands in the root process and equally distribute subbands to other processes through broadcast.

The data parallel strategy is shown in the following python script, which is a fragment of the dd_MPI.py in my submission file. First, we define MPI.COMM_WORLD , get the number of prosesses and give them ranks. Second, the root process receives the de-disperion scheme stored in the list variable dmlist and broadcast the data of subbands to other processes. Third, the task of every process is distributed equally and then execute commands parallel to complete de-dispersion.

For folding step, we take a similar data parallel strategy using mpi4py, which is done by the folding_mpi.py in our submission file.

```
#parallel compute de-dispersion
comm = MPI.COMM_WORLD
size = comm.Get_size()
rank = comm.Get_rank()
numjobs = len(dmlist)
# the collection of jobs
job_content = []
for i, dm in enumerate(dmlist):
    lodm = dm[0]
    subDM = np.mean(dm)
```

```
    job_content.append((lodm, subDM))

# arrange the works and jobs
if rank == 0:
    # this is head worker
    # jobs are arranged by this worker
    job_all_idx = list(range(numjobs))
else:
    job_all_idx = None

job_all_idx = comm.bcast(job_all_idx, root=0)
njob_per_worker = int(numjobs / size)
this_worker_job = [job_all_idx[x] for x in range(rank *
                                    njob_per_worker, (rank + 1) *
                                    njob_per_worker)]

# map the index to parameterset [eps,anis]
work_content = [job_content[x] for x in this_worker_job]
#execute subband preprocessing on each process
for lodm, subDM in work_content:
    o_1 = my_prepsubband_1(subDM)
    log.write(o_1)
    o_2 = my_prepsubband_2(lodm, subDM)
    log.write(o_2)
```

For fft and search steps, we use multiprocesses package to implement data parallel. First, we define a function runcmds() to represent the same execution on every input file. Then we use the threads API to map commands to the function runcmds() with just one statement. In this way, we can implement data parallel computing by multiple processes.

```
#parallel compute fft
c = ''
def runcmds(cmds):
    traceback.print_exc()
    output = []
    for cmd in cmds:
        output.append(getoutput(cmd))
    return c.join(output)

commands = []
threads = Pool()
commands.clear()
for df in datfiles:
    fftcmd = "realfft %s" % df
    print(fftcmd)
    commands.append([fftcmd])

logfile.write(c.join(threads.map(runcmds, commands)))
```

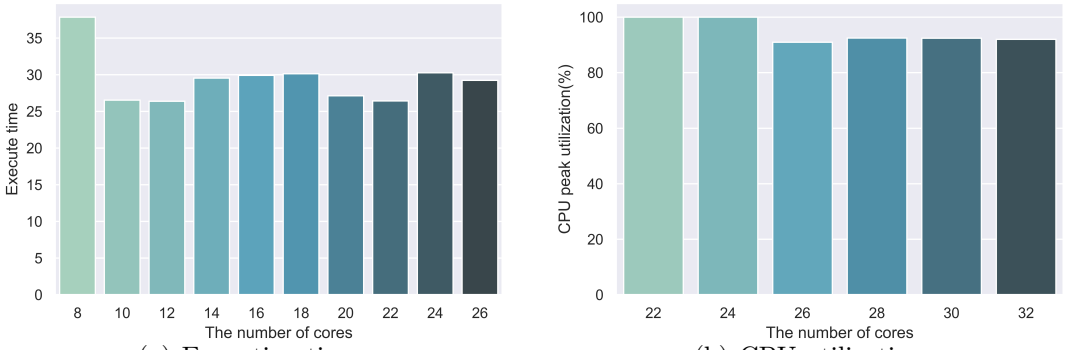
```
19 threads.map(runcmds, commands)
20 threads.close()
21 threads.join()
```

The complete computation procedure is done by my_pipeline.py. This script calls dd_MPI.py and fold_mpi.py to compute de-dispersion and folding the best result parallel. And it use the multiprocesses module to compute fft and searching parallel. Then the four most time consuming steps are all computed parallel using data parallel strategy.

7.5 Performance Optimization

In this subsection, We test our parallel script with different number of cores, aiming to find the best core number for our parallel strategy. We use two metrics, execution time and CPU utilization. We use $top command to monitor CPU utilization.

We change the number of cores from 8 to 32, recording execution time and CPU utilization. When monitoring CPU utilization, we find that there are two peaks of CPU utilization. The highest peak achieves 100% for all numbers of cores. And the second highest peak achieves 100% when the number of cores is smaller than 24. The execution time achieves the best at about 22 cores.



(a) Execution time (b) CPU utilization

Figure 20: The change of execution time and CPU utilization.

As shown in Figure 20(a), the execution time of our parallel script fluctuates with changes of the number of cores. When the number is smaller than 10, the execution time soar highly because there aren't enough processes for parallel computing. Although the execution time may fluctuate randomly by one to two seconds, there is still a minimum about 25.434s at 22 cores. The execution time at 12 cores also looks short but it is about 1.0 second longer than the minimum.

As shown in Figure 20(b), the peak value of CPU utilization is always bigger than 90%, which reflects that our parallel strategy could take full advantage of hardware computing power. It also reflects that when the number of cores is bigger than 24, the CPU computing power is not fully used. And the execution time from 26 cores to 32 cores hardly changes. So using less than 26 cores has the best computing power utilization.

7.6 Performance Estimation

In this subsection, we compare our total execution time and step execution time with benchmark(pipeline.py), under the premise of ensuring the correct pulsar searching result. All experiments are executed on our CPU node described in section 7.1 and we use all 32 cores for parallel computing.

We compare our total execution time with pipeline.py on two given datasets, as shown in Table 12. On GBT_Lband_PSR dataset, our execution time is 11.292s and pipeline.py consumes 72.448s, where our parallel script saves 84.4% execution time. On Dec+1554_arcdrift+23.4-M12_0194 dataset, our execution time is 30.182s and pipeline.py consumes 431.096s, where our parallel script saves 93.0% execution time.

DataSet	Execution time	Execution time(parallel)	Time reduced
GBT...	72.448s	11.292s	84.4%
Dec...	431.096s	30.182s	93.0%

Table 12: Comparison between our parallel script and the given pipeline.py

We compare our step execution time with pipeline.py on the second dataset, as shown in Figure 21. In de-disperision step, our parallel script consumes 8.64s, saving 87.0% execution time comparing with 66.25s(serial execution time). In FFT step, it consumes 0.89s, saving 93.4% time comparing with 24.34s. In search step, it consumes 7.18s, saving 95.3% time comparing with 152.57s. In folding step, it consumes 13.48s, saving 92.8% time comparing with 187.94s.

The comparison result reflects that our parallel script greatly optimizes the execution time of presto. It is reasonable that data parallel strategy is a efficient method for optimizing presto, since there are many serial data processing that could be processed parallel. What's more, our parallel script has the same number of search result to the given pipeline. They both have 11 candidates in GBT_Lband_PSR dataset and 36 candidates in Dec+1554_arcdrift+23.4-M12_0194 dataset. And the final prefolding result of them is correct according to pulsar catalogue.

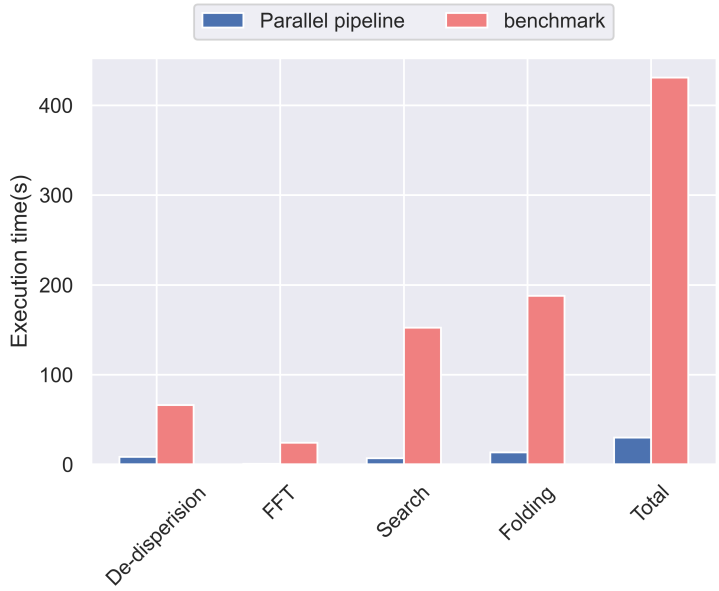


Figure 21: Comparision between parallel script and benchmark.

7.7 Problem and Solution Analysis

The problem we meet is that when specifying certain number of processes, the result would be wrong. As we change the number of processes used in dd_MPI.py and fold_mpi.py, the candidates become fewer, fftfiles and datfiles become fewer too. We first think that our parallel scripts have a bug. But when the number of processes is set appropriately, our scripts still run without exception.

When we looking up our parallel code, we realize that the number of processes must be an integer multiple of the number of tasks. As shown in the following code block, we equally distributed our tasks to processes. When the tasks couldn't be equally distributed, some tasks would be ignored by our parallel script. The easiest solution of this problem is to use the number of tasks as the number of processes. And we can set more CPU cores to ensure our efficiency. If we manage to divide every single task and utilize parallel computing for subtasks, then we could set more processes to tackle this problem.

```
1 #the statement in parallel scripts that should be mentioned
2 njob_per_worker = int(numjobs / size)
3
4 #the proper setting of the number of processes
5 pros = len(dmlist)
```

Acknowledgement

We thank for supercomputering courses and training organized by Chien-Shiung Wu College, especially, Professor Yinghui Kuang for her great support.

Thank for our teachers Yuan Li, Hui Zhon, Wei Xie, Xiaobing Shen, and Chunhong Xu. We also thank for the volunteers: Qi sheng Huang, Hongyu Gong, Jing Zuo, Kaiyu Xue for their kind help. Finally, we are appreciated for great support from Southeast University.

References

[1] Bai, S., Kolter, J. Z., and Koltun, V. An empirical evaluation of generic convolutional and recurrent networks for sequence modeling, 2018. cite arxiv:1803.01271.

[2] Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)* (Minneapolis, Minnesota, June 2019), Association for Computational Linguistics, pp. 4171–4186.

[3] Hochreiter, S., and Schmidhuber, J. Long short-term memory. *Neural computation 9*, 8 (1997), 1735–1780.

[4] Jones, T., Brown, A., Bush, I., and Benjamin, S. C. QuEST and high performance simulation of quantum computers. *Scientific reports 9*, 1 (2019), 1–11.

[5] Pennington, J., Socher, R., and Manning, C. D. Glove: Global vectors for word representation. In *EMNLP* (2014), vol. 14, pp. 1532–1543.

[6] Peters, M. E., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K., and Zettlemoyer, L. Deep contextualized word representations, 2018. cite arxiv:1802.05365Comment: NAACL 2018. Originally posted to openreview 27 Oct 2017. v2 updated for NAACL camera ready.

[7] Xie, Q., Lai, G., Dai, Z., and Hovy, E. H. Large-scale cloze test dataset created by teachers. In *EMNLP* (2018), E. Riloff, D. Chiang, J. Hockenmaier, and J. Tsujii, Eds., Association for Computational Linguistics, pp. 2344–2356.

A Appendix of QuEST

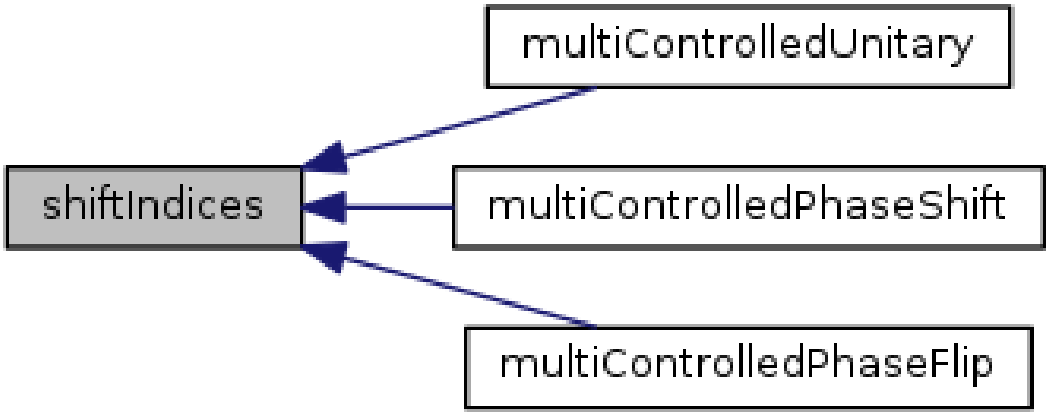


Figure 22: Function call relation

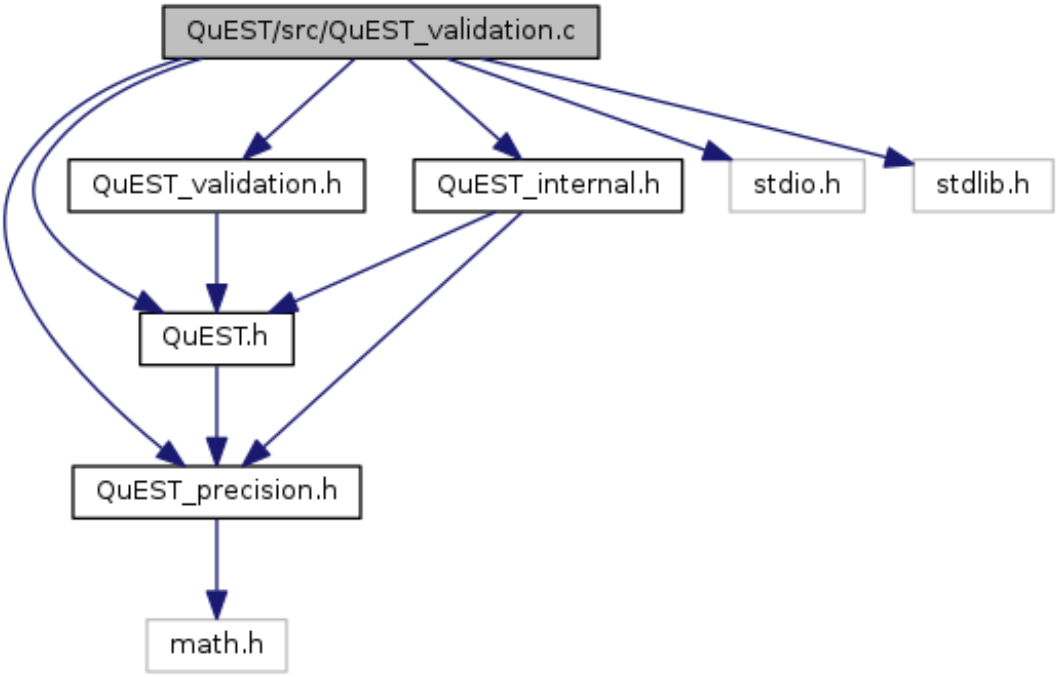


Figure 23: Dependence structure of QuEST_validation.c

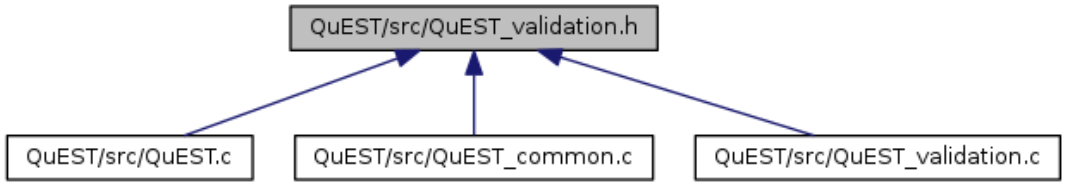


Figure 24: Dependence structure of QuEST_validation.H

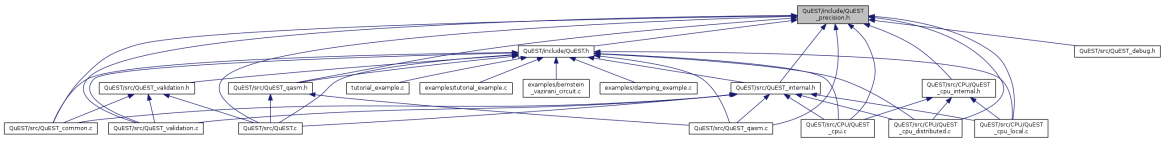


Figure 25: Dependence structure of QuEST_precision.h

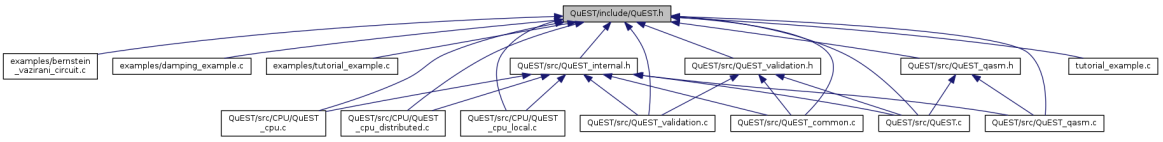


Figure 26: Dependence structure of QuEST_precision.h

B Appendix of Presto

B.1 Presto Dependencies Installation

Code Listing 5: Commands for installing presto denpendencies

```
1  # FFTW Installation
2  $ tar -zxvf fftw-3.3.8.tar.gz
3  $ cd fftw-3.3.8.tar.gz
4  $ ./configure --enable-shared --enable-single --prefix=/
     home/astrosoft/fftw
5  $ make
6  $ make install
7
8  # PGPLOT Installation
9  $ cd /home/download
10 $ wget ftp://ftp.astro.caltech.edu/pub/pgplot/pgplot5.2.
     tar.gz
11 $ tar -zxvf pgplot5.2.tar.gz
```

```
$ cd /home/astrosoft
$ mkdir pgplot
$ cd pgplot
$ cp -r /home/download/pgplot/drivers.list .
$ vim drivers.list
In this step, we delete the exclamation point before PS/VPS/VCPS/XWINDOWS to make them effective. After that, we shall modify makefile.
$ /home/download/pgplot/makemake /home/download/pgplot linux g77_gcc
$ vim makefile
$ (change from FCOMPL = g77 to FCOMPL = gfortran)
We notice that the installation of PGPLOT relies on lib X11, thus we execute the following command.
$ yum install libX11-devel
$ make
$ make cpg

# TEMPO Installation
$ git clone git://git.code.sf.net/p/tempo/tempo
$ yum install csh
$ cd tempo/
$ ./prepare
$ ./configure prefix =/home/astrosoft/tempo
$ make \&\& make install

Glib Installation
$ yum install glib2-devel

# Cfitsio installation
$ tar -zxvf cfitsio-3.49.tar.gz
$ cd cfitsio-3.49
$ mkdir /home/astrosoft/cfitsio
$ ./configure --prefix=/home/astrosoft/cfitsio
$ make
$ make install

# Modify the environment configure.
$ vi .bashrc
ADD
```

```
PATH=$PATH:$HOME/bin:/usr/local/bin:/home/astrosoft/pgplot
   /bin
LD_LIBRARY_PATH=/home/astrosoft/pgplot:
/home/astrosoft/fftw/lib:/home/astrosoft/cfitsio/lib
C_INCLUDE_PATH=/home/astrosoft/cfitsio/include:/home/
   astrosoft/fftw/include
PKG_CONFIG_PATH=/home/astrosoft/cfitsio/lib/pkgconfig
:/home/astrosoft/fftw/lib/pkgconfig:/usr/lib64/pkgconfig
PGPLOT_DIR=/home/astrosoft/pgplot:/usr/local/lib
PGPLOT_FONT=/home/astrosoft/pgplot/grfont.dat
PGPLOT_DEV=/Xserve
PGPLOT_LIB="-L /usr/lib64 -lX11 -L /home/astrosoft/pgplot
   -lpgplot"
TEMPO=/home/download/tempo
export PATH
export LD_LIBRARY_PATH
export C_INCLUDE_PATH
export PKG_CONFIG_PATH
export PGPLOT_DIR
export PGPLOT_FONT
export PGPLOT_DEV
export PGPLOT_LIB
export TEMPO
$ source .bashrc
```

B.2 Presto Installation

Code Listing 6: Commands for installing presto

```
$git clone git://github.com/scottransom/presto.git
$cd presto
$git pull
$cd src
$make makewisdom
$make prep
$make
$cd $PRESTO
$pip install --user .
$python setup.py build
$python setup.py install
```

```
$vi .bashrc
# .bashrc
PATH=$PATH:$HOME/bin:/home/astrosoft/presto/bin:/usr/local
   /bin:/home/astrosoft/pgplot/bin:/home/astrosoft/presto/
   bin:/home/astrosoft/optimus:/home/astrosoft/fv:/home/
   astrosoft/psrcat_tar:/home/download/tempo/src/:/usr/
   local/lib/openmpi/bin
LD_LIBRARY_PATH=/home/astrosoft/presto/lib:/home/astrosoft
   /pgplot:/home/astrosoft/fftw/lib:/home/astrosoft/cfitsio
   /lib:/usr/local/lib/openmpi/lib
C_INCLUDE_PATH=/home/astrosoft/presto/include:/home/
   astrosoft/cfitsio/include:/home/astrosoft/fftw/include
PKG_CONFIG_PATH=/home/astrosoft/cfitsio/lib/pkgconfig:/
   home/astrosoft/fftw/lib/pkgconfig:/usr/lib64/pkgconfig
PYTHONPATH=/home/astrosoft/presto/lib/python
PGPLOT_DIR=/home/astrosoft/pgplot:/usr/local/lib
PGPLOT_FONT=/home/astrosoft/pgplot/grfont.dat
PGPLOT_DEV=/Xserve
PGPLOT_LIB="-L␣/usr/lib64␣-lX11␣-L␣/home/astrosoft/pgplot␣
   -lpgplot"
PRESTO=/home/astrosoft/presto
TEMPO=/home/download/tempo
PSRCAT_FILE=/home/astrosoft/psrcat_tar/psrcat.db
FFTW=/home/astrosoft/fftw

export PATH
export LD_LIBRARY_PATH
export C_INCLUDE_PATH
export PKG_CONFIG_PATH
export PYTHONPATH
export PGPLOT_DIR
export PGPLOT_FONT
export PGPLOT_DEV
export PGPLOT_LIB
export PRESTO
export TEMPO
export PSRCAT_FILE
export FFTW
$source .bashrc
```