EasyChair Preprint

№ 10529

Evaluating the Performance of Parallel Computing in Hybrid Models

Yongyu Chen

July 10, 2023

Saint Petersburg Electrotechnical University
(LETI)

Direction	09.04.01 Computer Science and Engineering
Profile	Computer Science and Knowledge Discovery (In English)
Faculty	Computer Science and Technology
Department	Information Systems

Accepted for defense
Head of the department Tcehanovsky V.V.

MASTER'S GRADUATE QUALIFICATION WORK

THEME: "EVALUATING THE PERFORMANCE OF PARALLEL COMPUTING IN HYBRID MODELS"

Student _____ Yongyu Chen
Signature

Scientific advisor PhD
_____ Alexey Paznikov
Signature

Advisors PhD _____ O.U. Syrovatskaya
Signature

PhD _____ S. S.Egorov
signature

PhD _____ N.A. Nazarenko
signature

Saint Petersburg
2022

SUMMARY

Explanatory note 58 p., 35 fig., 15 tab., 1 app..

MPI, MPI HYBRID PROGRAMMING, OPENMP, PTHREAD, PARALLEL COMPUTING MODEL, LOGP, LOGGP, PRAM

The subject of the research is: Evaluating the performance of parallel computing in hybrid models.

The target of the GQW – EVALUATING THE FINE-GRANIED MPI PROGRAMMING PERFORMANCE, DESIGN A NEW MODEL OF PARALLEL COMPUTING.

Parallel computer nodes need to use MPI communication, and parallel internal nodes need OpenMP, Pthread and other libraries. The combination of the two is called MPI hybrid programming.

On the premise of ensuring thread safety, we will introduce some mutex locks. But these locks will lead to thread contention, which undoubtedly increases the running time of the program. One of the versions of MPI, MPICH, provides some fine-grained program partitioning. In theory, the contention time can be reduced. Therefore, our research is mainly based on the thread-safe MPICH+Pthread parallel computing library of various granularities, to explore the factors that affect the performance of parallel systems.

Our experiments are done on NSU's computing cluster and get completely opposite results to some of the previous papers. We propose a new LogP-based computing model, LogPCK, with two additional parameters that should be considered when designing parallel programs. Hope it can bring a little help to parallel programmers.

ABSTRACT

This paper is mainly a summary of two years of my research. I will start from the basic theory of the book and continue to the latest research. My first year of research was evaluating the performance of hybrid MPI programming. The second year is to design a better parallel computing model.

I analyzed all the papers published so far on MPI hybrid programming, the LogP model, the LogGP model and their variants. Gain a better understanding of the history, hotspots and difficulties of research in this field.

I also sorted out the theory and hardware aspects of the entire parallel computing. Know why the computer develops like this, and what is the bottleneck of development. Lay the foundation for the introduction of the following research.

In terms of MPI fine-grained hybrid programming, I have conducted some experiments on supercomputing systems and found that the performance of MPI fine-grained is not necessarily faster, which is contrary to the opinions of some papers.

I also compared the performance of OpenMP and Pthread in certain scenarios and found that OpenMP runs faster, although Pthread is a more low-level operation.

I summarize all variants of LogP and LogGP, and based on my experiments, propose two parameters that should be considered for fine-grained lock contention problems.

TABLE OF CONTENTS

	Introduction	8
1.	The modern state of the problem under study	9
1.1.	Literature reviews	9
2.	Foundations of supercomputing	12
2.1.	Introduction to supercomputers	12
2.2.	Rankings	13
	2.2.1. TOP500	13
	2.2.2. GREEN500	14
	2.2.3. Famous benchmarks	14
	2.2.4. TOP1 SUPERCOMPUTER FUGAKU	16
	2.2.5. CPU OF FUGAKU – FUJITSU A64FX	16
	2.2.6. Tofu interconnection	16
3.	Theory of parallel computation	19
3.1.	Date parallel model	19
	3.1.1. Instruction-level parallelism	19
	3.1.2. Thread-level parallelism	20
	3.1.3. Single Instruction Multiple Data	21
	3.1.4. Multiple Instruction Multiple Data	21
	3.1.5. Morre’s law	22
	3.1.6. Dennard scaling	23

3.1.7. The real Morre's law – development of RAM 24

3.2. Parallel computing models 25

3.2.1. PRAM model 25

3.2.2. Speedup ratio – Amdahl's law 26

3.2.3. Gustafson's law and differences 27

3.2.4. LogP model 28

3.2.5. Variants of Logp model 28

3.2.6. LogPCK – a new logP model 29

3.3. Communication and broadcast 29

3.3.1. Binomial Tree Broadcast 30

3.3.2. Pipelined binary tree broadcast – Fibonacci tree 31

4. MPI Hybrid programming 32

4.1. Introduction to MPI, OpenMP, Pthread 32

4.2. Hybrid MPI programming 32

4.2.1. Summary and Conclusion of MPI + OpenMP 34

4.2.2. Comparisons of MPI and OpenMP 35

4.2.3. Related Works 36

4.2.4. Thread Safety in MPI 37

4.2.5. Critical Section Granularity 37

4.2.6. Performance Evaluation 38

4.2.7. Performance with blocking send 39

4.2.8. Performance with non-blocking send 39

4.2.9. Performance with diffrent per-vci modes 41

4.2.10. Process performance vs Thread performance 44

4.2.11. Mutex counting and time measurement 45

Conclusions 49

Future plan 50

Acknowledgements 51

Bibliography 52

DEFINITIONS, DESIGNATIONS AND ABBRIVIATIONS

The present explanatory note uses the following abbreviations and designations:

MIMD – Multiple instruction multiple data

MPI - Message Passing Interface

OpenMP - Open Multi-Processing

Pthread - POSIX Threads

SIMD - Single Instruction Multiple Data

INTRODUCTION

Supercomputers are widely used in scientific computing, such as computational fluid dynamics, numerical weather prediction, large-scale equation solving, physical simulation, etc.

Since the development of computers, many breakthroughs have been made. But it is currently facing difficulties. Current quantum computers are of no use at all. Parallelization is an important direction to improve computer performance at present.

Parallel computer nodes need to use MPI communication, and parallel internal nodes need OpenMP, Pthread and other libraries. The combination of the two is called MPI hybrid programming.

On the premise of ensuring thread safety, we will introduce some mutex locks. But these locks will lead to thread contention, which undoubtedly increases the running time of the program. One of the versions of MPI, MPICH, provides some fine-grained program partitioning. In theory, the contention time can be reduced. Therefore, our research is mainly based on the thread-safe MPICH+Pthread parallel computing library of various granularities, to explore the factors that affect the performance of parallel systems.

Our experiments are done on NSU's computing cluster and get completely opposite results to some of the previous papers. We propose a new LogP-based computing model, LogPCK, with two additional parameters that should be considered when designing parallel programs. Hope it can bring a little help to parallel programmers.

1. THE MODERN STATE OF THE PROBLEM UNDER STUDY

1.1. Literature reviews

On dblp, there are 46 papers about mpi hybrid programming (MPI+OpenMP).

In article Parallel optimization of three dimensional wedge shaped underwater acoustic propagation based on MPI+OpenMP hybrid programming model [1], a 43x speedup was achieved by using MPI+OpenMP hybrid programming. But not all parts can be accelerated. In some places it will be slower.

Structured mesh-oriented framework design and optimization for a coarse grained parallel CFD solver based on hybrid MPI/OpenMP programming [2], they propose a spinning and nonblocking locking method and a double-buffer mechanism based on the fine-grained mutex, which has better acceleration performance than the existing barrier. Whether in small-scale computing units or large-scale computing units.

Parallelizing MPI using Tasks for Hybrid Programming Models [3], they used the Intel Xeon Phi processor and embedded the MPI into the OpenMP program. The larger the computing unit (including the number of cores and ram size), the greater the speedup ratio.

On dblp, there is about 10 papers on logp.

LogP: Towards a Realistic Model of Parallel Computation [4], this article proposes a model called logp on the basis of PRAM, which contains 4 parameters, L represents the upper bound of the delay, o represents the overhead, g represents the processing limit, and P represents the number of processors.

Optimal Broadcast and Summation in the LogP Model [5], this paper proposes an optimal propagation and synchronization algorithm on the Logp model.

Efficient Multiple-Item Broadcase in the LogP Model [6], this paper proposes an efficient multi-object propagation algorithm and gives a partial proof.

Upper Time Bounds for Executing PRAM-Programs on the LogP-Machine [7], this paper presents a method for subclassing a PRAM program into an efficient logp program and gives its upper bound.

BSP vs LogP[8], this article argues that BSP is easier to program for programmers.

Fast Parallel Sorting Under LogP: Experience with the CM-5 [9], this article analyzes the performance of 4 parallel classification algorithms on CM-5 and compares it with the prediction of logp. They found that logp is more accurate in predicting communication, but insufficient in local performance analysis, which may be due to too few analysis parameters for local operations.

Parallel 'Go with the winners' algorithms in the LogP model [10], this article analyzes the performance of the AJdous and Vazirani algorithm under the logp model, achieving almost linear speedup.

On dblp, there are 9 papers on loggp.

In article LogGP: Incorporating Long Messages into the LogP Model - One Step Closer Towards a Realistic Model for Parallel Computation [11], a model called loggp is proposed, which adds a parameter g to logp, which represents the bandwidth when sending long messages.

LogGP Quantified: The Case for MPI [12], this paper analyzes the performance and comparison of loggp models on three supercomputing platforms. It is found that loggp is not very suitable for fitting the performance of mpi communication. But it can be used to analyze his performance bottleneck.

Predictive Analysis of a Wavefront Application using LogGP [13], this article uses the loggp model to analyze the performance bottleneck of the Sweep3D application.

Low-Overhead LogGP Parameter Assessment for Modern Interconnection Networks [14], this article presents a method to measure the logp or loggp parameter. Also gave its open-source implementation.

LogGP in theory and practice - An in-depth analysis of modern interconnection networks and benchmarking methods for collective operations [15], proposes a new method for measuring loggp parameters for collective operations.

Research on LogGP Based Parallel Computing Model for CPU/GPU Cluster [16], this paper proposes a model of heterogeneous computing, VLogGP, and measures the parameters of the model on the TH-1A platform.

2. FOUNDATIONS OF SUPERCOMPUTING

2.1. Introduction to supercomputers

Although, there is a relatively complete theory of parallel computing at present. But it seems that we still must decide the final design based on the benchmark.

Then the first question is if we can know the performance of the entire parallel system based on some existing parameters.

For example, when we buy a personal computer, we cannot estimate the performance of the computer only based on the CPU frequency and the number of cores.

But we can run a benchmark. But when designing supercomputing, running points are not always achievable.

In this paper, we will introduce the composition of supercomputers, performance bottlenecks, and create a mathematical model.

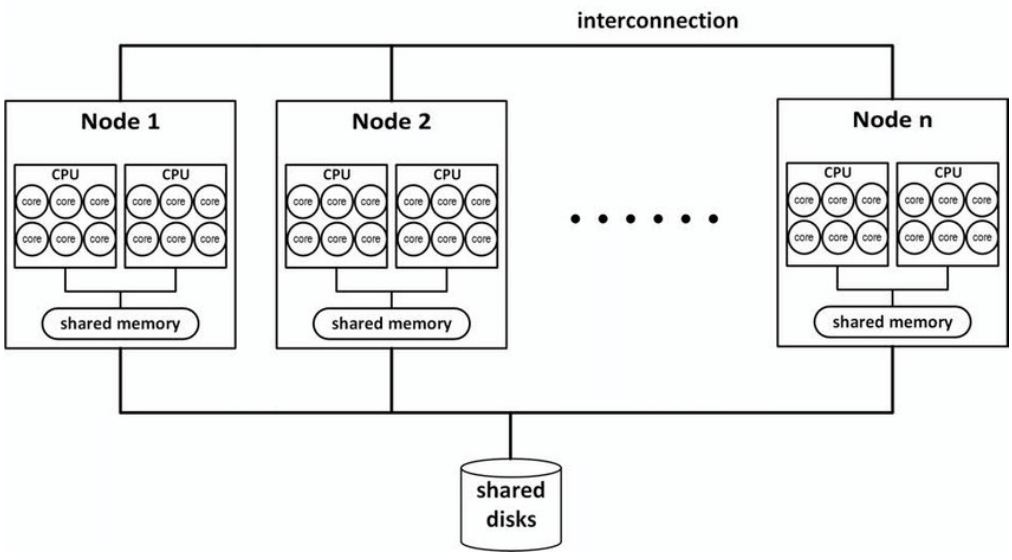

Fig. 1 - Architecture of a supercomputer [17]

A supercomputer system usually consists of a certain number of computing nodes. The nodes are connected by a high-speed network. Each node has several independent processors and independent memory. Each node shares a file server, a gateway for external communication. A small server is used as the management node.

One difference between a parallel computing system and a server or distributed computing system is that it does not have a good hardware redundancy mechanism. Because it focuses more on price/performance, and its data is relatively less important.

Supercomputers use the highest speed internal internetwork (e.g. InfiniBand). Warehouse Computer focuses on interactive applications, hyperscale storage, reliable and high-speed external Internet。

Fig. 2 – A photo of supercomputer Fugaku [18]

2.2. Rankings

2.2.1. TOP500

The most famous ranking of supercomputers in the world is TOP500.

Supercomputers are also known as the most important weapons of the country. Almost all countries with supercomputers, such as the United States, Japan, China, and some European countries. The application scope of supercomputers is mainly concentrated in several fields, scientific research, weather and climate prediction, research and development of new materials, and information services. These fields all require high costs, a lot of time, and even harsh conditions.

In the top500 list, most supercomputers are in these countries. Japan, the US and China have monopolized the best supercomputers since the 21st century. Several

supercomputers in Russia have also entered the top500 list, but none of them were designed and manufactured by Russia itself. Russia has the 8th largest number of supercomputers in the world. More than 90 percent of supercomputers are computer clusters, and the rest are almost MPPs. Only about 2 percent of supercomputers have coprocessors. More than 85% of them are configured with NVidia accelerators, about 3% for Intel, and less than 1% for AMD. About 52% of supercomputers use an Ethernet connection, and about 28% use InfiniBand. Almost all supercomputers use the Linux operating system. More than 45% of supercomputers have at least 20 processors per socket. Almost all supercomputers have at least 30,000 cores. All supercomputers have a computing power of 2.8E, but the United States and Europe are stepping up the development of E-class supercomputers. The computing speed of supercomputers has grown much faster than the development of processors (Moore's Law) [19]. Many programs running on supercomputers has millions of lines of code. For example, the computational chemistry software nwchem developed by the Pacific Northwest National Laboratory under the U.S. Department of Energy has about 6 million lines of code.

2.2.2. GREEN500

For supercomputers, other than performance is the most important thing. The energy consumption ratio is also very important. For example, the top500-ranked Supercomputer Fugaku consumes 29,899 kilowatts per hour.

The number one of Green500 is MN-3, which was developed by Preferred Networks in Japan.

It has 1664 cores, 110,592G of memory, uses a Xeon Platinum 8260M 24C 2.4GHz, and the connection between nodes is MN-Core DirectConnect.

2.2.3. Famous benchmarks

Linpack is tested by solving linear equations.

Linpack tests the system's ability to perform floating-point calculations. It tests how fast a computer can solve a dense system of n*n linear equations Ax=b.

HPCG testing as a complement to Linpack. It tests the solution of differential equations, mainly including the following operations,

Sparse matrix-vector multiplication.

Vector update.

Global dot product.

Locally symmetric Gauss-Seidel is smoother.

Sparse trigonometric solving (as part of the Gauss-Seidel smoother).

Powered by a multigrid preconditioned conjugate gradient algorithm that executes critical kernels on a set of nested coarse grids.

Graph500 is a test set that focuses on data intensive. Graphs are a core part of many workloads. Its benchmark problem is searching for elements and finding shortest paths.

Computational Physics Biological Applications

GTC-P (Gyrokinetic Toroidal Code) simulates the movement of ions through a tokamak by solving the Vlassov-Poisson equation using a particle element algorithm

Meraculous is a massively parallel genome assembly benchmark that constructs and traverses a de Bruijn graph of all overlapping substrings of length k (k-mers) present in a redundant short sequence input dataset. By traversing the de Bruijn graph, and discovering all (possibly disconnected) linear subgraphs, Meraculous is able to construct high-quality contiguous sequences of genomic data.

MiniDFT is a simulation application of plane-wave density functional theory (Density Functional Theory) for modeling materials. MiniDFT computes self-consistent solutions to Kohn-Sham equations using LDA or PBE exchange-correlation functions. For each iteration of the self-consistent field cycle, the Fock matrix is constructed and then diagonalized. To construct the Fock matrix, a fast Fourier transform is used to

convert the plane-wave basis (where the kinetic energy is the easiest to calculate) into real space (where the potential is evaluated) and the orbit back [20].

2.2.4. TOP1 supercomputer FUGAKU

In the top500 supercomputing ranking, the first is supercomputer FUGAKU. It was developed by the RIKEN Center for Computational Science. It has 7,630,848 cores and 5,087,232 GB of memory. The processor it uses is A64FX 48C 2.2GHz. It uses an original interconnection method between nodes, Tofu interconnect D.

Its Linpack performance is 442,010 TFlop/s and the theoretical peak is 537,212 TFlop/s.

It uses a single CPU to form a node. There are a total of 158,976 nodes. Each CPU is mounted on a CMU board. Eight CMUs form a tool holder. Three tool holders form a frame. 8 racks form a cabinet. Fugaku consists of 432 cabinets.

2.2.5. CPU of FUGAKU – FUJITSU A64FX

Fujitsu A64FX is based on the ARM instruction set. It is based on the instruction set architecture Armv8.2-A SVE 512 bits.

It has 48 cores, plus 2 auxiliary cores. Every 12 cores form a core memory group CMG, so there are 4 core memory groups in total.

The CPU performance is 3.3792TF for double precision, 6.7584TF for single precision, and 13.5168 TF for half precision. It shares 64KB of 4-way L1 cache for each core, and each CMG shares 8M of 16-way L2 cache.

The RAM it uses is HBM2 32G.

The interconnection method between nodes it uses is an original ToFu interconnection D, which will be described in detail below.

This CPU is based on TSMC's 7nm technology [21].

2.2.6. Tofu interconnection

The four CMUs are interconnected through NOC (network on chip). The NOC is connected to a PCIe controller and 6 TNIs. The TNI is connected to a Tofu Network Router for conversion, and the output is 10 dual ports [22].

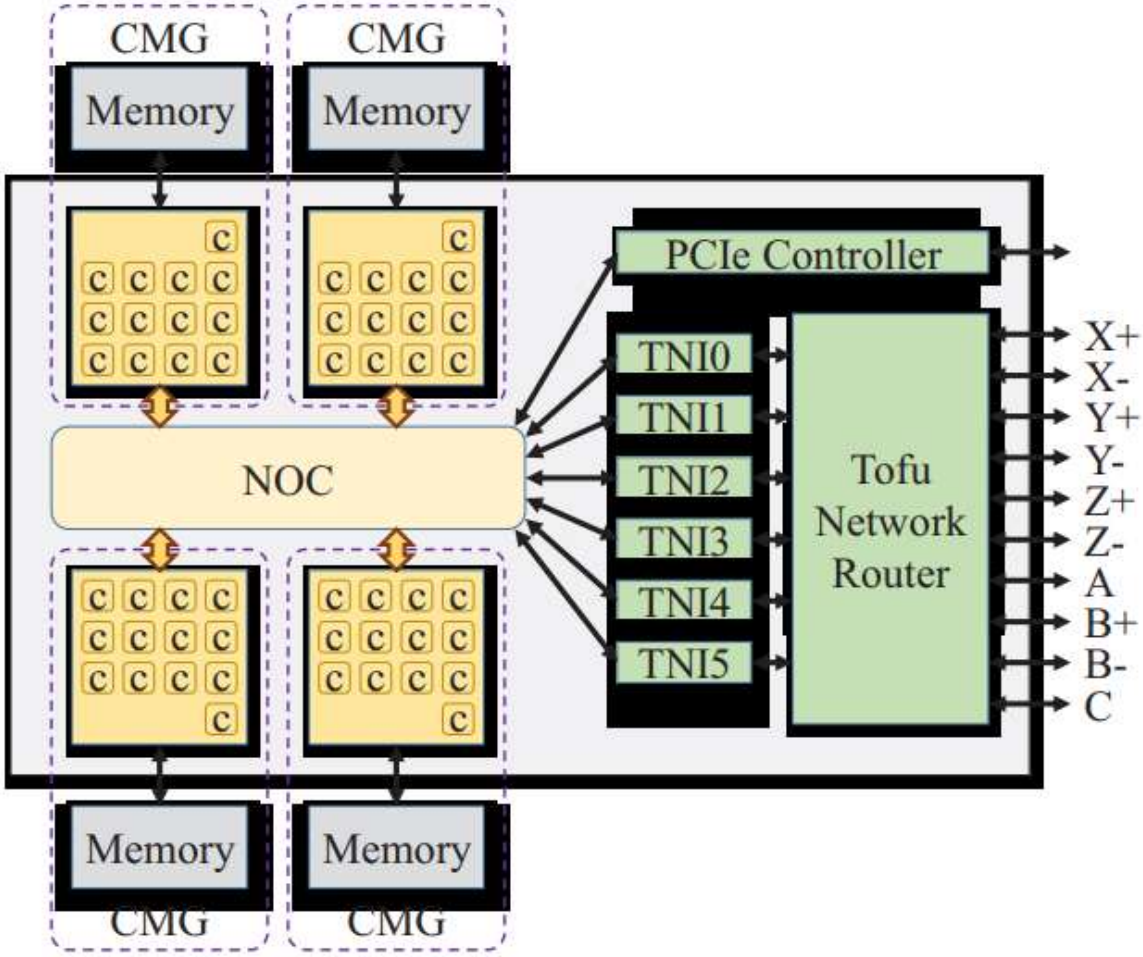

Fig. 3 – TOFU interconnection

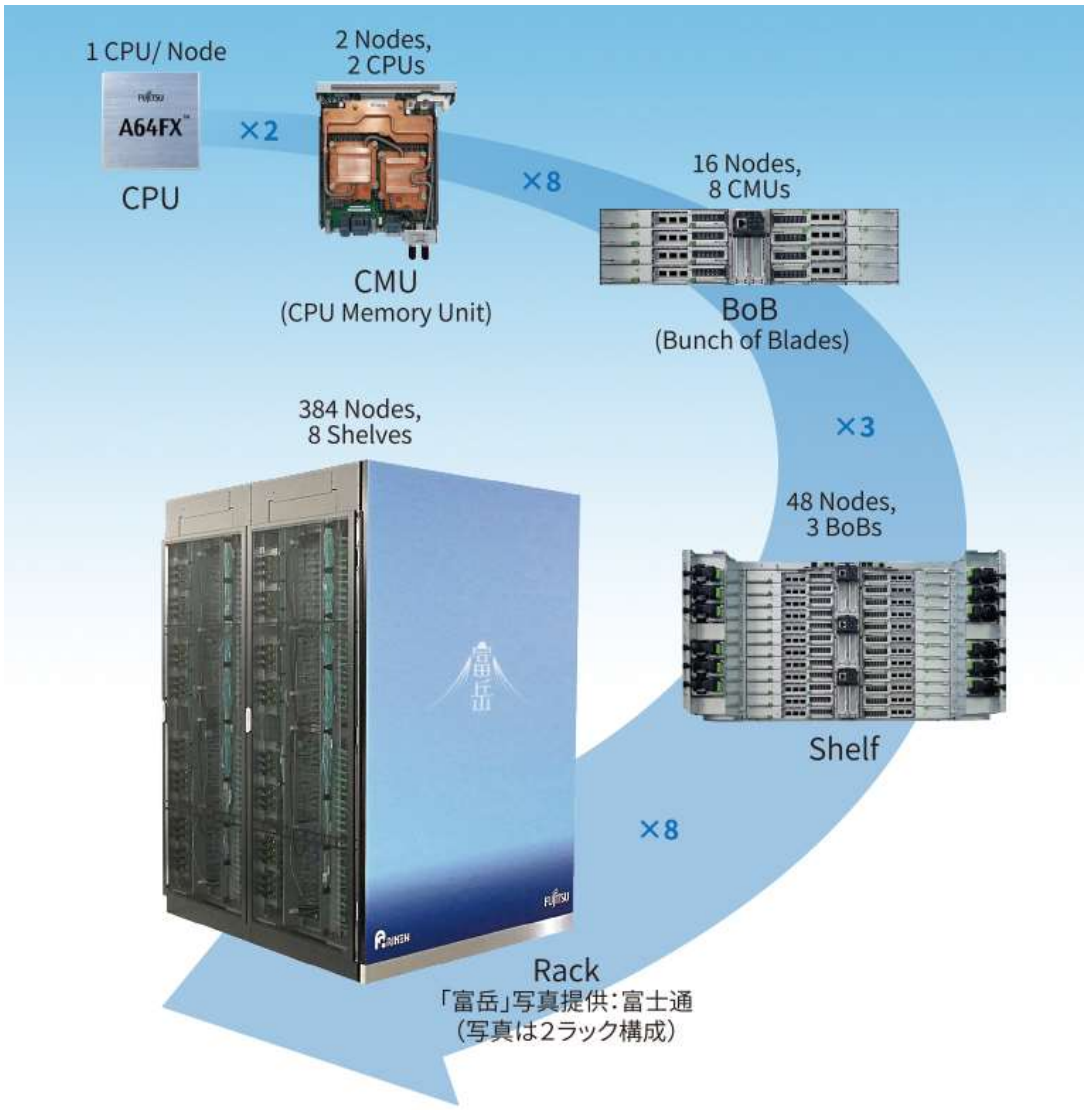

Fig. 4 – the combinations of supercomputer Fugaku

3. THEORY OF PARALLEL COMPUTING

3.1. Date parallel model

3.1.1. Instruction-level parallelism

Instruction-Level Parallelism refers to the fact that multiple instructions can be executed simultaneously. It can be implemented in software or hardware. The hardware method has superscalar, such as the following figure.

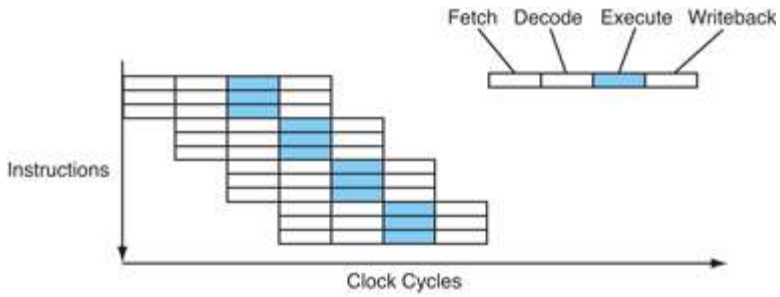

Fig. 5 – Instruction-level parallelism [23]

Data-Level Parallelism refers to the fact that the processor can process multiple pieces of data at the same time, which belongs to the SIMD model.

Disadvantages of DLP:

1. Increasing the pipeline clock frequency may result in an increase in CPI

2. Difficulty prefetching and decoding multiple instructions per clock cycle

3. Large-scale scientific computing and media stream processing have poor locality and low Cache hit rate.

But the SIMD model has the following advantages:

1. SIMD can effectively mine DLP, such as matrix operations, image and sound and other multimedia data processing

2. SIMD is more energy efficient than MIMD, and only needs to fetch an instruction for the same operation on a set of data

3. (Important) SIMD allows programmers to think serially (serial algorithms applied to parallel architectures)

Therefore, DLP has developed rapidly in recent years, especially in the field of supercomputers.

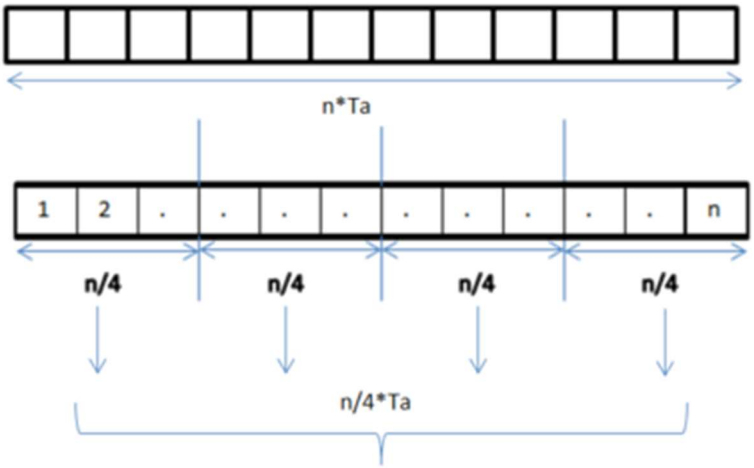

Fig. 6 – Data parallelism [24]

3.1.2. Thread-level parallelism

Thread-level parallelism means that multiprocessors support multiple threads to execute in parallel at the same time. Multiprocessor architectures are roughly divided into two types:

Symmetric Shared Memory Multiprocessor (SMP): Also known as centralized shared memory architecture, the number of cores is small and shares a centralized memory that all processors can access equally (also known as UMA, Uniform Memory Access).

Distributed Shared Storage (DSM): Multiprocessing uses physical distributed memory, and multiple cores/distributed memories are connected through a high-speed interconnection network; DSM access times to different memories are inconsistent, and it is clear that the core access speed to the internal memory of its own node The memory access speed is higher than that of other nodes, and the access speed of accessing other nodes' memory is also related to the network topology between nodes (also known as NUMA, non-uniform memory access).

Both modes need to focus on consistency issues between storages [25].

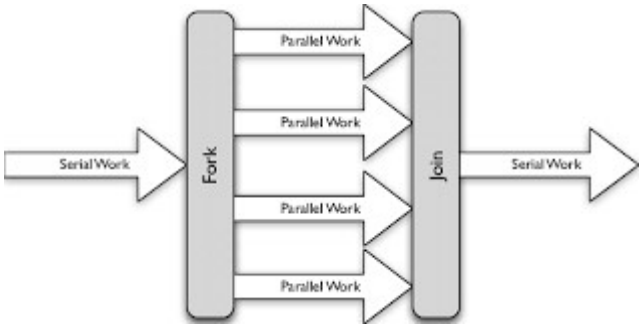

Fig. 7 – Thread parallelism [26]

3.1.3. Single Instruction Multiple Data

Single Instruction Multiple Data (SIMD) is a method that uses a controller to control multiple processors and simultaneously perform the same operation on each of a set of data (also known as a "data vector") to achieve spatial parallelism Technology.

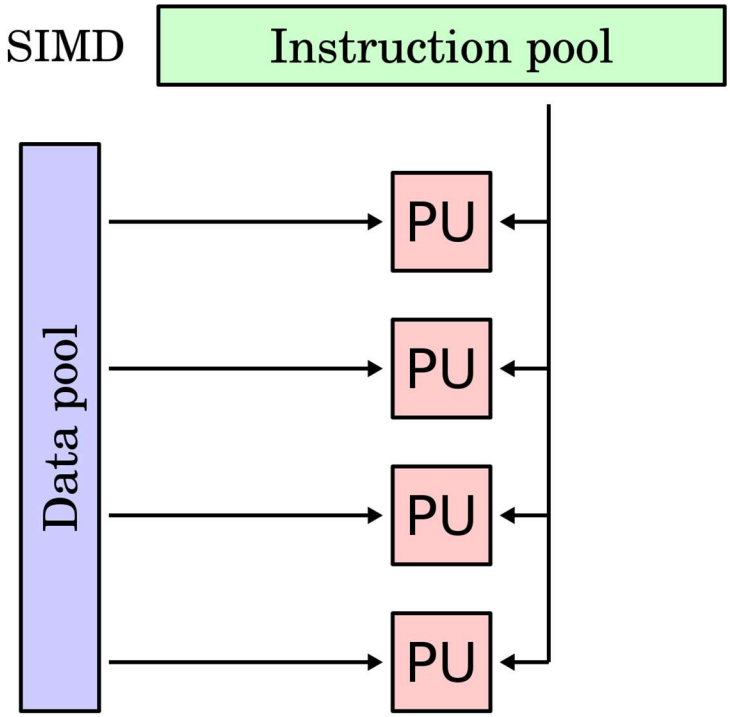

Fig. 8 – SIMD [27]

3.1.4. Multiple Instruction Multiple Data

Multiple Instruction Stream Multiple Data Stream (MIMD) is a technology that uses multiple controllers to asynchronously control multiple processors to achieve spatial parallelism.

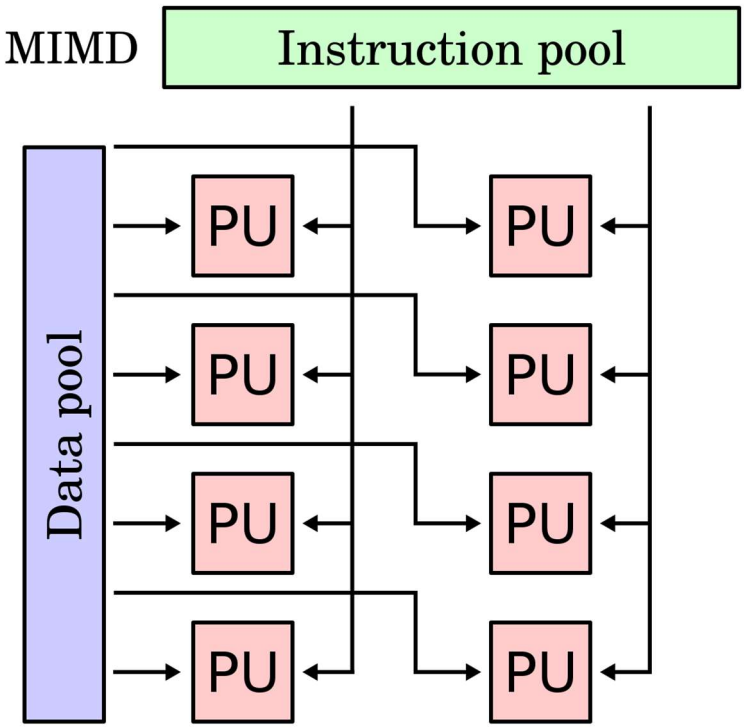

Fig. 9 – A figure of MIMD [28]

3.1.5. Morre’s law

Morre's law refers to an expected 18-month doubling of the chip's performance.

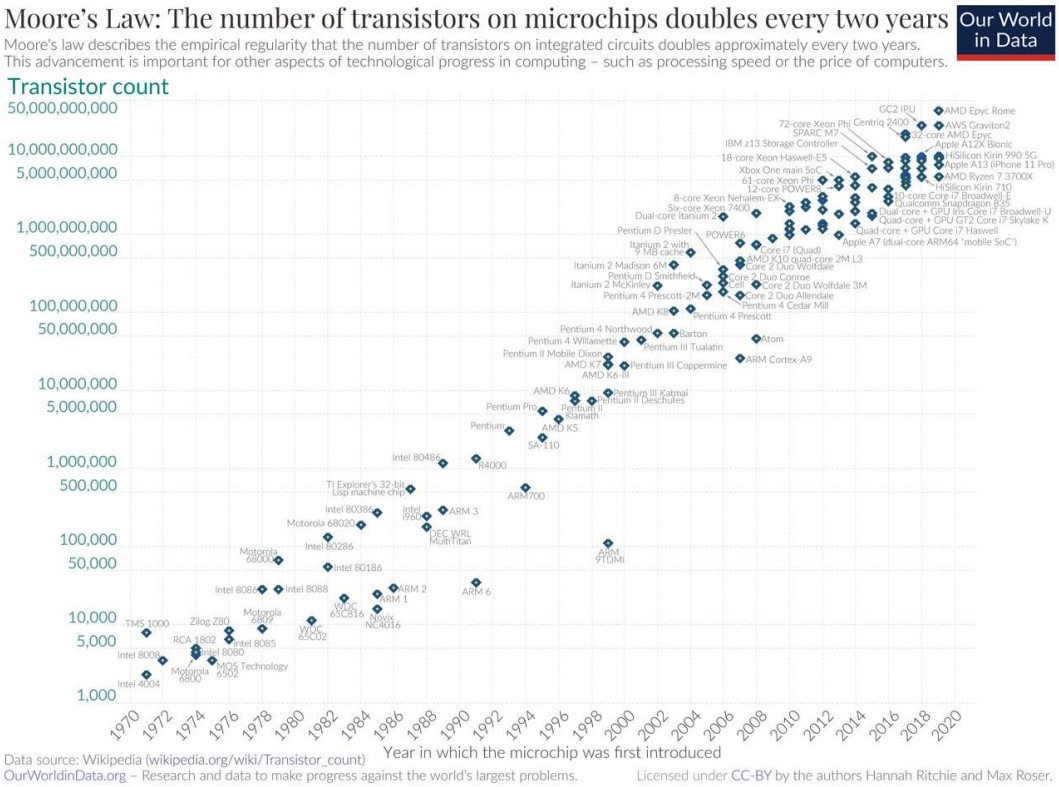

Fig. 10 – Morre's Law [29]

3.1.6. Dennard scaling

Dennard scaling is roughly saying, roughly, that as transistors get smaller, their power density stays the same, so power usage is proportional to area; voltage and

current ratios decrease with length.

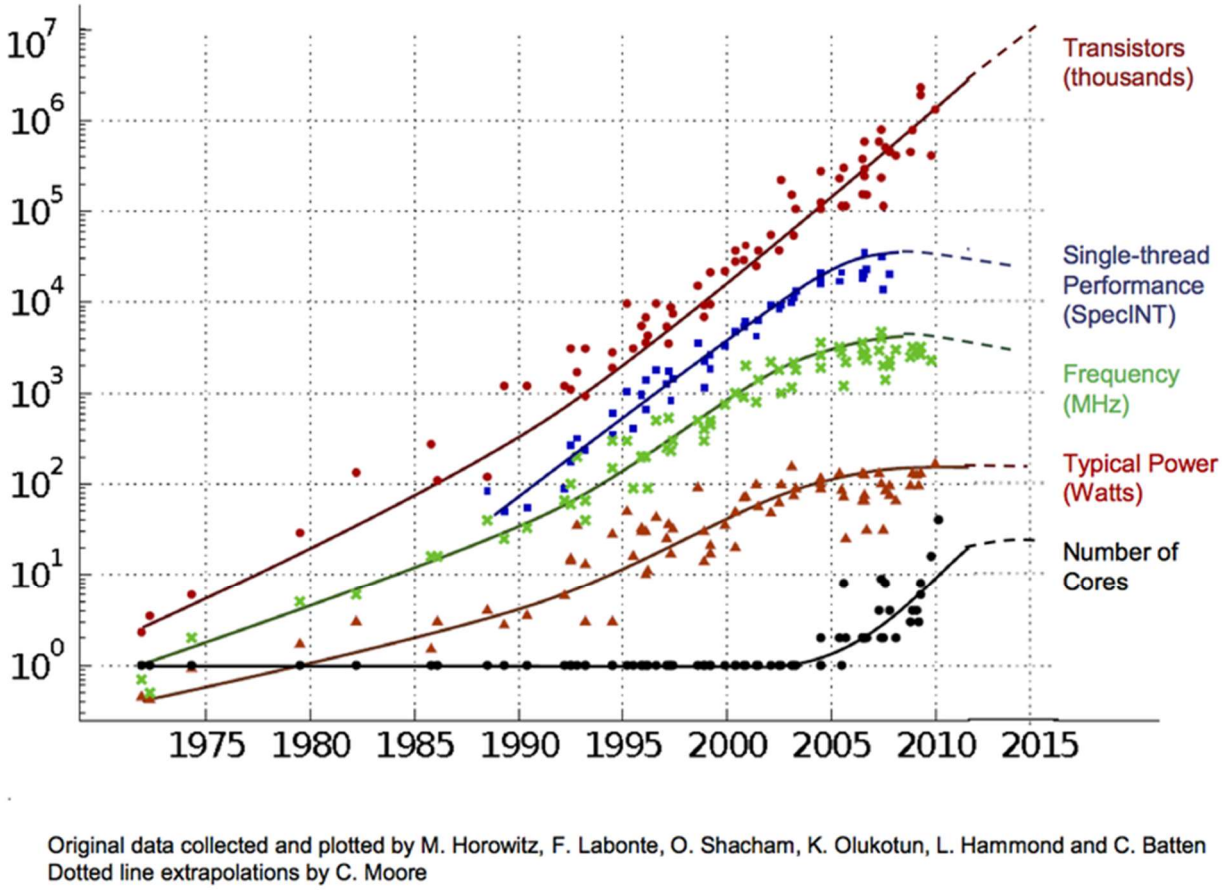

Fig. 11 – Dennard Scaling [30]

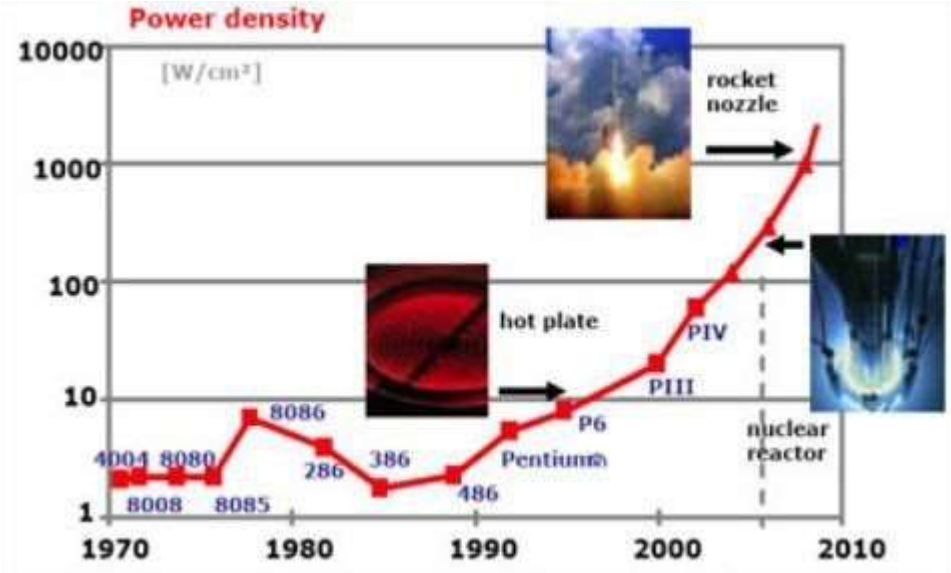

Fig. 12 – Power density by Dennard scaling

3.1.7. The real Morre's law – development of RAM

Compared with the Moore's Law-style development of CPU, the continuous increase of hard disk speed, and the emergence of solid-state drives, the speed of RAM is dwarfed by growth.

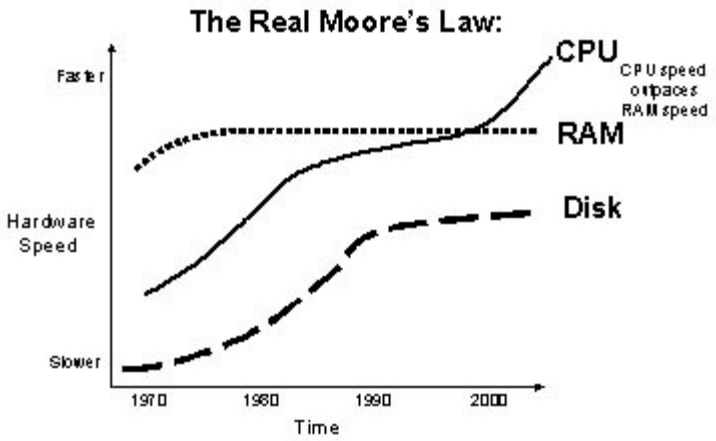

Fig. 13 – Real Moore's law [31]

As mentioned earlier, due to many constraints, the speed of development of computing systems faces many challenges. However, we currently do not know the minimum upper bound for actual computers, because modern chips may be further developed with the help of new technologies or materials. But we have proved the upper bound for Turing machines. We can also generalize the results to parallel computers.

3.2. Parallel computing models

3.2.1. PRAM model

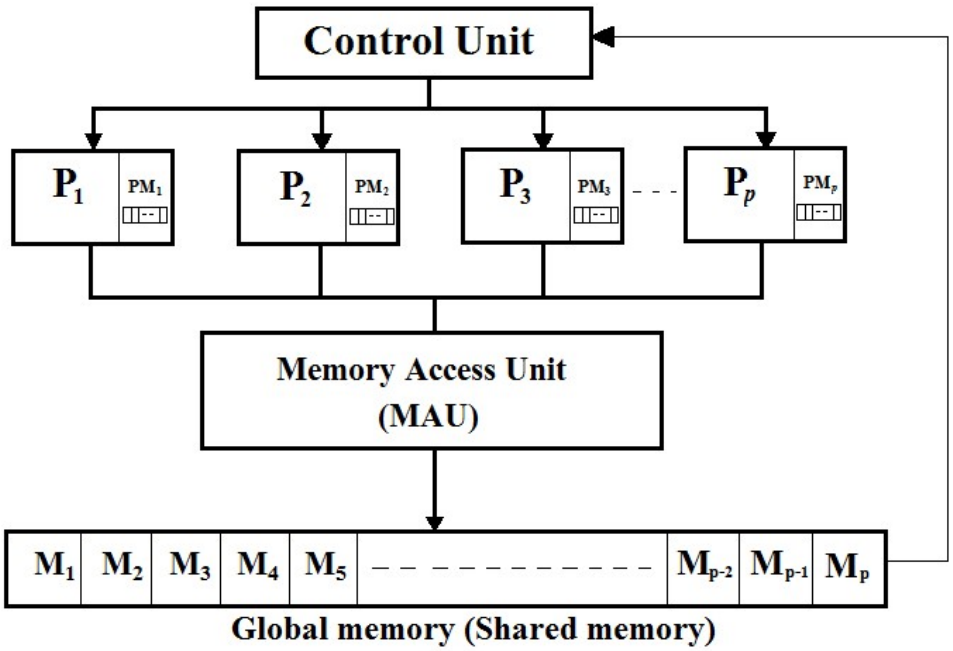

Fig. 14 – PRAM model [32]

A computational task can be said to be an efficient parallel algorithm if input of size n can be used by a parallel computer using n processors in logn running time.

A problem is in NC complexity if it can be decided in polylogarithmic time on a parallel computer with a polynomial number of processors.

A language is a set of decision problems in polynomial space if it can be solved by a parallel computer in polynomial space in polylogarithmic function time O(logn).

So, we can say that an algorithm is efficient parallel algorithm if it is in NC.

Since NC belongs to P, the computing power of a parallel computer is equivalent to that of a Turing machine. Therefore, problems that cannot be solved on Turing machines can also be solved by parallel computers. Although we do not know whether NC is equal to P, this is currently an open question.

And if P = NC, then P-complete is equal to NC.

3.2.2. Speedup ratio – Amdahl's law

Amdahl's law states that,

$$S = \frac{1}{(1-p) + \frac{p}{s}}$$

in which, S is the total speedup,

s is the speedup ratio of the part that can be accelerated,

p is the total running time ratio of the part that can be accelerated.

It is not difficult for us to find, As s approaches infinity, S = $\frac{1}{1-p}$.

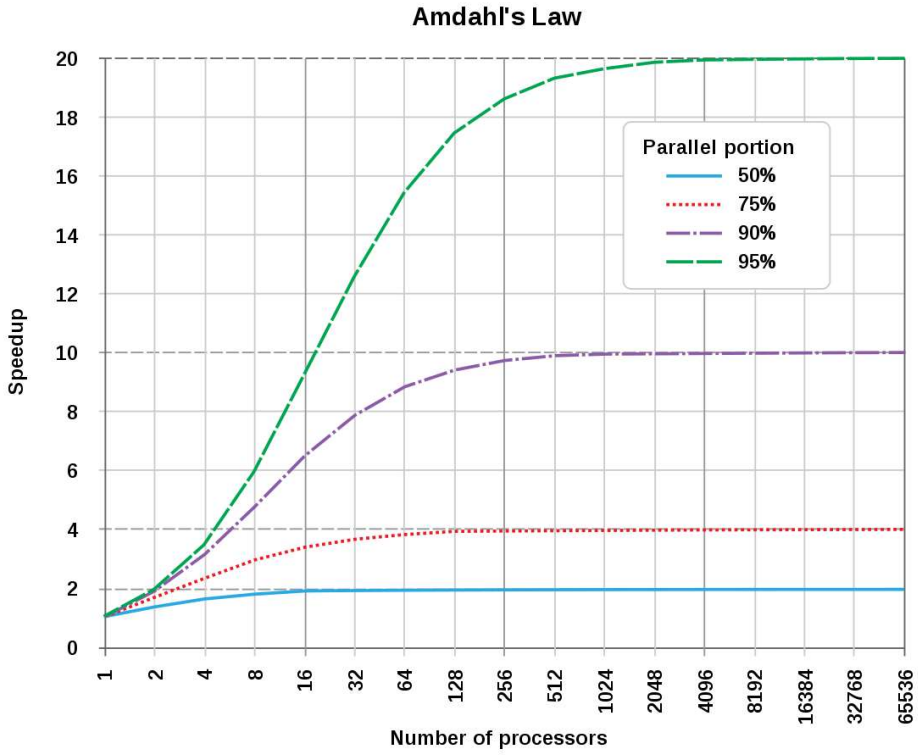

Fig. 15 – Amdahl's law [33]

3.2.3. Gustafson's law and differences

The Gustafson acceleration ratio S is defined as

$S=N+(1-N)*s$

N is the number of processors and s is the ratio of time spent in the serial part.

If Amdahl's law tells us an upper bound for parallel acceleration, no matter how many processors we have, the speedup cannot exceed the upper bound. Then Gustafson's law tells us that as long as we increase the number of processors, the

speedup can still increase, although it still cannot exceed the upper bound.

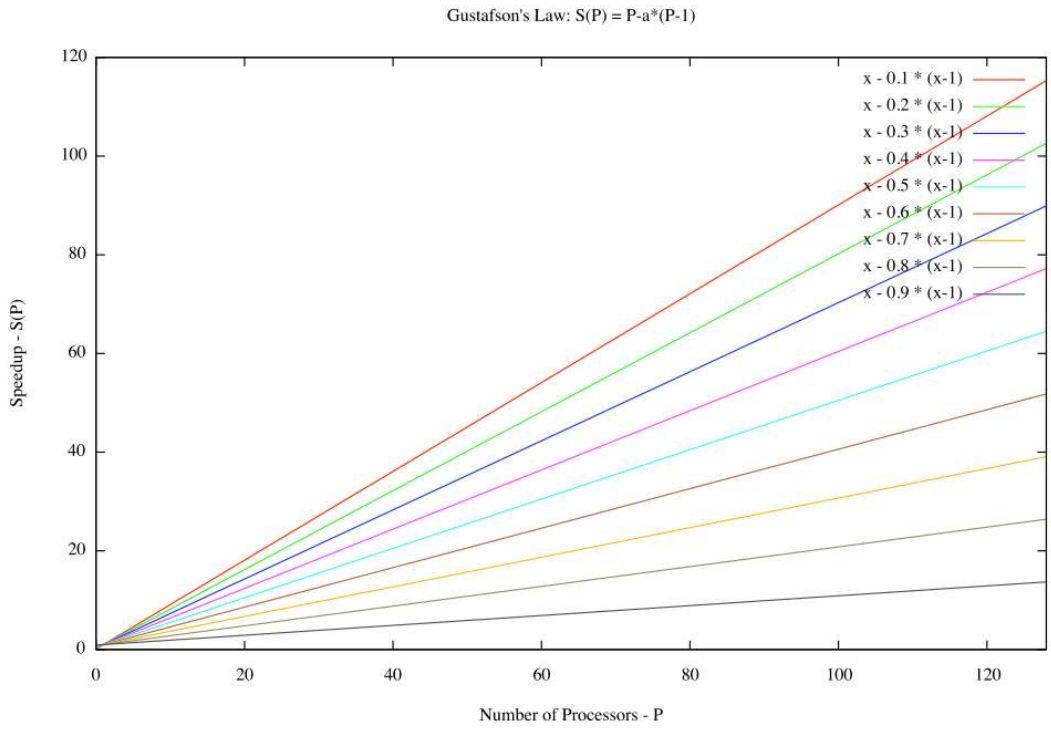

Fig. 16 – Gustafson's law [34]

3.2.4. LogP model

David Culler, etc. proposed a model called logp on the basis of PRAM, which contains 4 parameters, L represents the upper bound of the delay, o represents the overhead, g represents the processing limit, and P represents the number of processors.

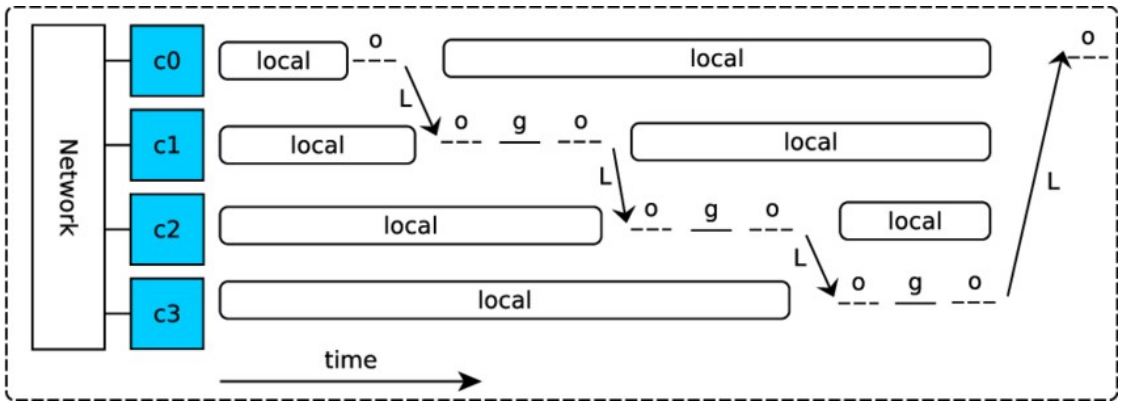

Fig. 17 – LogP model [35]

3.2.5. Variants of Logp model

In article LoGPG: Modeling network contention in message-passing programs, on the basis of logp and loggp, a parameter C is added, and a new model LoGPC is proposed. C is used to describe the network contention delay.

The LogP and MLogP models for parallel image processing with multi-core microprocessor, this article proposes an MLogP model, where M represents the number of threads to execute.

A LogP Extension for Modeling Tree Aggregation Networks, this article proposes a model called TAN based on LogP, adding two parameters, h and k. h represents the height of the propagation tree, the distance from the farthest node to the root node. K represents the number of children of the parent node.

LogGPH: A Parallel Computational Model with Hierarchical Communication Awareness, this article adds a parameter H based on LogGP, where h represents the hierarchical degree of communications of the system

3.2.6. LogPCK – a new model

If we are going to build a model for the fine-grainedness in MPI hybrid programming, I think two other parameters should be included.

C stands for contention overhead.

K means lock overhead.

So our parallel model is LogPCK.

3.3 Communication and broadcast

Broadcast is a collective communication operation in parallel programming. Used to send instructions or data to nodes in the cluster.

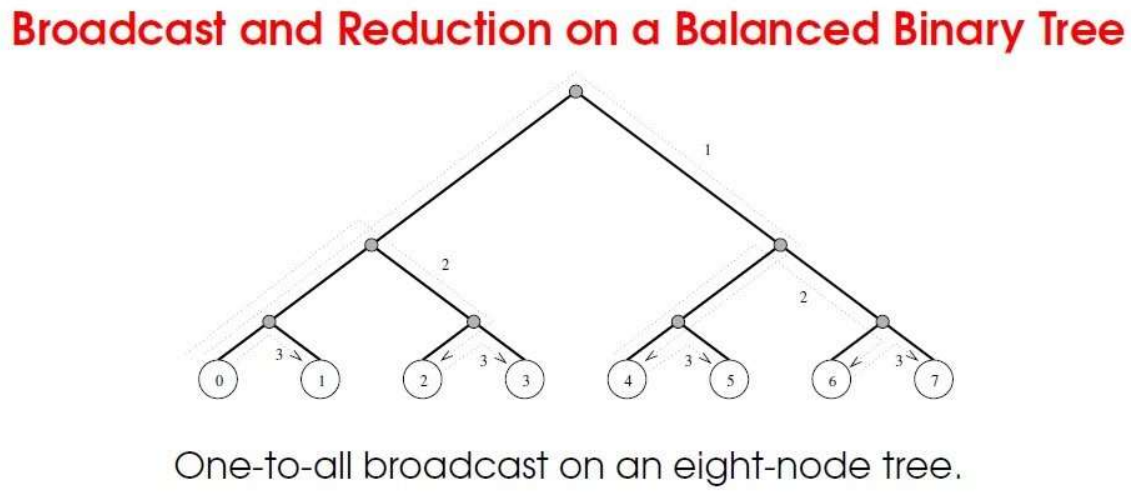

Fig. 18 – A broadcast tree [36]

The broadcast interface in MPI is MPI_Bcast.

A message of length n is distributed from one node to all other p-1 nodes.

Tb is the time it takes to send a byte.

Ts is the time it takes for a message to reach a node, regardless of length.

So the time for a message to get from one node to another is,

t = size*Tb + Ts.

p is the total number of nodes or processors.

3.3.1 Binomial Tree Broadcast

With binary tree broadcasting, every node that has already received the message continues to send it. This increases exponentially as the number of sending nodes doubles per time step. This algorithm is very suitable for short messages,

Sending a message to all nodes takes $\log_2(p)t$ time resulting in runtime of $\log_2(p)(mT_{byte} + T_{start})$.

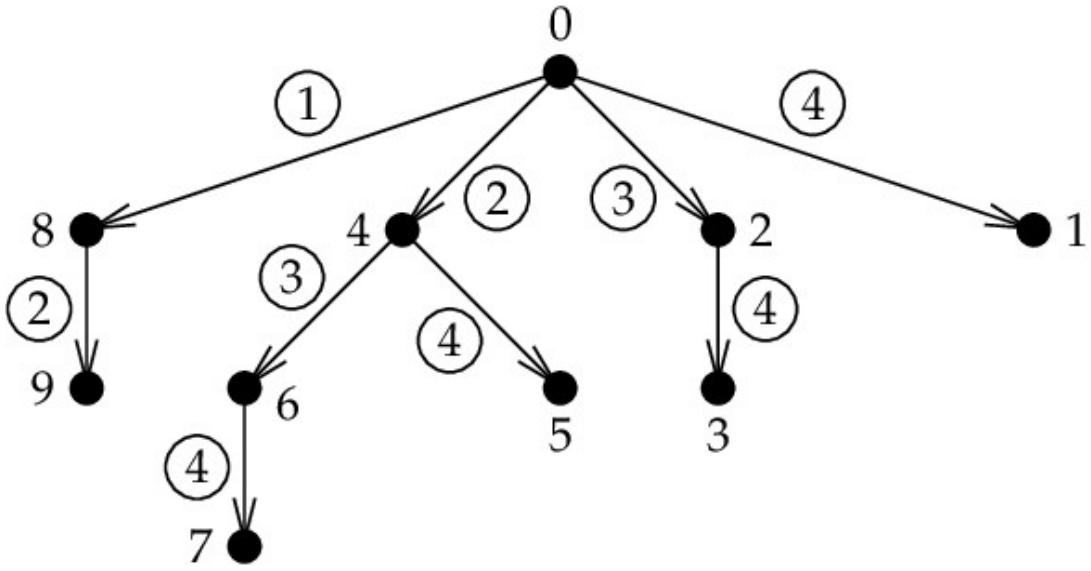

Fig. 19 - Binomial Tree Broadcast

3.3.2 Pipelined binary tree broadcast – Fibonacci tree

The algorithm combines binary tree broadcasting and linear pipeline broadcasting, making the algorithm suitable for both short and long messages. The goal is to get as many nodes working as possible while maintaining the ability to send short messages quickly. A good approach is to use a Fibonacci tree to split the tree, which is a good choice since messages cannot be sent to two children at the same time. This results in a binary tree structure [37].

We will assume that the communication is full duplex in the following. The depth of the Fibonacci tree structure is about $d \approx \log_{\Phi}(p)$, where $\Phi = \frac{1+\sqrt{5}}{2}$ is the golden ratio.

The result running time is $\left(\frac{m}{k}T_{byte} + T_{start}\right)(d + 2k - 2)$. The optimal tree is $k = \sqrt{\frac{n(d-2)T_{byte}}{3T_{start}}}$. So, this running is $2mT_{byte} + T_{start}\log_{\Phi}(p) + \sqrt{2mlog_{\Phi}(p)T_{start}T_{byte}}$.

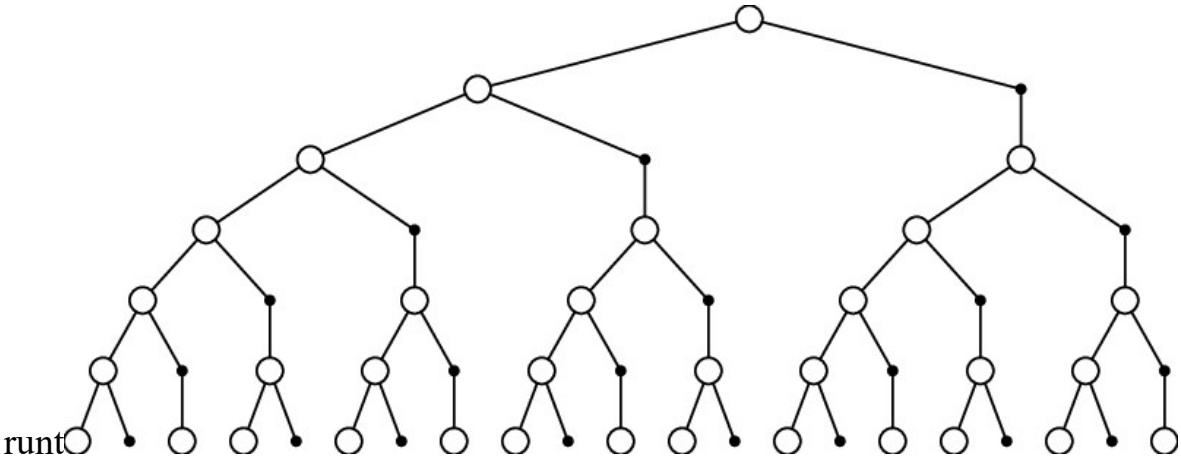

Fig. 20 – Fibonacci tree [38]

4. MPI HYBRID PROGRAMMING

4.1. Introduction to MPI, OpenMP, Pthread

The Message Passing Interface (MPI) is a standardized and portable message passing standard to work on parallel computing architectures.

OpenMP is a cross-platform multi-threaded implementation. The main thread (sequential execution of instructions) generates a series of sub-threads and divides tasks into these sub-threads for execution. These child threads run in parallel, and the runtime environment assigns the threads to different processors.

Pthreads defines a set of C language types, functions and constants, which are implemented with the pthread.h header file and a threading library.

There are roughly 100 function calls in the Pthreads API, all starting with "pthread_", and can be divided into four categories:

1. Thread management, such as creating threads, waiting for (join) threads, querying thread status, etc.

2. Mutex lock: create, destroy, lock, unlock, set properties, etc.

3. Condition Variable: operations such as creating, destroying, waiting, notifying, setting and querying properties

4. Synchronization management between threads using mutex

4.2. Hybrid MPI programming

First, MPI is based on a distributed memory system, while openmp and pthread are based on a shared memory system;

That is to say, the data sharing between mpi needs to be passed through message, because the programs synchronized by mpi belong to different processes, even different

processes on different hosts. On the contrary, since openmp and pthread share memory, there is no need to transfer data between different threads, just transfer pointers directly.

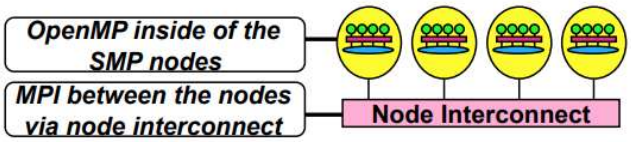

Fig. 21 – Illustration of MPI and OpenMP

4.2.1. Summary and Conclusion of MPI + OpenMP

– Usability on higher number of cores

Advantages

– Reducing memory requirements

– Improving performance (Figure)

– Exploiting additional layers of parallelism

-- Reducing communication overheads

-- Reducing load imbalance

-- Reducing memory access costs

Disadvantages

-- Development and maintenance costs

-- Portability

-- Libraries

Huge amount of pitfalls

Pitfalls of MPI & OpenMP & combination of them

– Idle threads

-- Synchronization

-- MPI derived datatypes

-- Memory effects and placement

Applications:(e.g., Computational Fluid Dynamics, Numerical Weather Prediction, Parallel machine learning [39][40])

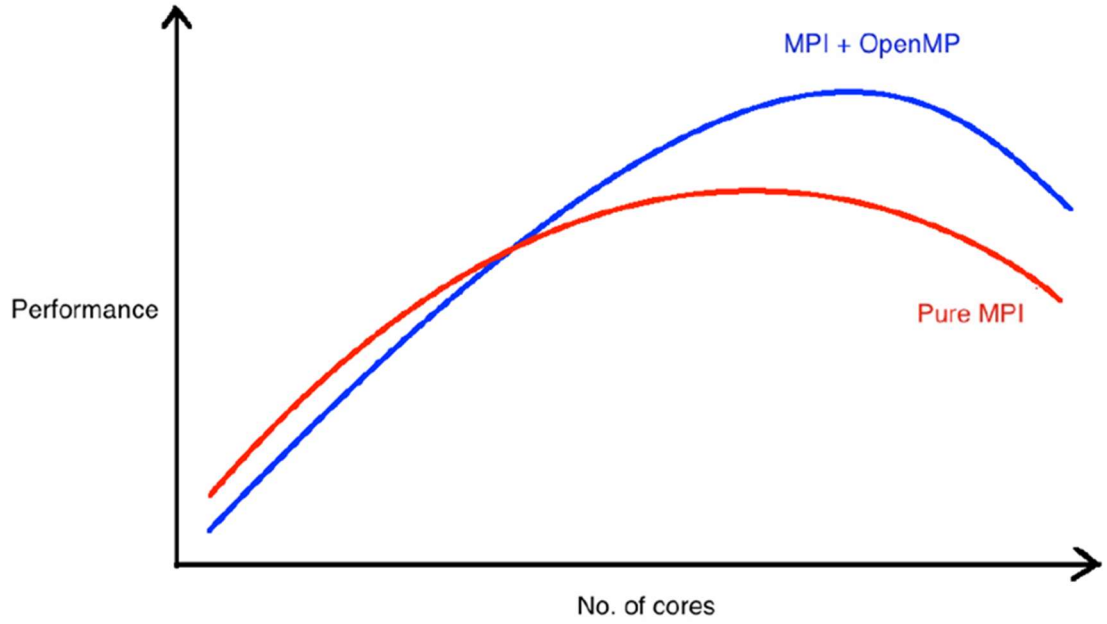

Fig. 22 – MPI performance vs MPI + OpenMP

MPI(Message Passing Interface) is one of the most popular software in High-Performance Computers. It describes parallelism between processes and it has several advantages, universality: expressivity, ease of debugging, performance. At the same time, Pthread(POSIX Threads), OpenMP are widelyused in thread parallelism which provides a shared-memory model within a process. However, Sometimes using them individually is not the most efficient way in computational models. Hybrid Programming combines these two things and grantees the thread-safety(threads have no races). This paper would design a benchmark to compare their performance and hope to give valuable results for later studies.

4.2.2 Comparisons of MPI and OpenMP

Most parallel applications running on high-performance computing(HPC) Systems usethe Message Passing Interface(MPI) for interprocess communication. However, MPI is notthe most efficient way to exploit resources, application developers usually combine multi-threaded programming with MPI. There are many multithreaded programming techniques,such as Pthread, OpenMP(Open Multi-Processing), and Intel Threading Building Blocks[43].The following table shows that the communication bandwidths compared pthread and mpi [44]. Since in nowadays computing devices, data movement is the bottleneck of whole computation [45], it's needed to be concerned.

Platform	MPI Bandwidth	Pthreads Bandwidth
Intel 2.6 GHz Xeon E5-2670	4.5	51.2
Intel 2.8 GHz Xeon 5660	5.6	32
AMD 2.3 GHz Opteron	1.8	5.3
AMD 2.4 GHz Opteron	1.2	5.3
IBM 1.9 GHz POWER5 p5-575	4.1	16
IBM 1.5 GHz POWER4	2.1	4
Intel 2.4 GHz Xeon	0.3	4.3
Intel 1.4 GHz Itanium 2	1.8	6.4

Tab. 1 - Bandwidths of MPI and Pthread

Pthread is light-weighted and has a much larger bandwidth, because it's based on shared memory structure.

So in this paper, we combine MPI with Pthread, which is called hybrid-threaded pro- gramming. We also use different MPI communication methods.

4.2.3. Related Works

In [42], The paper compared 4 granularities in MPICH2, an showed that bandwidth rankings are lock-free, per-object, per-global, global.The article [43] described the critical-section granularity in details, and improve the abstraction model, they proposed a first-come, first-served arbitration and a priority locking scheme, and gets a 5x accelerating result.

4.2.4 Thread Safety in MPI

The MPI-4 Standard specifies five levels of thread-safety that a user must explicitly select

single - No threads (MPI_THREAD_SINGLE)

funneled - Only the main thread calls MPI (MPI_THREAD_FUNNELED)

serialized - User serializes calls to MPI (MPI_THREAD_SERIALIZED)

multiple - Fully multi-threaded (MPI_THREAD_MULTIPLE)

runtime - Alias to "multiple"

The user must call the function MPI_Init_thread to indicate the level of thread support

desired. The user program must meet the restrictions of the level supported. A fully thread-compliant implementation will support MPI_THREAD_MU LTIPLE [42].

4.2.5. Critical Section Granularity

MPICH2 has 4 granularities, which are Global, Brief Global, Per object, Lock-free.

Global, which has a single, global critical section that is held for the procedure of mostMPI functions.

Brief Global, which has a single, global critical section, only held when required.

Per Object, which has separate critical sections for different objects and classes of ob-jects.

Lock Free is implemented by using atomic operations that exploit processor-specificfeatures [42].

However, Brief Global was not supported anymore from the latest version of MPICH2, and lock-free was not supported yet.

MPICH3 also has 4 granularities, which are Global, Per-vni, Per object, Lock free. Lockfree is not supported yet.And there are also 4 granularities in MPICH4, which are Global,Per Vci, Per object, Lock free. Lock free is not supported yet [46].

4.2.6. Performance Evaluation

We used a high-performance computer cluster which is provided by NSU. The following table shows the specification of this system.

This server cluster includes the following servers,

30 HP XL230a Gen9 servers, each containing:

Two 12-core Intel Xeon E5-2680v3 processors clocked at 2.5GHz.

192 GB RAM

48 HP BL2x220c G7 dual host servers, each containing two motherboards, each motherboard containing:

Two 6-core Intel Xeon X5670 processors at 2.9GHz

24 GB RAM

Communications network:

Infiniband 4x EDR, QDR, and DDR have bandwidths of 100, 40, and 20 Gbps, respectively, with latency around 1-7 μs. Designed to access network storage systems and the interaction of parallel processes running on different servers in a cluster (for example, to send MPI messages).

Transport network: Gigabit Ethernet for managing the server operating system and running processes.

Service Network: Fast/Gigabit Ethernet, providing access to management interfaces such as HPE integrated lights out or switching devices.

The other part only runs on a single node, and the programs with strong locality run on my personal computer. Its single-core performance is close to the HP XL230a Gen9 that PBS assigns to us. Its detailed configuration is as follows.

Component	Model
CPU	Core i7_6700HQ
Memory	8G DDR4
GPU	Nvidia GTX 970M
Disk	Phison SATA SSD 256GB

Tab. 2 – Configuration of my computer

To evaluate the performance, we use a model showed in Figure 25, The senders

have 1 process, n-1 threads, the receivers have n-1 processes. The Senders send messages to the receiver according to their labels. We set n to be 4, 8, 21, 51, 101, each iteration will repeat 6 times, and satirize their average used time. Inside of the sender, n-1 threads are created by pthread create functions.

MPICH4 has 4 granularities, which are global, per-vci, per-object, lock-free.

Since lock-free is not supported yet, and per-object is not supported by MPICH4, and when compiling by MPICH3 or MPICH2, it always occurred some errors. With a little shame, we only used two granularities to do our experiments.

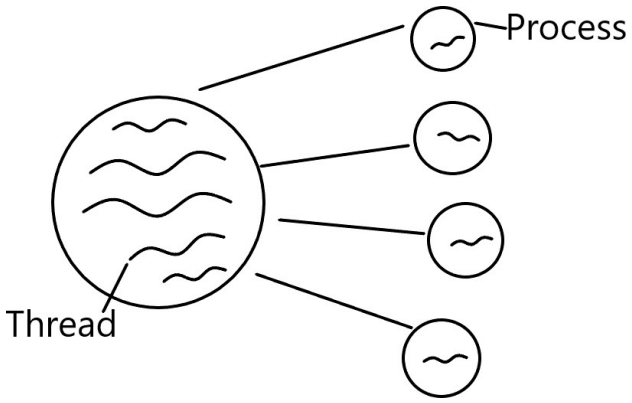

Fig. 23 - A model to test the performance

4.2.7 Performance with blocking send

Blocking communication is done by using MPI_Send() and MPI_Recv(). These functionsdo not return (i.e., they block) until the communication is finished. Simplifying somewhat, this means that the buffer passed to MPI_Send() can be reused, either because MPI savedit somewhere, or because it has been received by the destination. Similarly, MPI_Recv() returns when the receive buffer has been filled with valid data [7].

In this section, we use MPI_Send and MPI_Recv to send and receive the messages.As shown in Figure 26, per-vci takes more time than global.

4.2.8. Performance with non-blocking send

In contrast, non-blocking communication is done using MPI_Isend() and MPI_Irecv(). These functions return immediately (i.e., they do not block) even if the communication is not finished yet. We must call MPI_Wait() or MPI_Test() to see

whether the communicationhas finished [5].

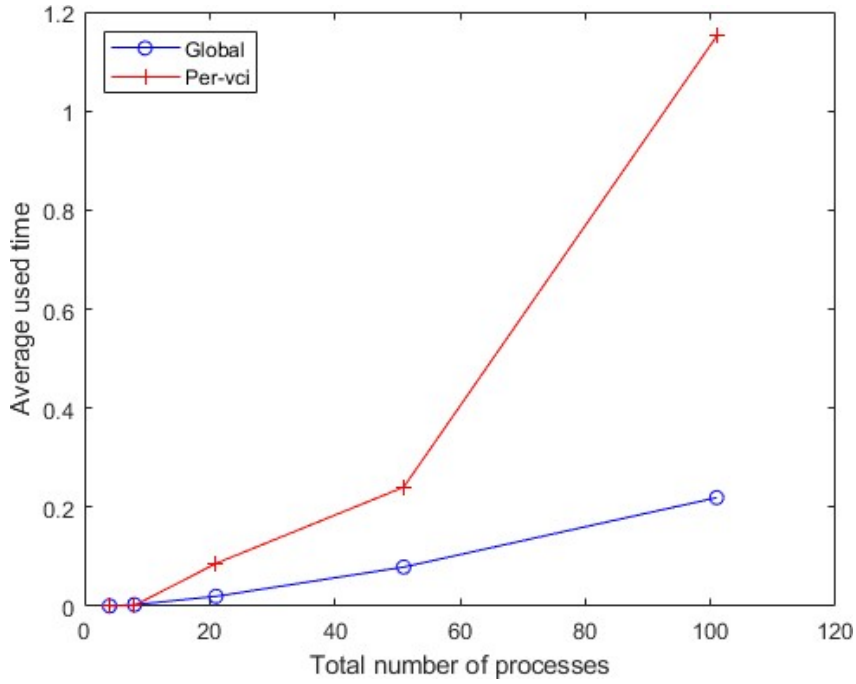

Fig. 24 - The average time used for a multithreaded process sending to n-1 processes withGlobal and Per Vci granularities

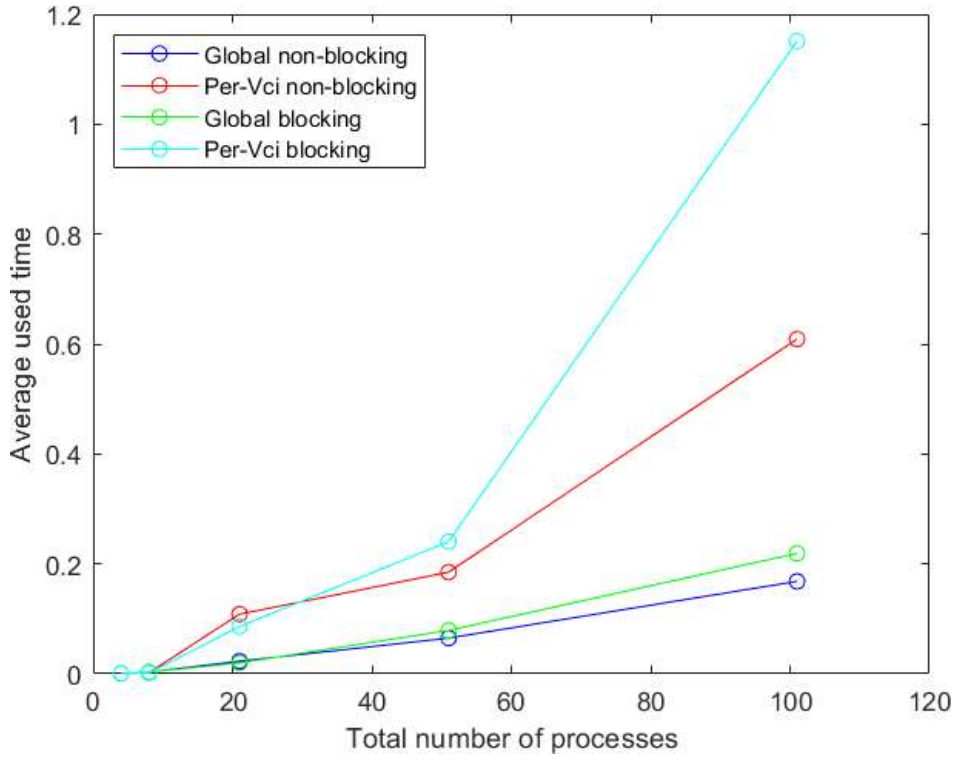

Fig. 25 - The average time used for a multithreaded process sending to n-1 processes withGlobal and Per Vci granularities

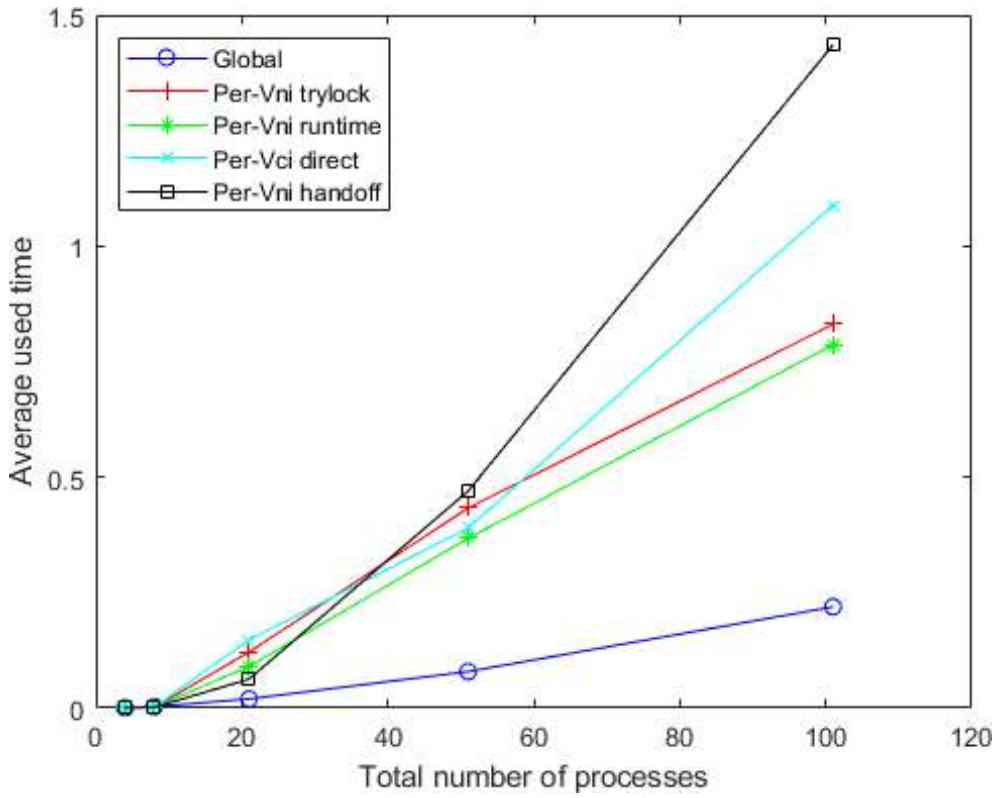

Fig. 26 - The average time used for a multi-threaded process sending to n-1 processes withGlobal and different Per Vci granularities

4.2.9. Performance with diffrent per-vci modes

In per-vci granularity, it has 4 different modes, which are direct, handoff, trylock, run- time. And we have to experiment to compare their performance. We just use the blocking communication model.

As you can see in the below figure 28, global is the fastest, runtime is the second, then trylock, then direct, at last handoff.

OpenMP Performance vs Pthread Performance

The application programming interface (API) OpenMP (Open Multi-Processing) supportsmulti-platform shared-memory multiprocessing programming in C, C++, and Fortran, on

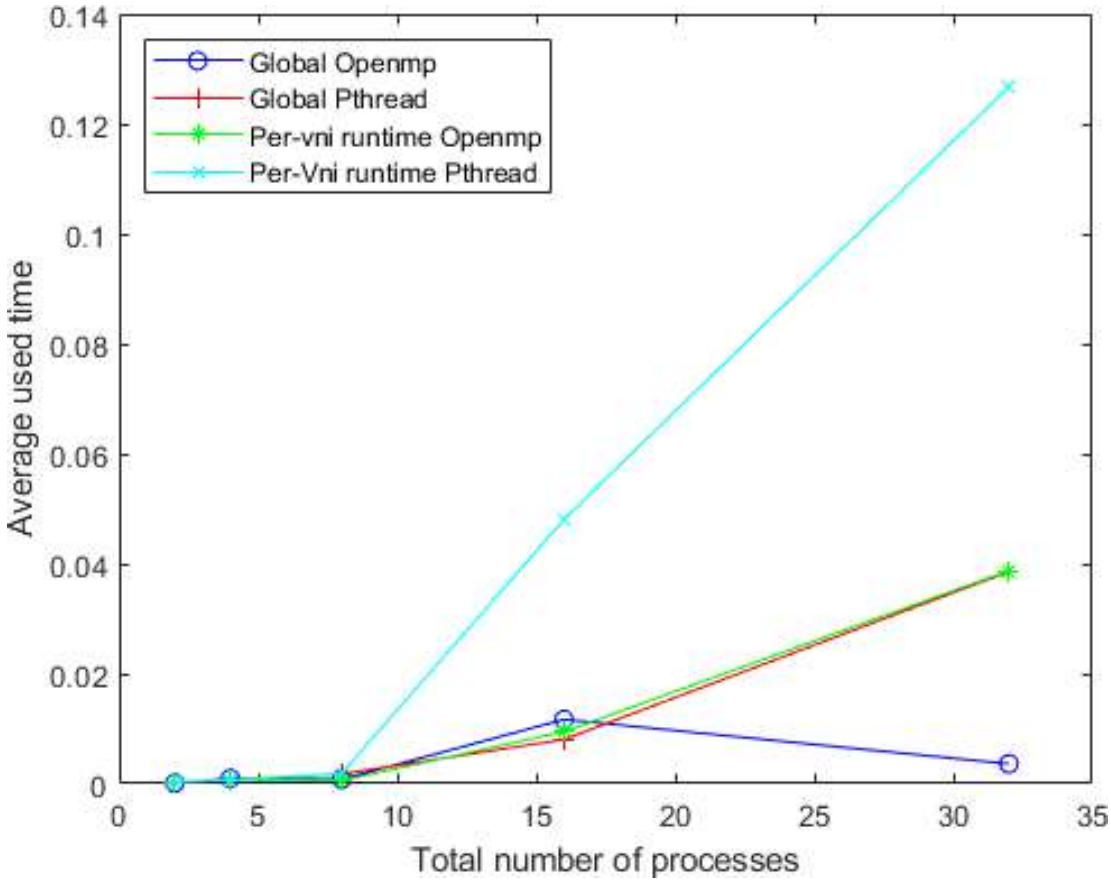

Fig. 27 - The average time used for a multi-threaded process sending to n-1 processes with OpenMP and Pthread

many platforms, instruction-set architectures, and operating systems, including Solaris, AIX, HP-UX, Linux, macOS, and Windows. It consists of a set of compiler directives, library routines, and environment variables that influence run-time behavior.

OpenMP uses a portable, scalable model that gives programmers a simple and flexible interface for developing parallel applications for platforms ranging from the standard desktopcomputer to the supercomputer [48] [49] [50].

As figure 5 shown, OpenMP although is an integrated function, Pthread is a light- weighted, efficient communicated library, but the performance of OpenMP overcomes Pthread.And please note that the Pthread version is written by an average level programmer, maybe an excellent programmer can write a Pthread with many optimizations.

Collective communication Performance vs point to point

Performance

MPI library provides us with a function bcast to broadcast the message to the whole world. It's just a combination of sends and receives. But we still test the performance of this function in different granularities. Here we need to take the same effect, all the send functionswill send the same content information to the receivers, but to different destinations with different tags.

As figure 30 shows that MPI_Bcast is worse than MPIand MPI_Recv with Pthread im- plementation, much worse than that with OpenMP.

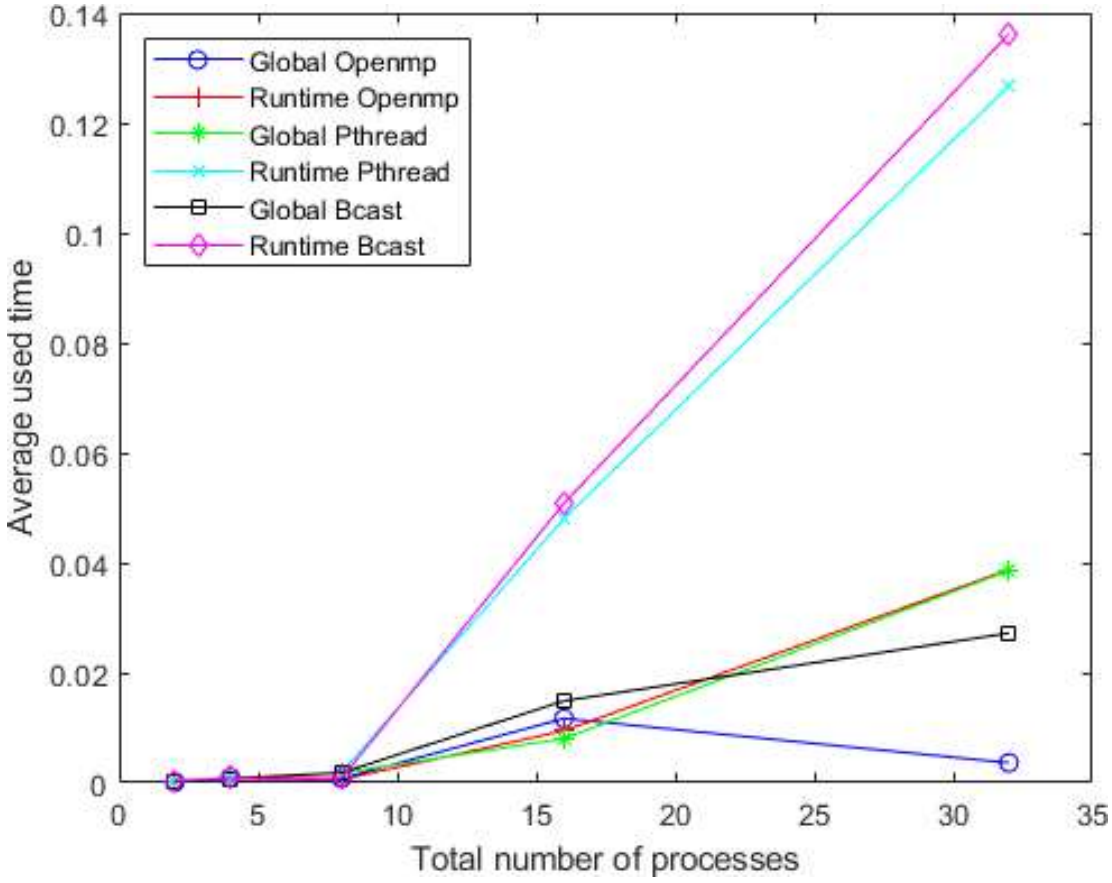

Fig. 38 - The average time used for a multi-threaded process sending to n-1 processes withbcast and multiple sends

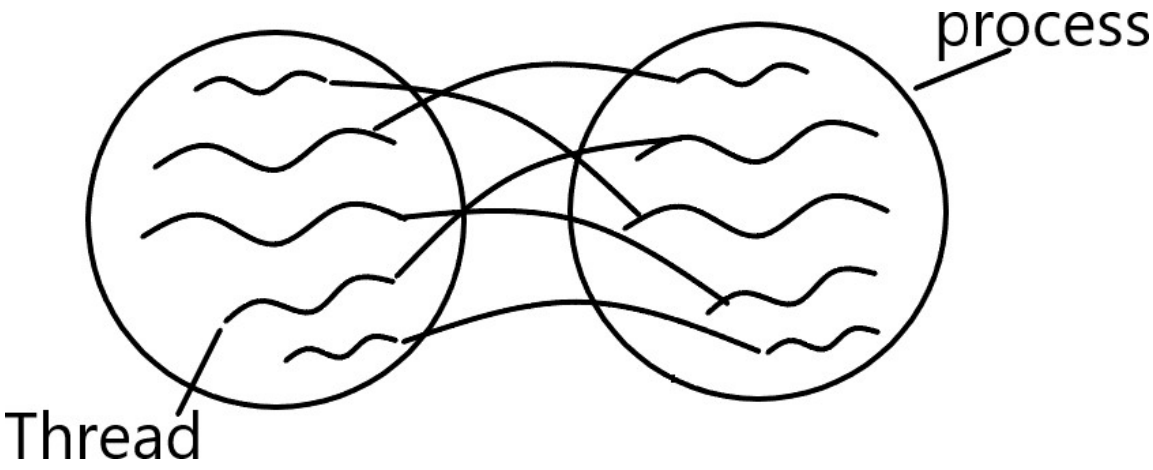

Fig. 39 - A model to test the performance of threads

4.2.10 Process performance vs Thread performance

MPI hybrid programming means we need to combine MPI with Pthreads or OpenMP, but why do we need to do this instead of just using more processes created by MPI world size. As we explain in the introduction part, Pthread is a light-weighted and efficient-communicated library, and has the following advantages [4]:

Threaded applications offer potential performance gains and practical advantages over non-threaded applications in several other ways:

Overlapping CPU work with I/O: For example, a program may have sections where it is performing a long I/O operation. While one thread is waiting for an I/O system call to complete, CPU intensive work can be performed by other threads.

Priority/real-time scheduling: more important tasks can be scheduled to supersede or interrupt lower priority tasks.

Asynchronous event handling: tasks which service events of indeterminate frequency and duration can be interleaved. For example, a web server can both transfer data from previous requests and manage the arrival of new requests.

The model of Pthread, figure 31 uses two processes, which each of them has the same number of threads as the number of receiver processes in figure 1.

The last experiment shows in Figure 32, which shows that Pthread is faster than the MPIprocess.

4.2.11 Mutex counting and time measurement

Mutual exclusion is a property of concurrency control, which is instituted for the purpose of preventing race conditions. It is the requirement that one thread of execution never enters a critical section while a concurrent thread of execution is already accessing critical section,

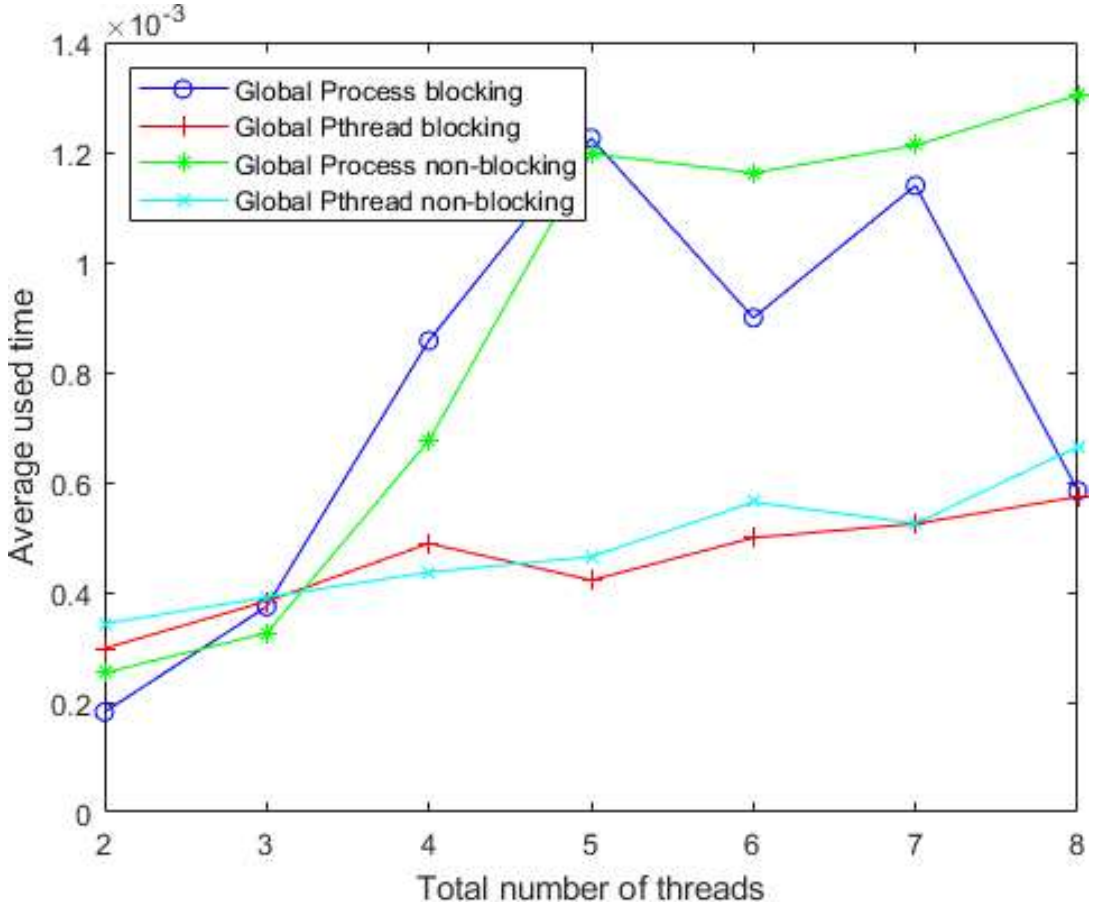

Fig. 30 - The average time used for a multithreaded process sending to n-1 threads/processeswith blocking or non-blocking sends/recvs

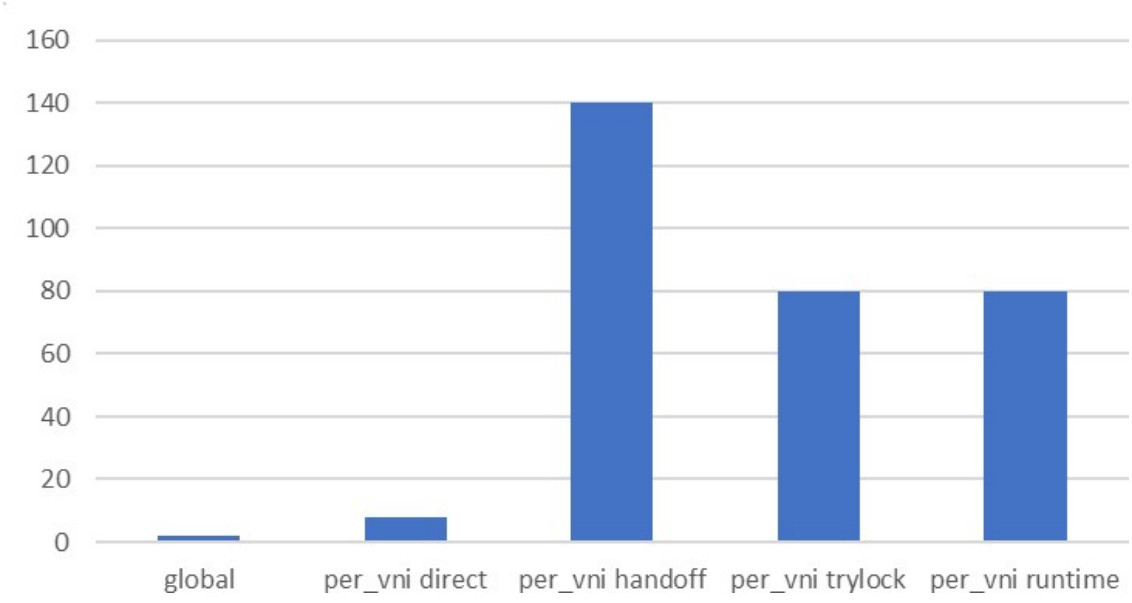

Fig. 31 – Mutex counting

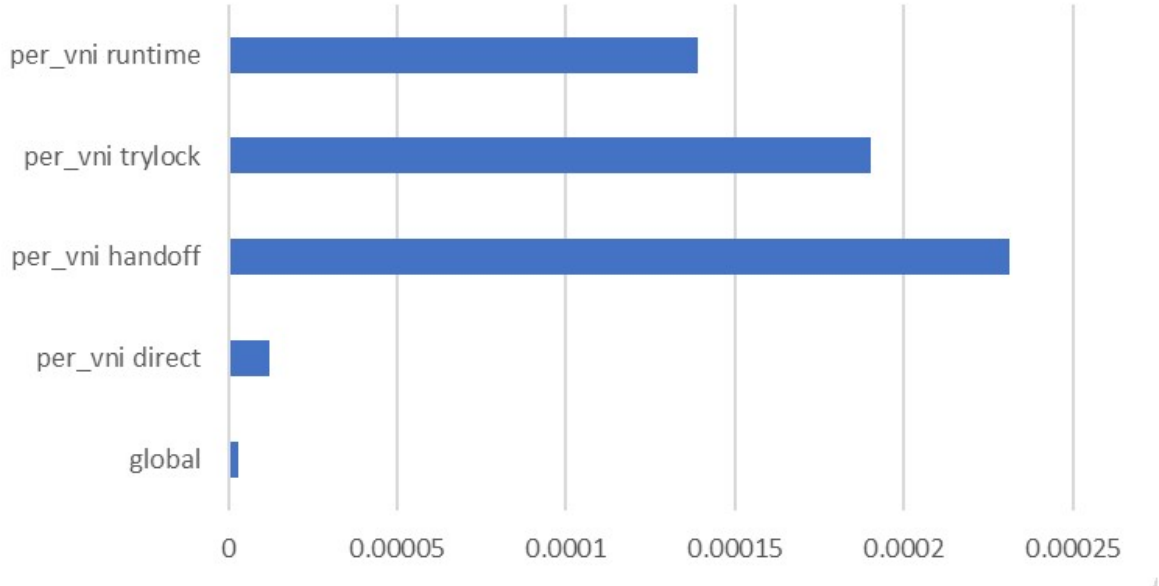

Fig. 32 - Mutex timing

which refers to an interval of time during which a thread of execution accesses a shared resource, such as shared data objects, shared resources, shared memory.

In MPI library, it used a mutex lock to protect the shared resources. Here, we want to measure the mutex count and timing to understand the influence of the mutex time to thetotal running time.

Figure 33 and 34 show mutex count and timing when sender and receiver has only one processor and one thread, correspondingly.

As we can see here, global acquires less mutex lock and spends less time on mutex locking.

CONCLUSIONS

By analyzing the above experiments, we found that it seems that coarse granularity consumes less time. This seems to be the opposite of the original paper's results. We repeated the experiments on different experimental platforms and different MPI versions. Still got this result.

We think it may be since the data in this experiment are not mutually exclusive, each is independent. There is no need to add too many mutex locks. Too many mutex locks create additional overhead.

If the data in the original paper is true, then this may be due to hardware developments, with processors supporting more threads these days. Various hardware limitations in the early days made it impossible to run subprograms at the same time.

In the case of high concurrency (high contention), the performance of mutex locks is generally higher than that of lock-free ones, which is due to the high cost of hardware-level mutual exclusion and a large number of CAS operations. [51]

So, let's review the advantages of mutex.

Simple to implement

Tends to be faster in high contention situations

shortcoming

deadlock livelock

hunger

priority flip

Lock free pros and cons

advantage

Avoid deadlocks and livelocks, priority inversion

Sometimes the delay is small

shortcoming

ABA problem, memory order

The code is complex and difficult to implement

lead to more cache misses

OpenMP is faster than Pthread for the most of programs?

Courterexample Quick Sort

openmp is based on the SIMD model, and if it is to implement MIMD, the code tends to be more complex than pthreads.

OpenMP is faster than Pthread for matrix multiplication and Mandelbrot set calculation, however Pthread tends to be faster for quicksort. This is because OpenMP has some problems with recursion. Therefore, recursive programs are recommended to use Pthread [52].

When the program supporting OpenMP, we can usually get about 4x speedup by using OpenMP [53].

By using our new model LogPCK, we should consider two more arguments, C and K. It means that lock-free is not always the fast, since CAS is a high hardware cost operation. When we decide to use mutex lock, we should also consider the contention time, whether it is worthy to change to lock-free algorithms

FUTURE PLAN

We have analyzed the bottlenecks and limitations of current computer development to know what is not computable and what can be computed efficiently.

Program parallelization, increasing the number of processors does improve performance. But this is a very limited improvement, such as the best-suited application, the best-case linear speedup can be obtained. But this is not universal.

Another algorithm to improve the speed of the program is to optimize the algorithm to reduce the computational complexity of the program. My Ph.D. study is in the study of how to optimize NP-hard problems for computational efficiency and accuracy.

ACKNOWLEDGEMENT

Finally, I would like to thank my mentor, Alexey Paznikov, who taught me a lot about doing research. Thanks to my classmates, they made me discover a bigger world. Thanks to Daria from the International Student Office at ETU, she helped me with many of the difficulties that international students would encounter, especially visa issues. Thanks to the staff at HSE's Student Dormitory No. 4, they taught me to have the courage to fight for my rights. Thanks to all the Russians who helped me, from the Far East to St. Petersburg. I would like to thank the Russian Foundation Research Fund for partially subsidizing me, and I would like to thank the China Scholarship Council for fully subsidizing my second year of study.

BIBLIOGRAPHY

1. Hennessy, John L., and David A. Patterson. Computer architecture: a quantitative approach. Elsevier, 2011.

2. Patterson, David A., and John L. Hennessy. Computer organization and design ARM edition: the hardware software interface. Morgan kaufmann, 2016.

3. Wikimedia Foundation. (2022, April 7). Supercomputer. Wikipedia. – [Electronic resource] URL: https://zh.wikipedia.org/wiki/%E8%B6%85%E7%BA%A7%E8%AE%A1%E7%AE%97%E6%9C%BA (date of access: 17.05.2022)

4. Hennessy, John L. "The 50 Year History of the Microprocessor as Five Technology Eras." IEEE Micro 41.6 (2021): 20-21.

5. MPI and Hybrid Programming Models William Gropp - [Electronic resource].URL: https://extremecomputingtraining.anl.gov//files/2014/01/B.Gropp1_.00hybrid-2014.pdf (date of access: 17.05.2022).

6. Hybrid Programming Alice Koniges, Berkeley Lab/NERSC – [Electronic resource]. URL: https://www.nersc.gov/assets/Training-Materials/hybridTalk.pdf (date of access: 17.05.2022).

7. Hasanov, Khalid, and Alexey Lastovetsky. "Hierarchical redesign of classic MPI reduction algorithms." The Journal of Supercomputing 73.2 (2017): 713-725.

8. Wes Kendall. Point-to-Point Communication Application - Random Walk – [Electronic resscource]. URL: https://mpitutorial.com/tutorials/point-to-point-communication-application-random-walk/ (date of access: 17.05.2022)

9. Wes Kendall. MPI Broadcast and Collective Communication – [Eletronic rescource]. URL: https://mpitutorial.com/tutorials/mpi-broadcast-and-collective-

communication/ (date of access: 17.05.2022)

10. Supercomputer ranking and benchmarks - [Electronic resource]. URL: https://blog.csdn.net/swingwang/article/details/73752896 (date of access: 17.05.2022)

11. BENCHMARK SPECIFICATION - [Electronic resource]. URL: https://graph500.org/?page_id=12 (date of access: 17.05.2022)

12. Gyrokinetic Toroidal Code at Princeton (GTC-P) - [Electronic resource]. URL: https://extremescaleglobalpic.princeton.edu/gtcp (date of access: 18.05.2022)

13. HPCG Benchmark - [Electronic resource]. URL: https://www.hpcg-benchmark.org/index.html (date of access: 19.05.2022)

14. Adiga, Narasimha R., et al. "An overview of the BlueGene/L supercomputer." SC'02: Proceedings of the 2002 ACM/IEEE Conference on Supercomputing. IEEE, 2002.

15. Chen, Yunji, et al. "Dadiannao: A machine-learning supercomputer." 2014 47th Annual IEEE/ACM International Symposium on Microarchitecture. IEEE, 2014.

16. Tabakov, Andrey V., and Alexey A. Paznikov. "Algorithms for optimization of relaxed concurrent priority queues in multicore systems." 2019 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (EIConRus). IEEE, 2019.

17. Fig 4 - uploaded by Beichuan Yan - [Electronic resource]. URL: https://www.researchgate.net/figure/Schematic-of-typical-architecture-of-a-modern-supercomputer_fig1_328946498 (date of access: 10.05.2022)

18. Japan’s Fugaku retakes supercomputing crown from US and China - [Electronic resource]. URL: https://www.i-micronews.com/japans-fugaku-retakes-supercomputing-crown-from-us-and-china/ (date of access: 15.05.2022)

19. TOP500 - [Electronic resource]. URL: top500.org (date of access: 18.04.2022)

20. Frequently used benchmarks for supercomputer - [Electronic resource]. URL: https://blog.csdn.net/zbgjhy88/article/details/79134116 (date of access: 12.05.2022)

21. About Fugaku - [Electronic resource]. URL: https://www.r-ccs.riken.jp/en/fugaku/about/ (date of access: 10.05.2022)

22. The tofu interconnect D - [Electronic resource]. URL: https://www.fujitsu.com/hk/imagesgig5/08514929.pdf (date of access: 10.05.2022)

23. Chapter 16 - Instruction-level-parallelism-and-superscalar-processors-flash-cards - [Electronic resource]. URL: https://quizlet.com/31893528/chapter-16-instruction-level-parallelism-and-superscalar-processors-flash-cards/ (date of access: 02.05.2022)

24. Data parallelism. Wikipedia - [Electronic resource]. URL: https://en.wikipedia.org/wiki/Data_parallelism (date of access: 17.05.2022)

25. Computer architecture. Thread parallelism - [Electronic resource]. URL: https://blog.csdn.net/u014030117/article/details/46591877 (date of access: 12.05.2022)

26. Figure 2 - uploaded by David Alexander Beckingsale - [Electronic resource]. URL: https://www.researchgate.net/figure/The-fork-join-model-used-

for-thread-level-parallelism-in-OpenMP_fig2_305806654 (date of access: 13.05.2022)

27. Single instruction, multiple data. Wikipedia - [Electronic resource]. URL: https://ru.wikipedia.org/wiki/SIMD (date of accesss: 16.05.2022)

28. Multiple instruction, multiple data. Wikipedia - [Electronic resource]. URL: https://ru.wikipedia.org/wiki/MIMD (date of access: 17.05.2022)

29. Moore's Law. Wikipedia - [Electronic resource]. URL: https://ru.wikipedia.org/wiki/%D0%97%D0%B0%D0%BA%D0%BE%D0%BD_%D0%9C%D1%83%D1%80%D0%B0 (date of access: 19.05.2022)

30. Figure 26 - uploaded by Luke Shulenburger. - [Electronic resource]. URL: https://www.researchgate.net/figure/The-end-of-Dennard-Scaling-44_fig24_301650491 (date of access: 11.05.2022)

31. Burleson Consulting. Oracle performance, hardware & RAM tuning optimization - [Electronic resource]. URL: http://www.dba-oracle.com/oracle_tips_hardware_oracle_performance.htm (date of access: 12.05.2022)

32. PRAM Model of computation : features & constraint - [Electronic resource]. URL: https://er.yuvayana.org/pram-model-of-computation-features-constraint-pram/ (date of access: 17.05.2022)

33. PARALLEL ALGORITHM - [Electronic resource]. URL: https://er.yuvayana.org/category/parallel-algorithm/ (date of access: 11.05.2022)

34. Gustafson's law. Wikipedia - [Electronic resource]. URL: https://en.wikipedia.org/wiki/Gustafson%27s_law (date of access: 16.05.2022)

35. Figure 7 - uploaded by Cristobal A Navarro - [Electronic resource]. URL: https://www.researchgate.net/figure/An-example-communication-using-the-LogP-model_fig10_256495766 (date of access: 09.05.2022)

36. Calculate the time of an all-to-all broadcast for a balanced binary tree (BBT) - [Electronic resource]. URL: https://www.physicsforums.com/threads/calculate-the-time-of-an-all-to-all-broadcast-for-a-balanced-binary-tree-bbt.970004/ (date of access: 07.05.2022)

37. Broadcast (parallel pattern). Wikipedia - [Electronic resource]. URL: https://en.wikipedia.org/wiki/Broadcast_(parallel_pattern) (date of access: 05.05.2022)

38. Fig 1 - uploaded by Ralf Hinze - [Electronic resource]. URL: https://www.researchgate.net/figure/Fibonacci-tree-of-height-seven-fib-tree-7_fig1_220676591 (date of access: 02.05.2022)

39. Woodsend K, Gondzio J. Hybrid MPI/OPENMP parallel linear support vector machine training[J]. Journal of Machine Learning Research, 2009, 10 : 1937-1953.

40. Waghmare V N, Kulkarni D B. Convex hull using k-means clustering in hybrid(MPI/OpenMP) environment[C]// Proceedings of the International Conference on Computational Intelligence and Communication Networks, Piscataway, 2010: 150-153.

41. Hennessy, John L., and David A. Patterson. Computer architecture: a quantitative approach. Elsevier, 2011.

42. Balaji, Pavan, et al. "Fine-grained multithreading support for hybrid threaded MPI program- ming." The International Journal of High Performance Computing Applications 24.1 (2010): 49-57.

43. Amer A. et al. MPI+ threads: Runtime contention and remedies //ACM SIGPLAN Notices. 2015. V. 50. – 8. pp. 239-248.

44. MPI Pthread tutorial - [Electronic resource]. URL: https://computing.llnl.gov/tutorials/pthreads/ (date of access: 15.11.2020)

45. Boroumand, Amirali, et al. "Google workloads for consumer devices: Mitigating data move- ment bottlenecks." Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems. 2018.

46. MPI Documentation - [Electronic resource]. URL: https://www.mpich.org/documentation/guides/ (date of access: 15.11.2020)

47. MPI: blocking vs non-blocking - [Electronic resource]. URL: https://stackoverflow.com/questions/10017301/mpi-blocking- vs-non-blocking (date of access: 15.11.2020)

48. OpenMP Official website - [Electronic resource]. URL: https://www.OpenMP.org (date of access: 25.11.2020)

49. OpenMP Tutorial at Supercomputing 2008 - [Electronic resource]. URL: https://www.openmp.org/uncategorized/openmp- tutorial-at-supercomputing-2008/ (date of access: 25.11.2020)

50. Using OpenMP вЂ“ Portable Shared Memory Parallel Programming вЂ“ Download Book Examples and Discuss - [Electronic resource]. URL:

https://www.openmp.org/uncategorized/download-book-examples-and- discuss (date of access: 23.11.2022)

51. Efficiency summary of lock-free programming and lock-free programming, implementation of lock-free queue - [Electronic resource]. URL: https://blog.csdn.net/qq_42214953/article/details/105750215 (date of access: 24.05.2022)

52. Pthreads and OpenMP, A performance and productivity study - [Electronic resource]. URL: http://www.diva-portal.org/smash/get/diva2:944063/FULLTEXT02 (date of access: 15.05.2022)

53. Harry Davis, Is pthreads faster than OpenMP? - [Electronic resource]. URL: https://quick-adviser.com/is-pthreads-faster-than-openmp/ (date of access: 17.05.2022)