Saint Petersburg Electrotechnical University
(ETU)

Direction	09.04.01 – Computer Systems Engineering and Informatics
Program	Computer Science and Knowledge Discovery
Faculty	Computer Science and Technology
Department	Information Systems

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

MASTER'S GRADUATE QUALIFICATION WORK

Topic: ALGORITHMS FOR IMPLEMENTING COLLECTIVE OPERATIONS IN THE REMOTE MEMORY ACCESS MODEL FOR DISTRIBUTED COMPUTING SYSTEMS

Student _____ M. Abuelsoud
signature

Supervisor _____ A.A. Paznikov
(*PhD,associate professor*) *signature*

Advisors _____ I. V. Medynskaya
(*B.E,Professor*) *signature*

_____ O. I. Brikova
signature

_____ N. A. Nazarenko
(*PhD,Associate Professor*) *signature*

Saint Petersburg
2023

TASK FOR THE GRADUATE QUALIFICATION WORK

APPROVED

Head of Information
Systems department

_____ V. V. Tcehanovsky

«____»_____ 2023 г.

Student M. Abuelsoud Group 7300

Title: Algorithms for Implementing Collective Operations in the Remote Memory Access Model for Distributed Computing Systems

Institution: Saint Petersburg Electrotechnical University ”LETI”

Source data (or technical requirement)): The technical requirements as well as the input data sets or data formats are specified

Contents: Broadcast implemented by Binomial,Binary and Linear sequential tree algorithms and the deference between sequential programming and message passing paradigm in addition to that shared memory model and finally the result of our research in multiple libraries

List of the reporting materials: text of graduate qualification work, figures, other reporting materials

Additional Chapters: Drafting a business plan for the Master’s thesis result commercialisation.

Date of task assignment
«____»_____ 2023 г.

Date of GQW submission
«____»_____ 2023 г.

Student		_____	M. Abuelsoud
		signature	
Supervisor		_____	A.A. Paznikov
	(*PhD,associate professor*)	*signature*	
Advisor		_____	I. V. Medynskaya
	(*B.E,Professor*)	signature	

CALENDAR PLAN
FOR THE GRADUATE QUALIFICATION WORK IMPLEMENTATION

APPROVED

Head of Information
Systems department

_____ V. V. Tcehanovsky

«____»_____ 2023 г.

Student M. Abuelsoud Group 7300

Title: Algorithms for Implementing Collective Operations in the Remote Memory Access Model for Distributed Computing Systems

№	Tasks	Due dates
1	Parallel Commuting	01.04 - 10.04
2	HYBRID MPI	10.04 - 17.04
3	PROGRAMMING PARALLEL PARADIGM	26.04 - 05.05
4	COLLECTIVE OPERATIONS	06.05 - 11.05
5	THE RESULTS OF THE RESEARCH (DEVELOPMENT)	11.05 - 15.05
6	Commercialization of the Research's Results	25.04 - 01.05

Student _____ M. Abuelsoud
signature

Supervisor _____ A.A. Paznikov
(*PhD,associate professor*) *signature*

Advisor _____ I. V. Medynskaya
(*B.E,Professor*) *signature*

ABSTRACT

The explainotory note consists of 79 p., 40 fig., 16 tables, 34 ref.

MPI,RMA,MPI SHARED MEMORY MODEL, COLLECTIVE COMMUNICATION, HYBRID PROGRAMMING,LOGP,LOGGP,PLOGP

The object of the research is The effectiveness of collective communication operations affects MPI application execution time. For one-to-all communication schemes (MPI_Bcast, MPI_Scatter, etc), many algorithms have been created and put into use, although none consistently outperformed the others.

As communication is accomplished through shared memory, system calls are only needed to set up the shared memory, giving shared memory a maximum speed of calculation. On the other hand, the message passing model is extremely time-consuming because it uses the kernel. When two processes frequently exchange large amounts of data, shared memory performs excellently. This advantage would be useful for collective algorithms because we will be sending a lot of data between processes.

Every item of data sent during message transmission necessitates two system calls: one for reading and one for writing. Also, two copies of the sent data must be made: one for the kernel memory and one for the receiving process. This time penalty is negligible for a single message and is unlikely to have an impact on the process's performance.

The goal of the research The cumulative impact of this penalty, though, may be substantial if a lot of messages are transmitted. We have chosen the binomial tree, binary tree, and Flat tree (linear) algorithms. Shared memory model for the RMA collective algorithm (Broadcast) we have compared them with different message sizes (16 Bytes to 34 MB).

DEFINITIONS AND ABBREVIATIONS

The following definitions and abbreviations are used in this explanory note to the graduate qualification work.

C - The number of of transmissions to transmit the the data between one process to another .
T - defined as the time it takes for one processor to transfer data of size m to one other processor in one operation.
d - the size of data in this paper we used float.
P - the number of processes in the communication group.
T_s - Time to transmit the data T_D for large d ($d * T_d$).T_s is not big but it become bigger when we decrease d.
T_{Total} - the time required for the broadcast to complete (the time at which all data has been transmitted to all processors).

CONTENTS

INTRODUCTION

Message Passing Interface (MPI) is the standard that is used to design parallel programs for distributed-memory computers.

Access to memory remotely. Remote Memory Access (RMA), commonly referred to as one-sided MPI, is one of the most promising parallel programming paradigms that has recently been presented. RMA provides unofficial one-sided communication methods that let a process do a ”put” or ”get” operation—writing data to or retrieving it from another processor—without contacting the other processor [1, 2]. RMA communications have been proven to be advantageous for many applications by substituting direct remote memory accesses for transmitting and receiving messages. Several strategies are used while developing with remote memory access (RMA). The majority of distributed memory machines presently support the Remote Direct Memory Access technology in terms of hardware [2].

Users may only utilize straightforward linear (sequential) algorithms under the existing MPI standard [3], which is far from ideal when the program's semantics support many parallel data accesses. Designing algorithms for these collective operations (collectives) under the RMA model is the main objective of this research. We anticipate that these algorithms will perform much better than sequential algorithms by speeding up MPI applications' execution and using less energy.

This article describes our attempts to improve the efficiency of RMA collective operations(Broadcast) within Open MPI [4]. In this technique, the direct execution of these operations based on a combination of shared and remote memory substitutes for point-to-point message passing (MPI_Send/MPI_Recv), normally utilized in the development of message-passing collectives. In order to reduce data transfers, improve flow control, and reduce internal overheads [5].

The performance is enhanced because there are fewer data movement operations in the shared memory implementations of collectives than there are in point to point message passing implementations. Second, significant savings are made by eliminating the costs of tag matching, coping with early message arrivals, and the difficulties of buffer management involved in general purpose point-to-point message passing systems[5].

Our approach is to use shared memory to make the Broadcast collective algorithms using the Binomial tree algorithm. We will be comparing our approach against the binary

tree and Linear sequential tree algorithms [6]. One of the MPI-defined subroutines for group communication is the broadcast function. Its function is to write data from a single processor to every other processor in a communications group. When all of the processors in a group need to exchange partial results from one processor in order for the calculation to proceed, for instance, the broadcast is applied. This paper looks at Binomial tree, Binary tree, and Linear sequential algorithms for implantation broadcast function using shared memory and shows the difference in performance. However, When using shared memory, you don't need to wait because the usage of passive and multi-threaded calls helps us to make synchronization calls [7].

1 RELATED WORK

Collective algorithms have historically received extensive research. There are several algorithms for each collective, with distinct algorithms catering to various message sizes and processes[8, 9].

In the hybrid MPI+MPI version, the collectives can be skipped if the process advances the subsequent calculation by just utilizing the received data as information without changing it, unlike in the pure MPI version, where the collectives need a copy of the received data for each process. [10]. Collectives [9, 10] only keep one copy of the replicated data that is shared by all on-node processes, in contrast to the normal MPI collectives. The explicit on-node inter-process data exchanges are entirely removed in this manner. To ensure the expected to receive of the shared data between the on-node processes, synchronization calls must nevertheless be sufficiently included. This can resolve the problem of memory usage in large-scale systems. According to the application kernel assessments, collective hybrid MPI+MPI implementations outperform pure MPI and hybrid MPI+OpenMP implementations in terms of time performance.

MPI libraries and other parallel programming models (such as PGAS - Partitioned Global Address Space) implement collectives using the ring algorithm, recursive doubling, recursive halving, J. Bruck algorithm [11], and algorithms organizing processes in various tree structures: balanced k-ar trees, flat trees , pipelines (k-chains) [12–14].

The algorithms mentioned above have distinct execution times and are founded on the assumption of uniformity of communication channels between the computing elements. However, modern systems include multiple architectures and have a hierarchical structure, and the duration for data transmission between two processor components depends on their locality in the system [15, 16].Therefore the relevant problem is the development of topology-aware collectives, which will account for the multi-architecture and hierarchical structure of modern HPC systems.

A one-to-all strategy The binomial tree algorithm, k-chain tree algorithm, binary tree algorithm, flat/tree method, and algorithm based on MPI_Scatter (binomial tree) with MPI_Allgather (ring, recursive doubling) are used to implement the operations MPI_Bcast and MPI_Ibcast. Binomial tree algorithm, flat/linear tree method, and MPI_Scatter, MPI_Iscatter, MPI_Scatterv, are used to accomplish these operations.

To the best of our knowledge, MPI one-sided interface (RMA model) techniques have not been presented. However, certain processes may add or remove particular data to or from all other processes' memory in RMA-programs (especially when employing passive target synchronization). It adheres to the well-known all-to-all, all-to-one, and one-to-all collective communication methods. In this situation, a simple linear algorithmic method (complexity $O(n)$) will not provide the optimal results. Even though the MPI one-sided interface does not now provide collective operations, we anticipate that this feature will be highly needed in the next MPI standards, particularly for the next generation of high performance computing systems. We anticipate that these collectives will significantly reduce communications (in comparison to the linear method) and consequently, overall execution time.

2 PARALLEL COMMUTING

Microprocessor performance improved by 50% year on average between 1986 and 2002[17].Because of this exceptional growth, customers and software developers may often just wait for the following generation of microprocessors to get an application program with a higher performance level. However, since 2002, the rate of improvement in single-processor performance has slowed to about 20% annually. The contrast is stark: performance will grow by roughly a factor of 60 at 50% each year in 10 years, but only by around a factor of 6 at 20%.

Additionally, a significant modification in processor design has been linked to this variation in performance increase. By 2005, the majority of the major microprocessor makers had come to the conclusion that parallelism was the way to go if you wanted performance to increase quickly. Instead of attempting to continue to create monolithic processors that are ever quicker, manufacturers began integrating numerous full processors onto a single integrated circuit.This modification has a crucial repercussion for software developers: the great majority of serial applications, or programs designed to operate on a single processor, won't instantly perform better just by adding additional processors. These programs are blind to the presence of many processors, therefore their performance on a system with multiple processors will be essentially identical to that of the program on a single processor of the multiprocessor system.

– **Reasons Why Constant Improvement Is Necessary**

  Many of the most significant developments in industries as diverse as science, the Internet, and entertainment have their roots in the enormous increases in computational power that we have been enjoying for several decades. Decoding the human genome, ever-more-precise medical imaging, astoundingly quick and accurate Web searches, and ever-more-realistic computer games are just a few examples. All of these things would not have been possible without these advancements. Indeed, without earlier increases in computational power, more recent increases would have been challenging, if not impossible. But we must never get comfortable. The amount of issues we can seriously contemplate tackling grows as our processing capacity improves.Here are a few illustrations:

  1. Data analysis :We produce an enormous quantity of data. The amount of data saved globally is said to double every two years, yet the bulk of it is mostly worthless until it

is examined. For instance, it doesn't help much to know the order of the nucleotides in human DNA on its own. It takes substantial investigation to comprehend how this sequence influences development and how it may lead to illness. The Large Hadron Collider at CERN, medical imaging, astronomical research, Web search engines, etc., in addition to genomics, produce enormous amounts of data.

2. Climate modeling:We need far more precise computer models, models that encompass interactions between the atmosphere, the seas, the solid land, and the ice caps at the poles, to better comprehend climate change. We also need to be able to conduct in-depth analyses of the potential effects of different initiatives on the global climate.

– Building parallel systems

The incredible rise in single processor performance has been largely fueled by integrated circuit transistor densities, which are the electronic switches. Transistors' speeds may be raised as their sizes shrink, which also speeds up the integrated circuit as a whole. However, as transistor speed increases, so does the amount of energy they use. The majority of this power is lost as heat, and an integrated circuit loses reliability if it becomes too hot. Air-cooled integrated circuits are nearing the limits of their heat dissipation in the first decade of the twenty-first century.It is thus becoming difficult to keep improving the speed of integrated circuits. However, the growth in transistor density is still possible—at least temporarily. Additionally, there is a moral obligation to keep boosting computational power given the potential of computing to enhance our existence. Finally, the integrated circuit business will essentially vanish if it doesn't keep creating new, improved devices.

So how can we take advantage of the ongoing rise in transistor density? Parallax is the solution. The industry has chosen to combine many, relatively basic, full processors on a single chip rather than creating ever-faster, more sophisticated, monolithic CPUs. Multicore processors are what these integrated circuits are known as, and the term "core" has evolved to mean the central processing unit, or CPU. In this context, a standard processor with one CPU is sometimes referred to as a single-core machine.

– Write parallel programs

The majority of applications designed for traditional, single-core computers are unable to take use of the availability of multiple cores. On a multicore system, we can run multiple instances of a program, but this is frequently ineffective. For instance, we don't actually want to be able to run several instances of our favorite gaming software;

instead, we want the program to run quicker and display images that are more accurate. To do this, we either need to build translation programs, that is, programs that will automatically transform serial programs into parallel programs, or we need to modify our serial programs so that they are parallel and can use multiple cores. The bad news is that relatively few academics have succeeded in creating software that transforms serial programs written in languages like C and C++ into parallel applications. This is not very shocking. Although we can create programs that automatically translate common serial program constructs into effective parallel program constructs, the order in which these parallel program constructs are used may be incredibly inefficient.

There is a quick overview of parallel computing in this chapter. It will cover the architecture of modern parallel systems and attempt to provide a concise summary of the state-of-the-art in programming techniques for these systems.

2.1 Hardware

The hardware architecture of parallel computing is distributed along the following categories as given below

1. Single-instruction, single-data (SISD) systems
2. Single-instruction, multiple-data (SIMD) systems
3. Multiple-instruction, single-data (MISD) systems
4. Multiple-instruction, multiple-data (MIMD) systems

Refer to learn about the hardware architecture of parallel computing.

2.1.1 Flynn's Taxonomy

Flynn's taxonomy, the first categorization of parallel computers, is well-known. Michael Flynn categorized systems in 1966 based on the quantity of instruction streams and data streams.The classical von Neumann machine has mentioned in section 2.1.

2.1.2 The Classical von Neumann Machine

A CPU and main memory make up the typical von Neumann computer. The control unit and the arithmetic-logic unit (ALU) are additional divisions of the CPU. Data and instructions are both stored in the memory. Program execution is guided by the control unit, and the ALU performs the computations specified by the program. Instructions and

data are kept in registers, which are very quick memory locations, until they are needed by the program.

Between memory and the registers of the CPU, both data and program instructions are transferred. The path they take is referred to as a bus. In essence, it consists of a network of parallel wires and technology for managing bus access.

Before the traditional von Neumann machine is functional, it requires several extra components: input and output devices, and often expanded storage devices like a hard disk.

The von Neumann bottleneck is the rate at which we can transmit the sequence of instructions and data between memory and the CPU. No matter how fast we build our CPUs, the pace at which we can execute programs is constrained by this rate. Few computers nowadays are strictly traditional von Neumann devices as a consequence. For instance, the majority of computers nowadays include a hierarchical memory structure, which includes cache, an intermediate memory that is quicker than main memory but slower than registers, in addition to main memory and registers. Caches are based on the fact that programs often access data and instructions in a sequential manner. Therefore, the majority of program memory accesses will use the fast memory rather than the slower main memory if we store a small block of data and a small block of instructions in fast memory.

SIMD Systems

A single control-only CPU and a large number of assistant ALUs, each with a tiny amount of memory, are features of SIMD. The control processor transmits a signal to all of the subordinate processors throughout each instruction cycle, and each subordinate processor either executes the instruction or does nothing.

The drawbacks of a SIMD system are evident: it is completely conceivable for many processes to be idle for extended periods of time in a program with several conditional branching or lengthy pieces of code whose execution relies on conditionals [18–20].

MIMD Systems

The main distinction between MIMD and SIMD systems is that each processor in a MIMD system is an independent CPU that has both a control unit and an ALU. As a result,

each processor has the ability to carry out its own program at its own speed. Particularly, MIMD systems are asynchronous in contrast to SIMD machines. Even though several processors are running the same software, there may not be a correlation between what is happening on each CPU unless the processors are expressly built to synchronize with one another. This is true even when there is a global clock[18–20].

The world of MIMD systems is divided into

1. shared-memory system.
2. distributed-memory system.

A network connects the processors and memory modules that make up the generic shared memory machine Figure 2.1. The types of the Architectures

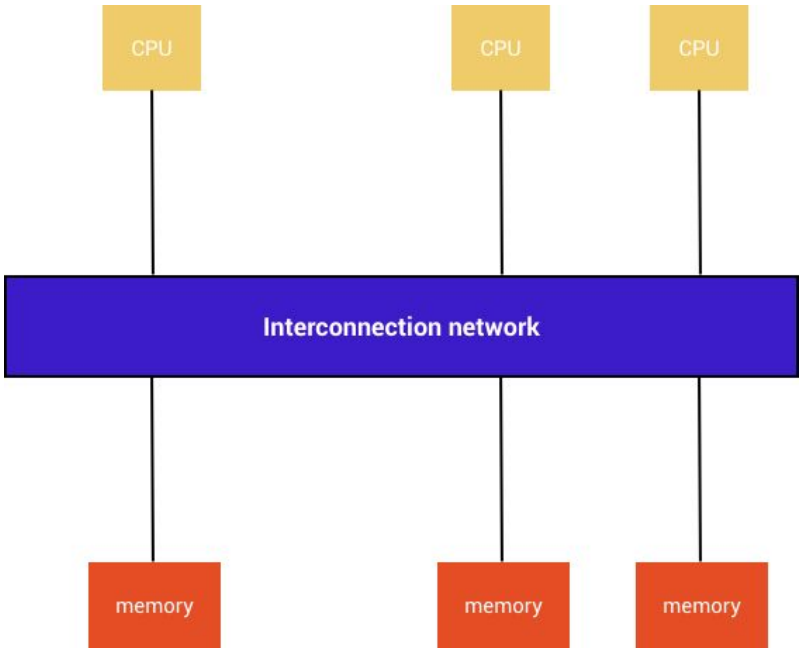


Figure 2.1 – Generic shared-memory architecture

– **Bus-based** connectivity networks are the most basic. The bus will become overloaded if several processors try to access memory at once, and there may be significant delays between initiating a retrieve or store and actually transferring the data.As a result, each CPU often has access to a sizable cache shown in Figure 2.2. These systems are not scalable to many processors due to the constrained bus bandwidth.
– **Switch Based** A crossbar switch may be seen as a rectangular mesh of wires with terminals on its top and left edges and switches at the places of intersection. The connectors may be used to connect processors or memory modules shown in Figure 2.3. The switches have two different signal-passing capabilities: they may either enable a signal to go in both the vertical and horizontal directions at once, or they can switch a

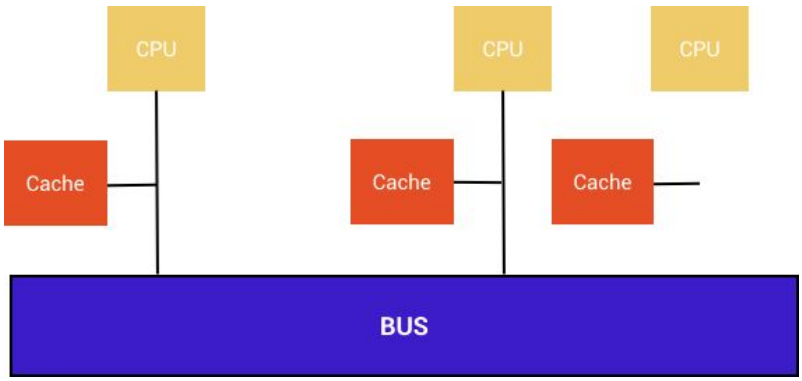


Figure 2.2 – Bus-based shared-memory architecture

signal from vertical to horizontal or vice versa .As a result, any processor may access any memory module, for instance, if processors are on the left and memory modules are at the top of the crossbar. Additionally, any other processor can access any other memory module concurrently. Communication between two units won't obstruct communication between any other two units, in other words. Therefore, crossbar switches do not experience the saturation issues that we did with buses.

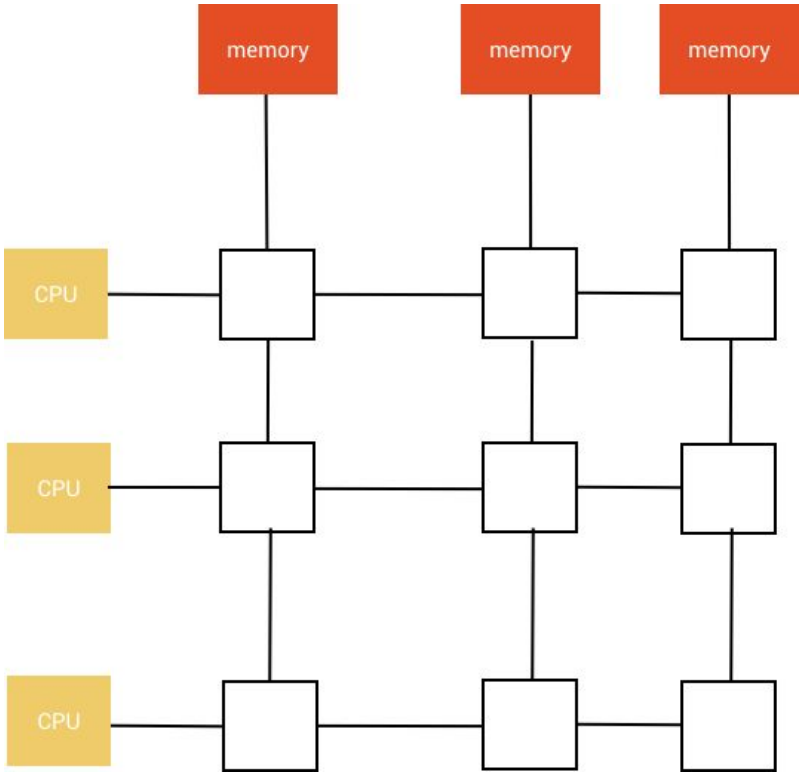


Figure 2.3 – Crossbar switch bus

– **Cache Coherence** or Cache consistency is a challenge with any shared-memory architecture that permits the caching of shared variables. How can a processor determine if

a shared variable in its cache has a value that is current? So, let's say processor A needs to access the shared variable x that is stored in its cache. How can A be certain that another process B hasn't changed its copy of x, making A's copy obsolete? There are several cache consistency techniques, and their level of complexity varies greatly. The snoopy protocol, which is appropriate for tiny bus-based devices, is most likely the simplest. The fundamental concept is that each processor has a cache controller in addition to the typical hardware connected to a CPU. The cache controllers "snoop" on the bus, or watch the traffic on the bus, among other things. A processor also modifies the associated main memory location when it modifies a shared variable.When a write is made to main memory, the cache controllers on the other processors notice it and label their copies of the variable as invalid. Take note of how the bus enables this: All controllers can keep an eye on any traffic on the bus. As a result, this strategy is inappropriate for other classes of shared-memory devices.

Each CPU in a distributed memory system has access to its own private memory. As a result, Figure 2.4 may serve as a general representation of a distributed memory system. There are two main categories of graphs if we think of a distributed memory system as a graph, with the edges representing the communication wires: graphs where each vertex represents a processor/memory pair, or node, and graphs where some vertices represent nodes and others represent switches.

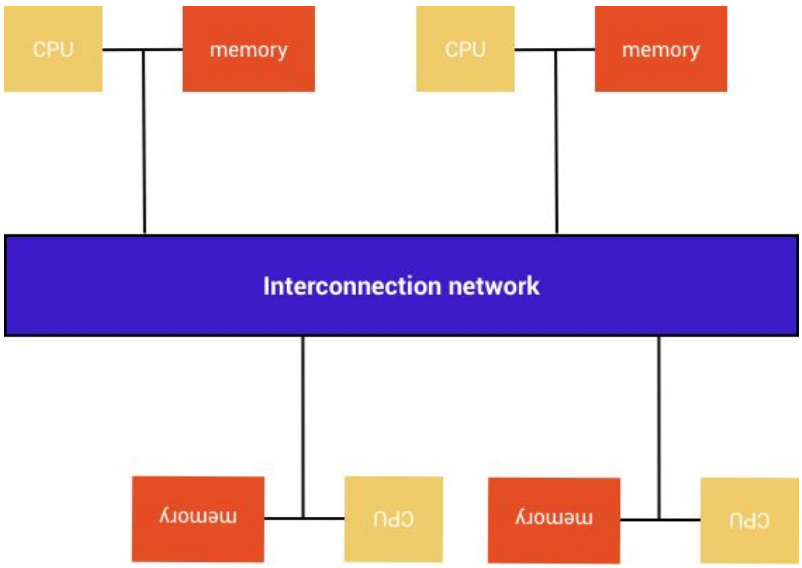


Figure 2.4 – Generic distributed memory system

2.1.3 PC Clusters

Cluster computing, which is built, for instance, on PCs running the Linux operating system, is the most well-liked and economical method of parallel computing. The communication network that links the PCs will determine how successful this method is; options include fast Ethernet and Myrinet, which may broadcast messages at a rate of several Ggabits per second (Gbs) [20].

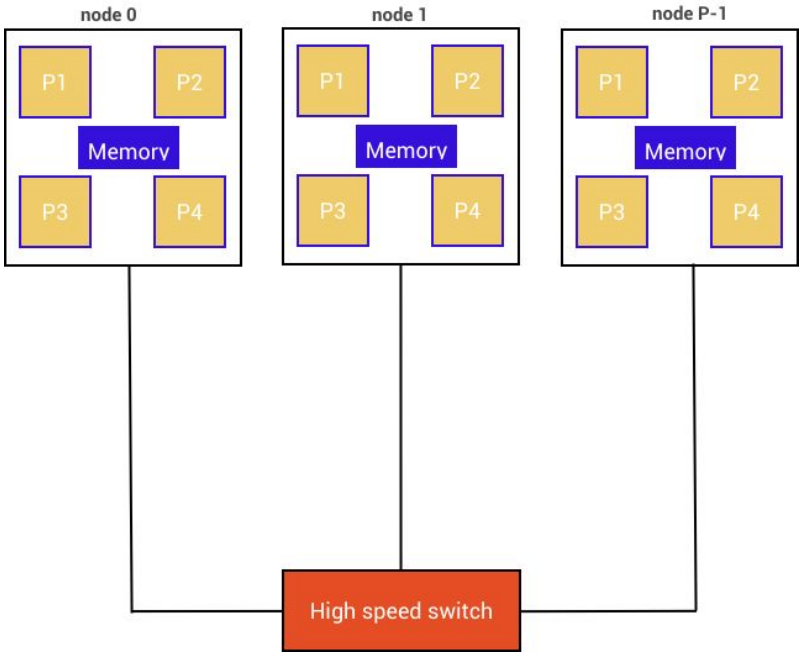


Figure 2.5 – Generic Parallel Computer (GPC)

By looking at the Generic Parallel Computer (GPC) as illustrated in Figure 2.5, issues of computer architecture, balancing memory, network speed, and processor performance may be solved.An interconnected group of P processing elements (PE) with distributed local memories, a shared global memory, and a quick disk system (DS) make up the GPC's main building blocks. The GPC serves as a model for the majority of PC-based clusters that have dominated supercomputing in the last ten years on both the scientific and business fronts.

There have been a number of significant advancements that have made commodity super computing possible, in addition to improvements in processor performance.

– the growth and maturity of Linux, a free operating system that is now accessible on all computer systems. Almost all PC clusters are based on Linux because to the freely distributable system and the open source software movement making it the preferred operating system.

– Parallel coding has become portable and simple thanks to the MPI standard. There are several implementations, including OpenMPI, SCore, and MPICH, but they all use the same fundamental instructions.
– Contrary to early days of exclusive and expensive systems available only by a few big vendors, today's fast switches with small latencies and rapid interconnect advancements are now widely available.

2.2 Software

Parallel hardware is now available. Multi-core processors are used in almost all desktop and server systems. Parallel software cannot be stated to be the same. There is presently relatively little commodity software that extensively utilizes parallel hardware, with the exception of operating systems, database systems, and Web servers. This is an issue since we can no longer depend on hardware and compilers to guarantee a constant rise in application performance, as we pointed out in Section 2.1. Software developers need to understand how to create applications that take advantage of shared- and distributed-memory architectures if we want to continue seeing regular advances in application speed and power. We'll examine some of the challenges associated with developing software for parallel systems in this section.

Firstly, some definitions. We often launch a single process and fork many threads when running our shared-memory apps. So, we'll talk about threads performing tasks when we discuss shared-memory programs. On the other hand, while running distributed memory applications, we launch many processes and refer to the tasks that the processes are doing. We'll speak about processes/threads doing tasks when the subject is applicable to both shared-memory and distributed memory systems.

2.2.1 Distributed memory

The only private memories that the cores may have direct access to in distributed memory applications are their own. There are several APIs in use. But message-passing is by far the most widely used. Therefore, message-passing will receive the majority of our attention in this section. Then, we'll quickly review a few additional, less popular APIs. The fact that distributed-memory APIs may be utilized with shared-memory hardware is perhaps the most important thing to keep in mind. Programmers may conceptually

divide shared memory into separate private address spaces for each thread, and a library or compiler can handle the necessary communication. As we previously said, distributed-memory systems are often run by launching several processes as opposed to numerous threads. This is because typical ”threads of execution” in a distributed-memory program may run on separate CPUs with separate operating systems, and it might not be possible to start a single ”distributed” process and have that process fork one or more threads on each system node due to a lack of software infrastructure.

2.3 Amdahl's Law

Gene Amdahl (1967) presented a more comprehensive model for the speed-up factor, which has been referred to as Amdahl's law [21].We could assume that the the percentage is α of the part of code cannot be parallelized and the rest of the code that can be parallelized is $(1-\alpha)$.Other communication delays, such as memory contention, latencies, etc., are disregarded.Amdahl's law for speed-up states that

$$S_P = \frac{T_1}{[\alpha + (1-\alpha)/P]T_1} = \frac{1}{\alpha + \frac{1-\alpha}{P}} \tag{2.1}$$

besides an upper bound in the limit of $P \to \infty$ is $S_P \leq \frac{1}{\alpha}$ .This indicates that even if $\alpha = 1\%$ for $P = 100$ we will have a result of $S_{100} = 50$ Therefore, we operate at half the desired efficacy.as shown in Figure 2.6

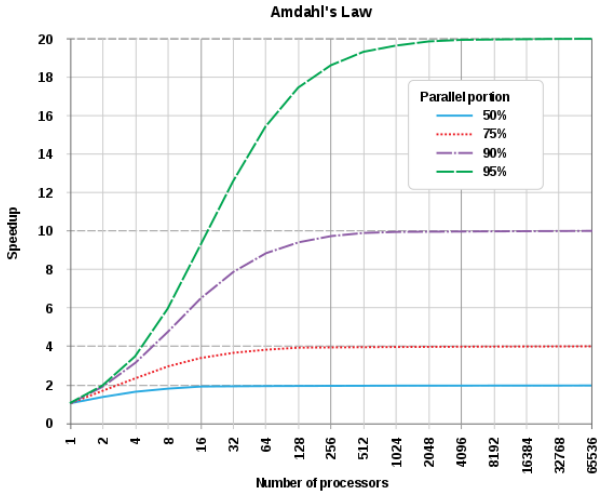


Figure 2.6 – Speed-up factor versus number of processors for three different degrees of parallelizable code

2.4 Scalability

If a technology can manage growing issue sizes, it is scalable. Nevertheless, scalability has a slightly more formal definition when it comes to discussions of parallel program performance. Let's say we execute a parallel program with a certain number of processes or threads and a set input size, and we achieve an efficiency of E. Let's say we now increase the amount of processes and threads the software is using. The program is scalable if we can determine a rate of growth in the issue size that corresponds to the program's efficiency E.

$$E = \frac{n}{p(\frac{n}{p+1})} = \frac{n}{n+p} \tag{2.2}$$

where $n = T_{serial}$ n is the problem size , T_{serial} is in micro-second(μ sec) , $T_{parallel} = \frac{n}{p+1}$ and $p \rightarrow$ the number of processes or threads. We want to determine the factor x by which we must raise the issue size such that E stays the same in order to determine if the program is scalable by increasing the number of processes/threads by a factor of k. The problem size will be xn and there will be kp processes or threads.

$$E = \frac{n}{n+p} = \frac{xn}{xn+kp} \tag{2.3}$$

if $x = k$, there will be a common factor of k in the denominator $xn + kp = kn + kp = k(n+p)$ so the answer will be as the first equation 2.2

However, our software is scalable if we raise the size of the problem at the same pace as we grow the number of processes or threads.

3 HYBRID MPI

3.1 Introduction to MPI

The Message Passing Interface (MPI) is a portable and standardized standard for message transmission in parallel computing architectures.

3.1.1 Goals

1. Develop a programming interface
2. Permit efficient communication by avoiding memory-to-memory duplication, allowing computation and communication to overlap, and offloading to communication co-processors when available.
3. Permit implementations that are compatible with a heterogeneous environment
4. Allow convenient C and Fortran interface bindings.
5. Assume a dependable interface for communication; the user need not deal with communication failures. The underlying communication subsystem handles these failures.
6. The interface is supposed to be designed to ensure thread safety.

Data Types

3.1.2 MPI Operations

A series of actions carried out by the MPI library to set up and allow data transmission and/or synchronization is known as an MPI operation. Initialization, Starting, Completion, and Freeing are its four steps. It is implemented as a collection of one or more MPI operations.

– **Initialization** gives the operation the argument list, but not the data buffers' contents, if any. Array parameters may not be altered until the operation is freed, according to the operation's specification.
– **Starting** gives the related operation control of the data buffers, when needed.
– **Completion** Returns control of the data buffers' contents and shows that, if any, the output buffers and parameters have been modified
– **Freeing** returns control of the rest of the argument list

Table 3.1 – Production cost

MPI datatype	**C datatype**
MPI_CHAR	**signed char**
MPI_CHAR	**signed char**
MPI_SHORT	**signed short int**
MPI_INT	**signed int**
MPI_Long	**signed long int**
MPI_Long_Long	**signed long long int**
MPI_UNSIGNED_CHAR	**unsigned char**
MPI_UNSIGNED_SHORT	**unsigned short int**
MPI_UNSIGNED	**unsigned int**
MPI_UNSIGNED_LONG	**unsigned long int**
MPI_FLOAT	**float**
MPI_DOUBLE	**double**
MPI_LONG_DOUBLE	**long double**
MPI_BYTE	
MPI_PACKED	

A highly strong and flexible API for creating parallel programs is message-passing. Almost every application that runs on the most powerful computers in the world makes use of message-passing. It is, however, also at a very low level. That is to say, the programmer must handle a tremendous quantity of detail. For instance, it is often essential to completely rebuild a serial program in order to parallelize it. The program's data structures may need to be explicitly spread throughout the processes or copied by each process. Additionally, it is typically impossible to rewrite something incrementally. For instance, it would be prohibitively costly to distribute a data structure for the parallel portions of the program and collect it for the serial (non-parallelized) portions. As a result, messagepassing is been referred to as "the assembly language of parallel programming," and several efforts have been made to create different distributed-memory APIs [22].

3.1.3 RMA

Access to memory remotely. Remote Memory Access (RMA), commonly referred to as one-sided MPI, is one of the most promising parallel programming paradigms that has lately been presented. Informally, RMA provides one-sided communication techniques that let a process do a 'put' or 'get' operation—writing data to or retrieving it from another processing—without contacting the other unit. RMA communications have been

proved to be advantageous for many applications by substituting direct remote memory accesses for transmitting and receiving messages.

Diverse techniques are used when programming with remote memory access (RMA). The majority of distributed memory machines now support the Remote Direct Memory Access (RDMA) technology, which frees internode communication from the control of the CPU and operating system. Many interconnects, including Infiniband, IBM's PERCS, Cray's Gemini, Aries, and RoCE over Ethernet, support RDMA.

When compared to the message-passing model, the non-blocking put and get operations of the RMA (which store data to and load data from remote memory, respectively), have low latency. Additionally, standard loads and stores as well as conditional atomic operations (compare-and-swap and fetch-and-add) are available. Additionally, RMA has a flush operation that synchronizes memories and guarantees data consistency. Typically, each process in RMA programs exposes a portion of its local memory for remote access (in MPI, this portion of memory is referred to as a window). RMA calls may access these locations, which are shared by the processes. Additionally, RMA frequently provides epochs, which are code segments designed for RMA calls. The RMA is synchronized throughout these epochs.Epochs come in two varieties: access epochs and exposure epochs. Only RMA memory access actions to distant (target) processes' memory are permitted in an access epoch. Other remote (origin) processes may perform RMA operations on the local memory of the current process while it is in the exposure epoch. In MPI, there are two types of synchronization: passive and active. Only two processes—one in access epoch and the other in exposure epoch—can make RMA calls when active synchronization is in place. Passive synchronization, which only requires one process, is a practical synchronization approach for concurrent data structure construction. The goal is to provide one-sided access to the target processes. The origin process alone performs passive synchronization; destination processes take no action. Passive synchronization, which only involves one process at a time, is a practical synchronization method for concurrent data structure design.

Advantages

1. **Reduce synchronization.**

   There is some implicit synchronisation involved in two-sided communication. For instance, a receive operation cannot finish before the associated send has begun. This

implies that synchronisation is required for each and every data transmission. One synchronisation step may be used to accomplish many separate PUT and GET operations using one-sided communication.

2. **No delay in sending data.** PUTs and GETs are non-blocking operations. For instance, this implies that a distant process may go on with its work while another process sends it data rather than having to wait for it.
3. **Functionality and scalability** You can overcome several issues while employing one-sided communication and scalable programs.The amount of time needed for two-sided conversation is much more.
4. **Functionality and scalability.** You can overcome several issues while employing one-sided communication and scalable programmes.The amount of time needed for two-sided conversation is much more.

3.1.4 Difference between two-sided and one-sided communication

Senders and receivers participate in two-sided communication. Both procedures are engaged in mutual communication. A communication procedure must be called by each party; the sending process calls MPI_Send, and the receiving process calls MPI_Receive. Each process has both transmitter and receiver capabilities shown in Figure 3.1.

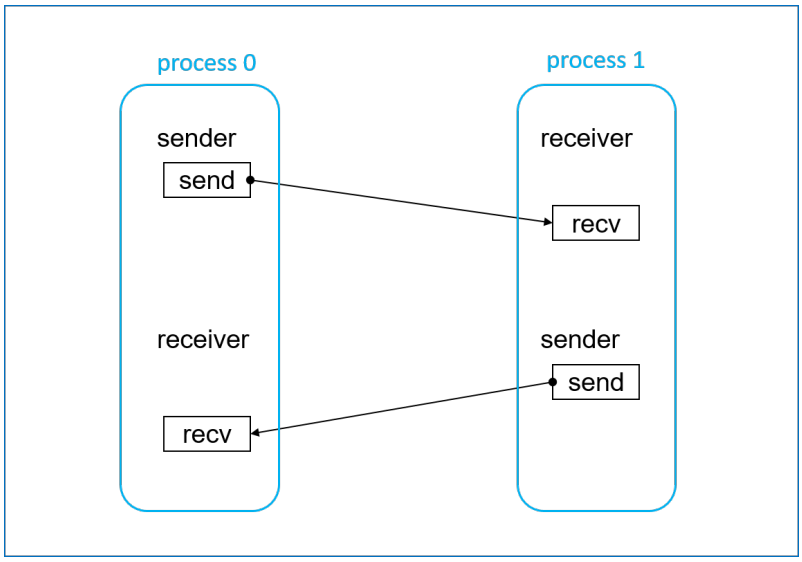


Figure 3.1 – Send and receive the data between process using P2P communication

Only one process—the so-called origin process—is active in one-sided communication. Process 0 is shown as the origin process in Figure 3.2. Process 0 uses MPI_Put to transfer data to process 1, the so-called target process. Process 1 gets the data without

using a receiving procedure. The execution of a put operation is therefore comparable to the execution of a send by the origin process and a corresponding receive by the destination process. One call made by the origin process that receives all parameters is all that is required. The MPI_Get function is called by the origin process to request data from the

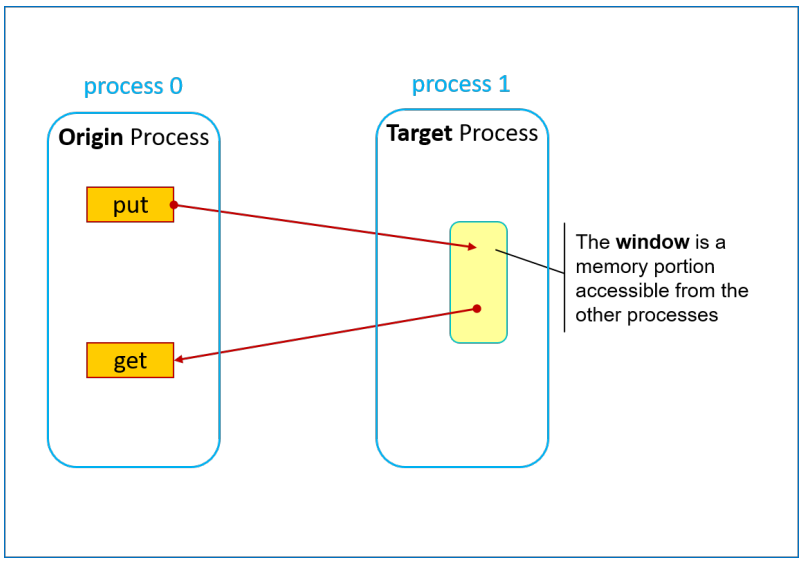


Figure 3.2 – this picture Put or Get data from the window

destination process. Similar to MPI_Put in operation, but with data transmission occurring in the other way.Therefore, MPI_Get is the same as the target process executing a send and the origin process responding with a corresponding receive.

Example of point to point communication

Here is an illustration Figure 3.3 of two-sided communication using MPI_Recv and normal MPI_Send. The data is read from the local send buffer by a process that calls MPI_Send. The data is subsequently placed in the local receive buffer by the matching MPI_Recv in the other process. The MPI library's responsibility is to transfer data from the sending process to the receiving process.

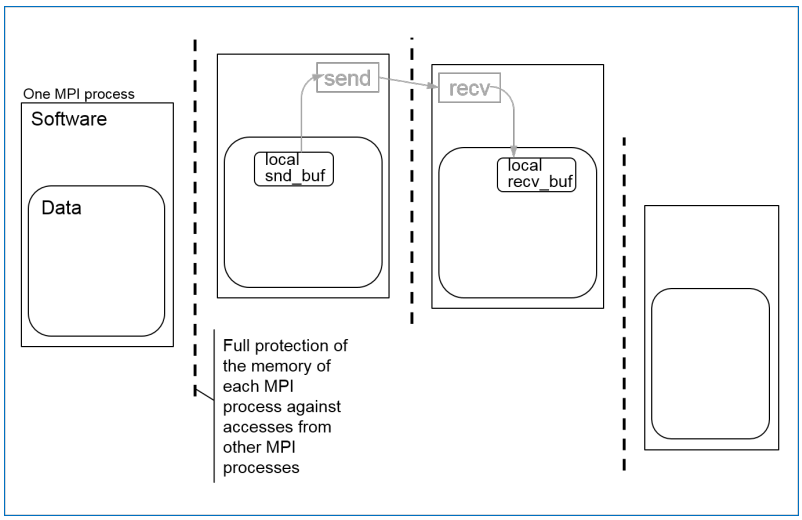


Figure 3.3 – two-sided using the local memory to send or receive data between processes

Example of one-sided communication

The data from a local transmit buffer may then be stored using MPI_Put in the window of a remote process in the following illustration Figure 3.4. Alternately, we can retrieve data from a remote window using MPI_Get and then store it in the local buffer. The only process that has to use the RMA procedures (MPI_Put and MPI_Get) is the origin side.

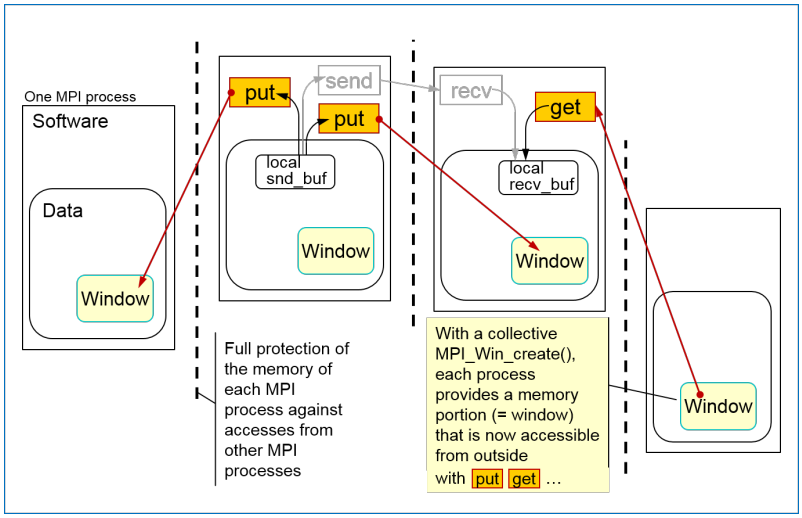


Figure 3.4 – one -sided put the data inside the window to make every process access it using the put or get protocols

3.1.5 Multi-threading in MPI

The MPI standard permits threads within processes in addition to the fact that an MPI application is made up of several processes. Different levels of multi-threading support, which allows threads to call MPI functions, are distinguished specifically:

– MPI_THREAD_SINGLE : no support is offered for multi-threading from an MPI
– MPI_THREAD_FUNNELED :In this instance, a limitation prevents anybody other than the thread that initiated MPI from calling MPI functions.
– MPI_THREAD_SERIALIZED :A single thread may only call an MPI function at a time.
– MPI_THREAD_MULTIPLE :There are no limitations in this situation; any thread inside a process is free to call MPI functions.

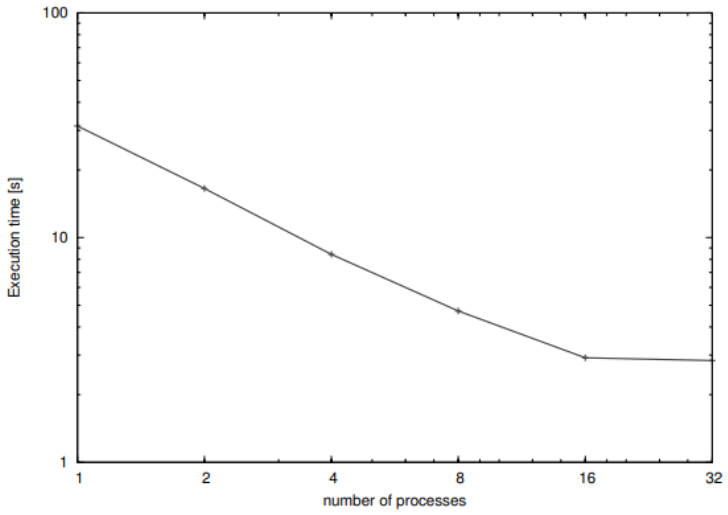


Figure 3.5 – Execution time of the tested MPI application run on a workstation with 2 x Intel Xeon E5-2620v4, 128 GB RAM, 16 physical cores, 32 logical processors, parameters: 300236771.4223 20000000000 [23]

3.2 Hybrid MPI programming

MPI is a distributed memory system, whereas OpenMP and Pthread are shared memory systems.

In other words, the data exchange between MPI requires a message because the MPI-synchronized programs belong to separate processes, or even different processes on different hosts. In contrast, because OpenMP and Pthread share memory, there is no need to transmit data between threads; simply transfer pointers.

3.2.1 OpenMP & PTHREADS

OpenMP is a platform-independent multi-threaded implementation. The primary thread (sequential execution of instructions) generates a number of sub-threads and divides tasks for execution between these sub-threads. These offspring threads execute concurrently, and the run time environment allocates them to separate processors.

The API for building multi-threaded programs is defined by POSIX threads or Pthreads, which in particular include constants, types, and supports:

– thread creation and waiting for other threads (joining)
– Mutexes that enable the implementation of thread execution that is mutually exclusive
– condition variables that are helpful when a thread might require waiting for another thread to satisfy certain requirements.

Utilizing mutexes, or mutual exclusion variables, the Pthreads API enables the implementation of critical sections that are carried out one thread at a time. Typically, a thread would read and alter certain variables inside such a code area. A mutex restricts access to this area; it is locked before it and unlocked after the execution of this code section. By definition, only one thread may lock a mutex at once; the other threads must wait for it to be freed. Initializing a mutex is possible using the Pthreads API.

3.2.2 MPI+OpenMP

MPI is used for communication across distributed memory nodes in the hybrid MPI+OpenMP paradigm, while OpenMP is responsible for on-node parallelization. Assuming there is a cluster of n shared memory nodes, each of which consists of m computational cores, each node has m computational cores. Figure 3.6 depicts the procedure for composing a composite MPI+OpenMP program (executed on n nodes) utilizing the OpenMP fine-grained parallelization technique[24, 25]. During initialization, n MPI processes are launched, and each one is assigned to a different node. The right portion demonstrates how each MPI process creates or releases memory that is located in its own address space and is thus inaccessible to other MPI processes. The calculation phase will use OpenMP directives, as seen in the left portion of the picture. Each process in this scenario creates m threads, each of which is doing the calculation simultaneously and is pinned to a core. In this case, the common MPI collective communication activities across the n MPI processes are directly used. The benefit of gradual parallelization offered by

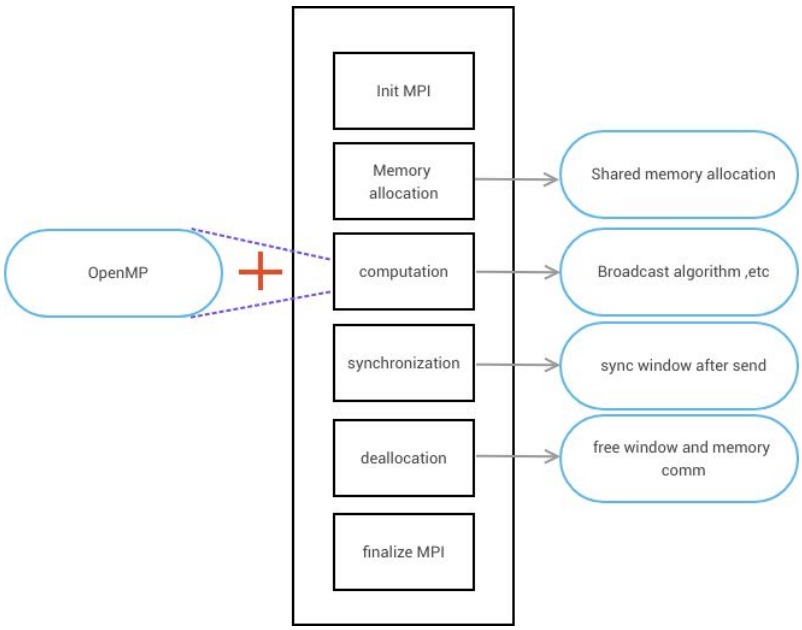


Figure 3.6 – The workflow of the hybrid MPI+OpenMP

OpenMP makes it easier to migrate MPI code to a hybrid MPI+OpenMP code. It is difficult and labor-intensive to achieve the same level of parallelism as in the MPI version in the hybrid MPI+OpenMP version, which will lessen the benefit of adopting OpenMP. The coarse-grained parallelism strategy is also investigated, although it is not as developed as the fine-grained approach. Thus, one of the benchmarks for assessing the hybrid MPI+MPI program is the fine-grained parallel hybrid MPI+OpenMP program [25].

3.3 Hybrid MPI + MPI-3 shared memory

3.3.1 Splitting into two levels of communicators

To split the communicator into distinct node-level communicators, the MPI_Comm_split type is called with the MPI COMM TYPE SHARED option. A collection of processes linked to the same shared memory system are identified by each node-level (also known as shared memory) communicator. Within this shared memory system, all processes are allowed to execute load and store operations rather than explicit remote memory access (RMA). The hybrid MPI+MPI programming paradigm also calls for an across-node communicator to provide explicit communication between processes running on other nodes in addition to the shared memory communicator. The other on-node processes are seen as its offspring, and one process per node (typically with the lowest rank) is designated as the leader to handle data exchanges among nodes. The

across-node communicator, which is created by executing MPI_Comm_split, sometimes goes by the name bridge communicator since it serves as a bridge across nodes[26].

3.3.2 MPI shared memory window

To construct a window that spans a section of accessible shared memory with a unique size that is provided by each on-node process, MPI_Win_allocate shared must be used. The memory in a window is typically allocated in hardware in a continuous fashion shown in Figure 3.7. When the parameter allocate shared noncontig is set to true, it is also possible to create non-contiguous memory. To get the base pointer to the shared memory segment given by another process, use the MPI_Win shared_query method. All on-node processes are able to access the allocated memory using instantaneous load and store instructions thanks to this base pointer. The MPI_Win_sync function is defined to synchronise between the private and public window copies, as would be expected. Nowadays, the vast majority of hardware designs include a unified memory model that allows for implicit consistency maintenance between the public and private copies. Even so, when many on-node processes are simultaneously accessing the same memory address, MPI_Win_sync may still be useful in establishing memory synchronisation [25].

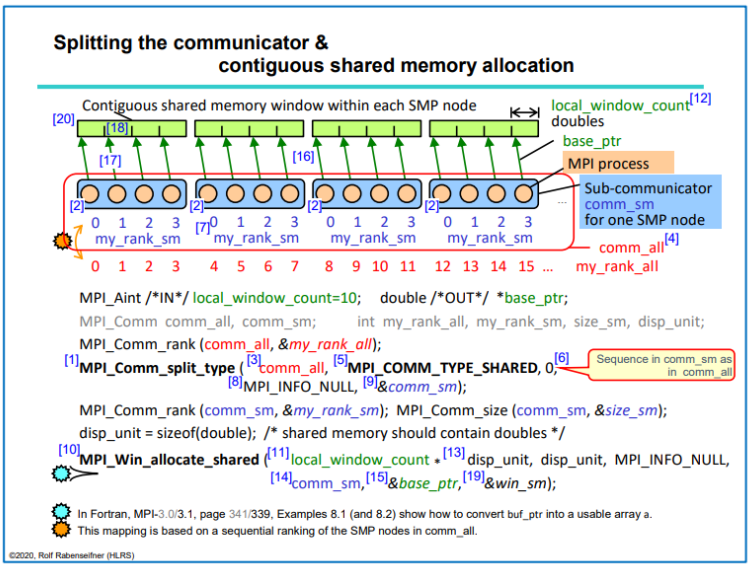


Figure 3.7 – shared-memory allocation.

3.4 Workflow

At the first step, we initialize our MPI program with multiple threads and our rank and number of processes with MPI_COMM_WORLD, then choose our root. At the second step, we allocate our shared memory with window for the maximum size of buffer type. At the third step, we compute our broadcast to send the data across the nodes from the root to other processes. At the fourth step, we synchronize our window after putting data into the destination rank, By performing load/store instructions with all the necessary node-level synchronizations to ensure its constant state, the shared area may be accessed. At the Fifth Step, we deallocate our memory by freeing our window and our shared memory. Finally, we finalize our program as it is shown in Figure 3.8.

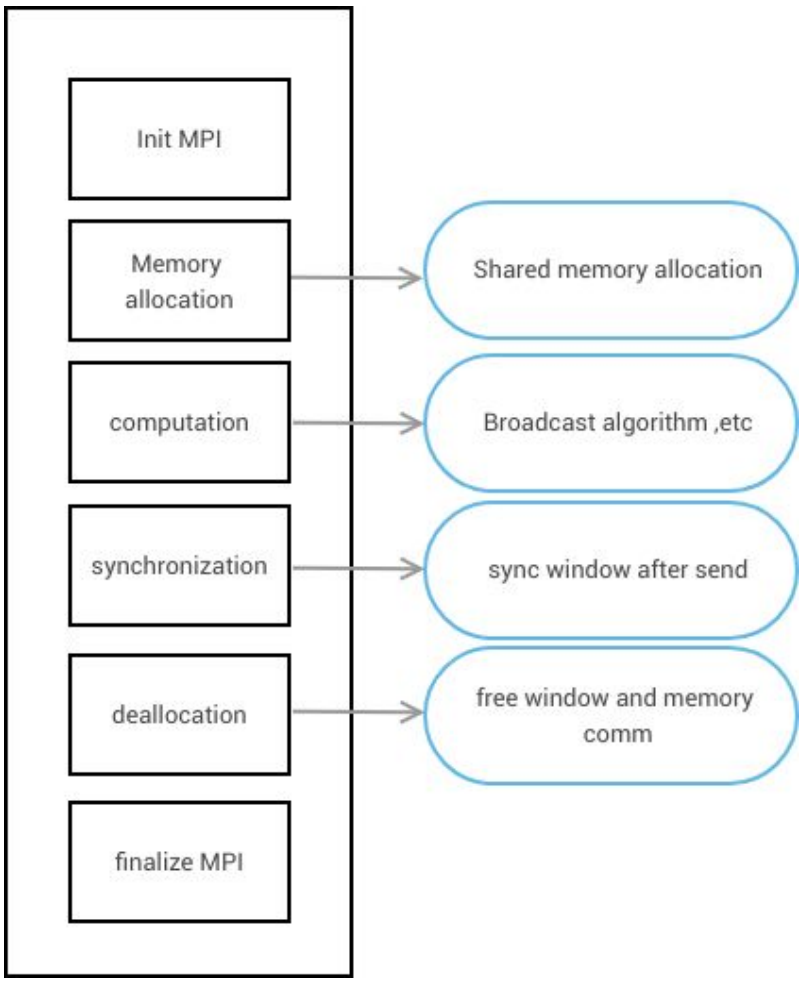


Figure 3.8 – Example of workflow

3.4.1 Advantages

– No message passing inside of the SMP nodes

– Using only one parallel programming standard
– No OpenMP problems then the thread safety isn’t a problem

3.4.2 DisAdvantages

– Communicator must be split into shared memory islands
– To minimize shared memory communication overhead: Halos (or the data accessed by the neighbors) must be stored in MPI shared memory windows
– Same work-sharing as with pure MPI

4 PROGRAMMING PARALLEL PARADIGM

4.1 Master slave

The following acts may be carried out differently depending on how the master-slave paradigm is implemented:

– executed dynamically by a master or statically before computing
– combining results from slaves, either dynamically when each result is received by the master or after all results are collected.

By giving slaves the authority to collect subsequent data packets, calculate, store results, and continue this process, the code may also be made simpler while eliminating the active role of the master. In this situation, a synchronisation method between slaves must be used [23].

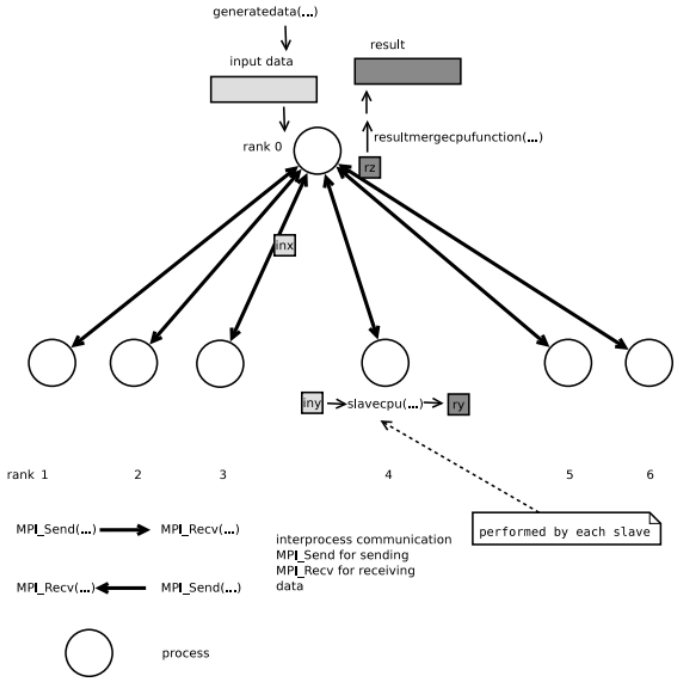


Figure 4.1 – Master slave strategy with MPI

In the Figure 4.1 illustrate :

1. creates packets of input data
2. waits for any data packet's processing to be finished, combines the results, and then replies with a new packet if one is still available.
3. Sends a notification of elimination to all slaves

4.2 Divide and Conquer

4.2.1 Balanced version

A balanced implementation that divides an input problem (and the data it generates) into issues of computationally equal size. This enables for effective computing and communication structure and is continued recursively up until a certain depth of the divide-and-conquer tree is achieved. Additionally, it calls for as many processes to create as many leaves. A variety of algorithms might benefit from such a strategy [23].

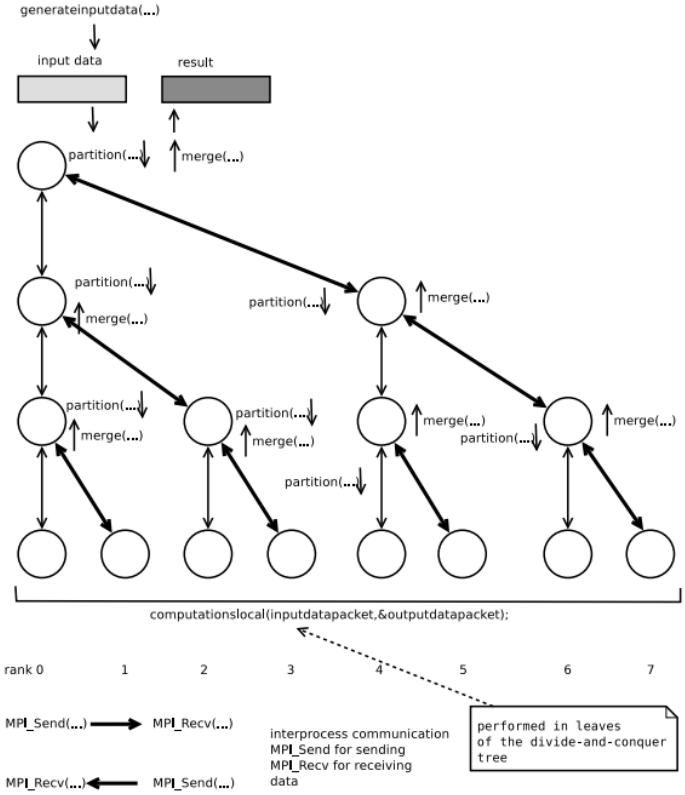


Figure 4.2 – Balanced divide and conquer with MPI

The Figure 4.2 shows:

1. In a binary tree structure, processes are organised such that they may interact with one another and pass on sub-problems to one another throughout each iteration.
2. This keeps happening until all processes have their components.
3. by calling $computationslocal()$ in parallel, processes process the data related to their sub-problems. The next step is parallel integration of the results from each process, which is carried out in the opposite direction of partitioning in a structure that resembles a tree and allows pairs of processes to interact throughout each iteration.

4.3 Pipeline

The input data is seen as a stream of input data packets that enter a pipeline in pipeline type processing, where it is possible to discern the separate steps of processing through which each input data packet has to pass. Each step of the pipeline is linked to a piece of code that may run on a different core, processor, or accelerator. To be more specific, it may be preferable for certain programs to be run on a computer that is particularly well-suited to run them. This might serve as justification for choosing this method of processing as illustrate in Figure 4.3. However, if some pipeline stages process data packets much more slowly than the others, such pipeline processing may experience performance bottlenecks [23].

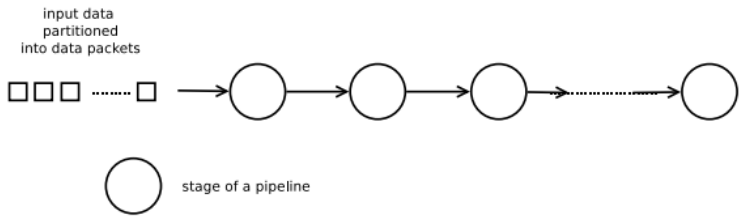


Figure 4.3 – pipeline processing

4.4 Sequential Programming Paradigm

One of the first paradigms for programming is sequential programming. It closely resembles machine architecture. On Von Neumann architecture is where it is based. By using assignment statements, it alters the program's state. By altering states, it completes tasks step by step. The method of achieving the goal is the major concern. The paradigm is made up of a number of statements, and once each one is executed, a result is saved.

4.4.1 Advantage

– relatively simple to Implement.

4.4.2 Disadvantage

– Complex issues are unsolvable.
– Less efficient and less productive.

– Programming in parallel is not achievable.

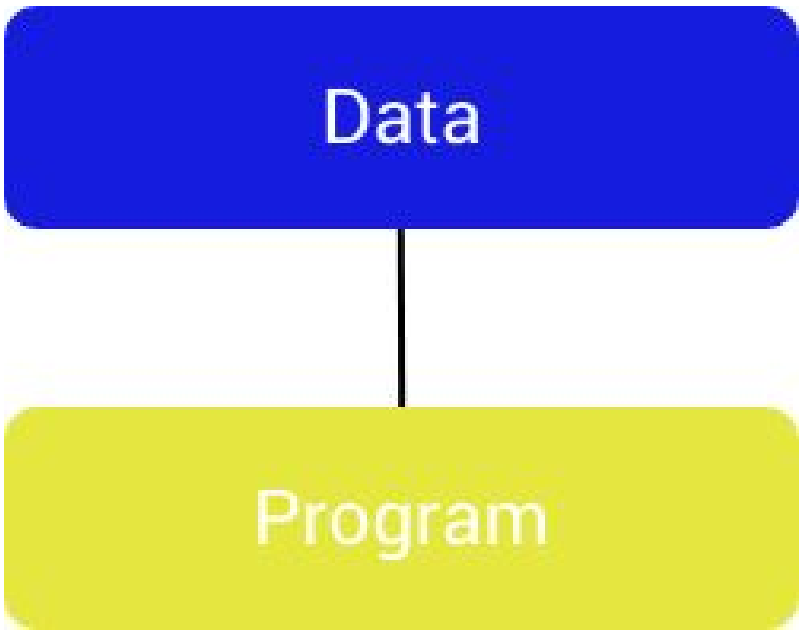


Figure 4.4 – Sequential Programming

Table 4.1 – Difference between Sequential and parallel computing

Sequential Computing	**Parallel Computing**
One by one, each instruction is carried out in order	Every command is carried out concurrently.
It has a single processor	It is having multiple processors
Due to the single processor, it performs poorly and puts a heavy burden on the processor.	Because several processors are operating at once, it has high performance and low processor workload.
The format utilised for data transport is bit-by-bit.	Data transfers are in bytes.
The whole procedure takes longer to finish.	The whole procedure takes less time to finish.
Cost is low.	Cost is high.

4.5 Message-Passing Programming Paradigm

On the other hand, under the message-passing paradigm, processes interact by messages; each process has its own address space, and no data is transferred between processes. Every piece of information must be statically linked to a certain process and only that process has access to it. As a result, all references to data are to local memory [27].

A system implementing the message-passing paradigm is seen logically as having p processes, each with a unique address space. Data must be properly partitioned and put since each data piece must correspond to one of the space's divisions. Every interaction, whether read-only or read/write, necessitates two processes working together: the process that has the data and the process that wants to access it. Message-passing programs are often written using the asynchronous or loosely synchronous paradigms. All concurrent processes run asynchronously under the asynchronous paradigm. Tasks or subsets of tasks synchronize to carry out interactions in the synchronous model. Tasks are entirely asynchronously executed in between these encounters shows at Figure 4.5.

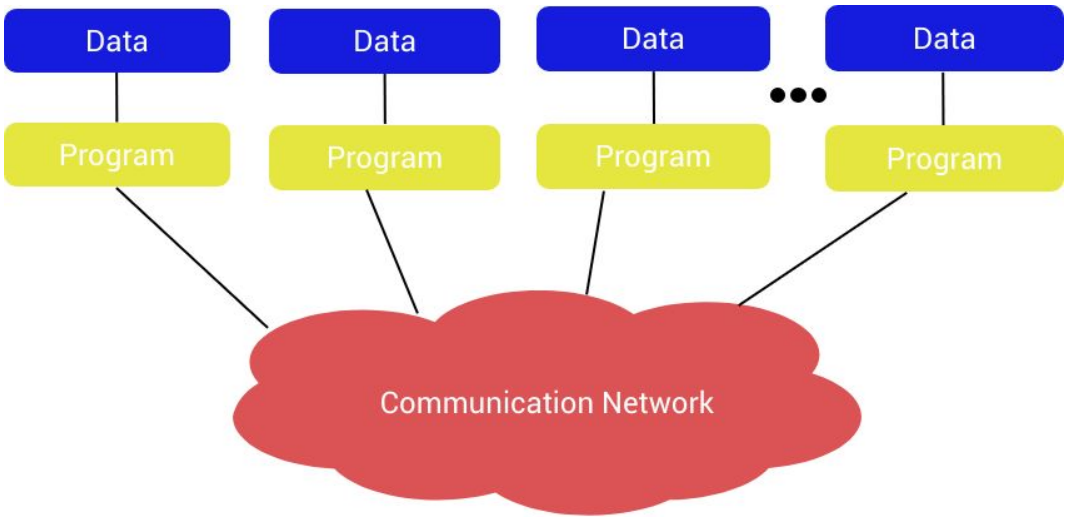


Figure 4.5 – Message-Passing Programming Paradigm

These processes interact with one another by delivering messages. These communications consist of data packets that are placed within envelopes with routing information. We may transfer data across processes by using the message passing mechanism the Illustration at Figure 4.6.

4.6 Shared Memory Model

The operating system's technique for allowing processes to interact with one another is called process communication. A process may inform another process of an event that has happened or transmit data from one process to another as part of this communication. The shared memory model is one of the models used to describe process communication. Threads interact through references to shared data in the shared-memory architecture. The shared data theoretically exists in a global memory that is accessible by

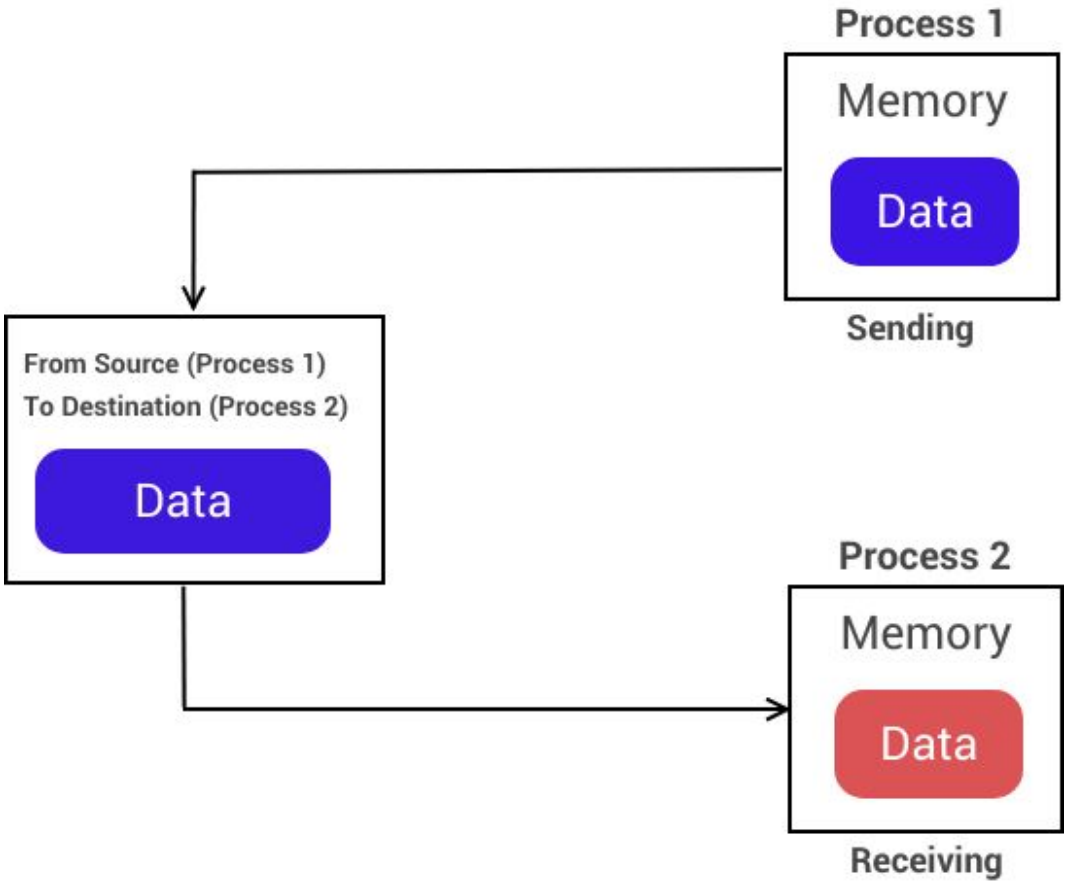


Figure 4.6 – Message communication

all threads as shown in Figure 4.7. Since only a portion of the shared data will be stored in local memory, certain data references will actually cost more than others. Typically, a program's global memory is implemented using a single address space that is shared by all of the threads [27].

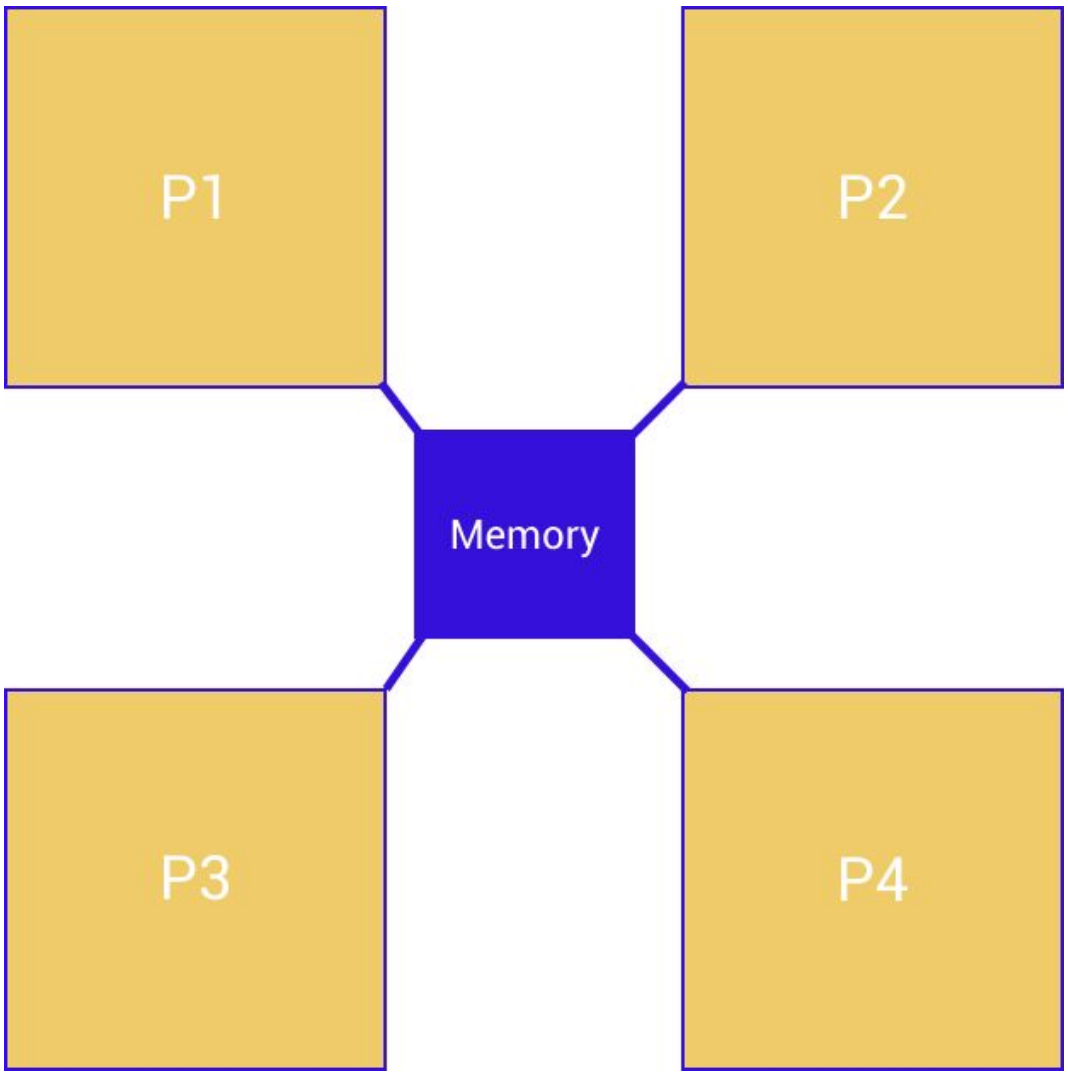


Figure 4.7 – Shared memory model

4.6.1 Advantage

– Since all communication is done via shared memory and just the shared memory needs to be set up once, it offers the fastest possible calculation performance.
– No kernel intervention.

4.6.2 Disadvantage

– The shared memory approach requires all processes to take care to avoid writing to the same memory region.
– Shared memory models might lead to issues that need to be resolved, such as synchronization and memory protection.

5 COLLECTIVE OPERATIONS

In Open MPI, A component offers functionality with a particular set of implementation aspects. For each collective operation, defined in MPI as a series of point-to-point transmissions between the relevant processes, a collective component called Tuned, for example, implements different algorithms. A communicator gives the collection of processes carrying out the collective activity an isolated communication context. Procedures in a communicator include a root integer number starting at 0 [28]. We implemented algorithms using a Binomial tree with one-sided remote memory access and shared memory access. When one MPI process, known as root, broadcasts the identical message to all other processes, it is known as a broadcast operation. The broadcast data is allotted a space in memory in each leader that may be shared by its offspring in our approach broadcast strategy. According to the MPI broadcast semantics, only the root is allowed to change the broadcast data. A local pointer to the start of this shared memory area allows any process on the same node to independently access the broadcast data. Since the size of the broadcast message is the same as that of the pure MPI broadcast, conducting the across-node broadcast operation (across all the leaders) is simple in this case [25].

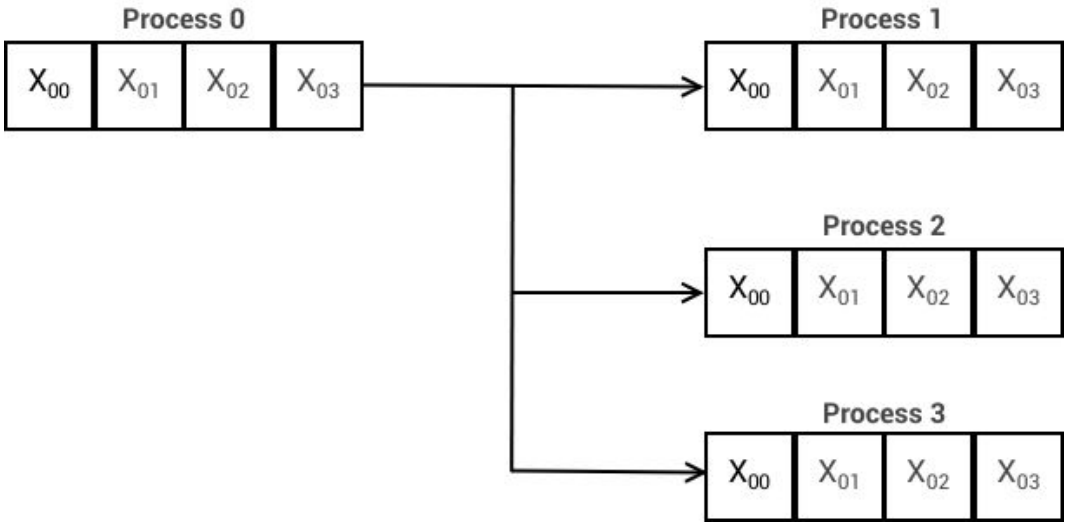


Figure 5.1 – Example of Broadcast for sending data from root to the processes number of processes 4

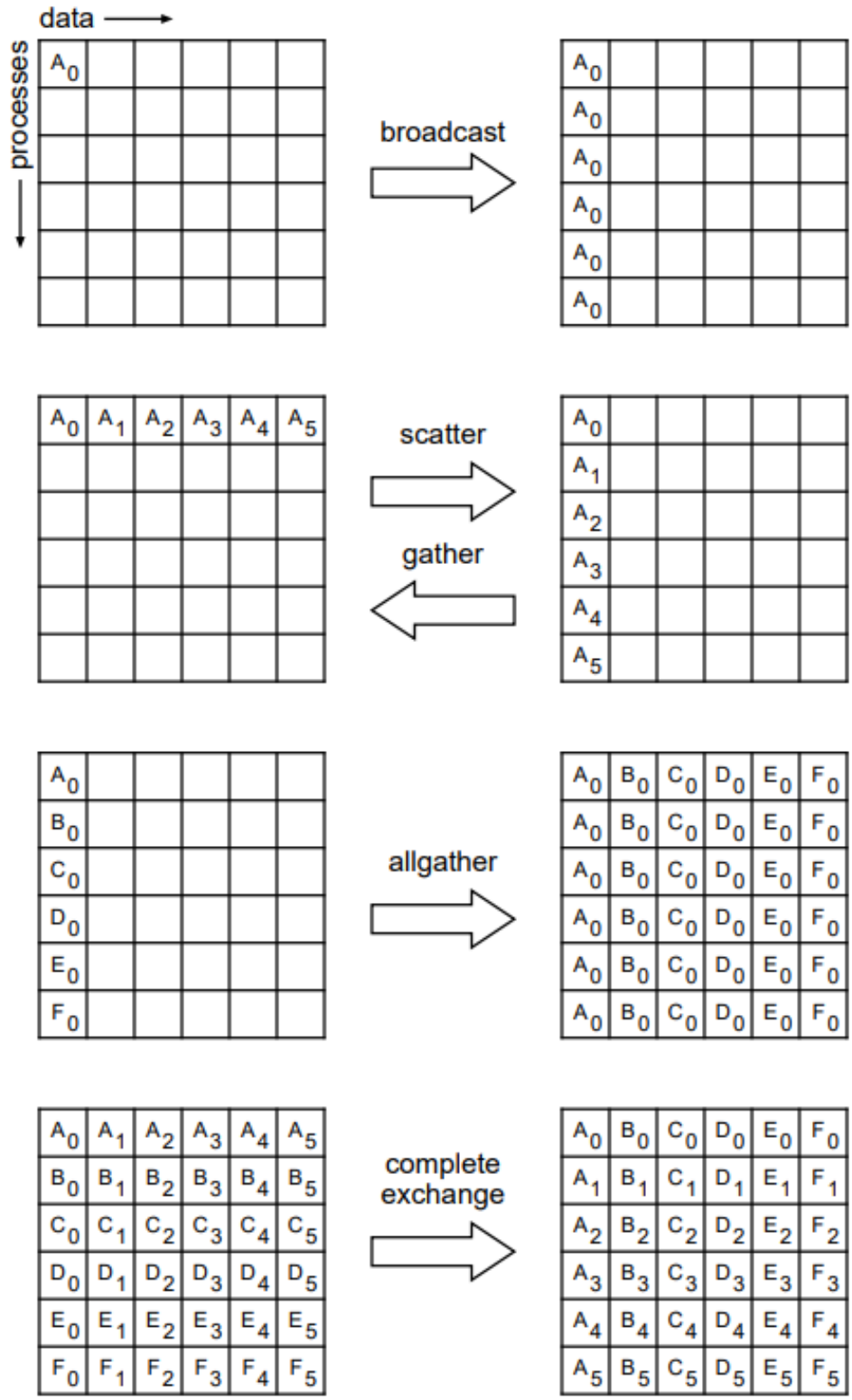


Figure 5.2 – Illustration of the collective move functions for a set of six processes. Each row of boxes in each scenario corresponds to a particular process's data locations. As a result, during the broadcast, the data A_0 is only present in the first process at first, but after the broadcast, it is present in all processes.

MPI_GATHER implements an all-to-one gather operation:All processes transfer data from inbufffer to root, including the root process. Data from process i is placed before data from process $i+1$ in continuous, nonoverlapping areas in the outbuffer throughout this process. Because there are P processes involved in the operation, the outbuffer in the root process must be P times larger than the inbuffer. Other than the root process, the outbuffer is not taken into account.

MPI_SCATTER implements a one-to-all scatter operation:It is MPI_GATHER's opposite. Each process gets data from root in outbuffer, while a defined root process provides data to all processes by sending the i th part of its inbufffer to process i. Therefore, the root process's inbufffer and outbuffer must be P times larger than one another. The slight distinction between this function and MPI_BCAST is as follows: Every process gets the same value from the root process in MPI_BCAST, but in MPI_SCATTER, each process receives a unique value.

5.0.1 Binomial Tree

At the first round on Figure 5.3 P_0 sends message to P_1 and at the second round P_0 send to P_1 and P_2 .Finally, at the third round P_0 sends messages to P_1 , P_2 and P_4 in addition to that P_1 sends messages to P_3 and P_5 also P_2 sends message to P_6 and finally P_6 send message to P_7 [9, 29]. With each round being shown a doubling of the number of nodes broadcasting and receiving. As a result, there are fewer transmission stages from $P-1$ to $log_2(P)$ and can be calculated with equation 5.1.

$$T_{Total} = T * log_2(P) \tag{5.1}$$

There are two fundamental drawbacks of broadcasting down a binomial tree. First, when the communicator size is not a correct power of two, the communication time is obviously out of balance [29].To finish the task, the final step (for instance, rank 7 in Figure 5.3) requires $log_2(P)$ communication rounds.

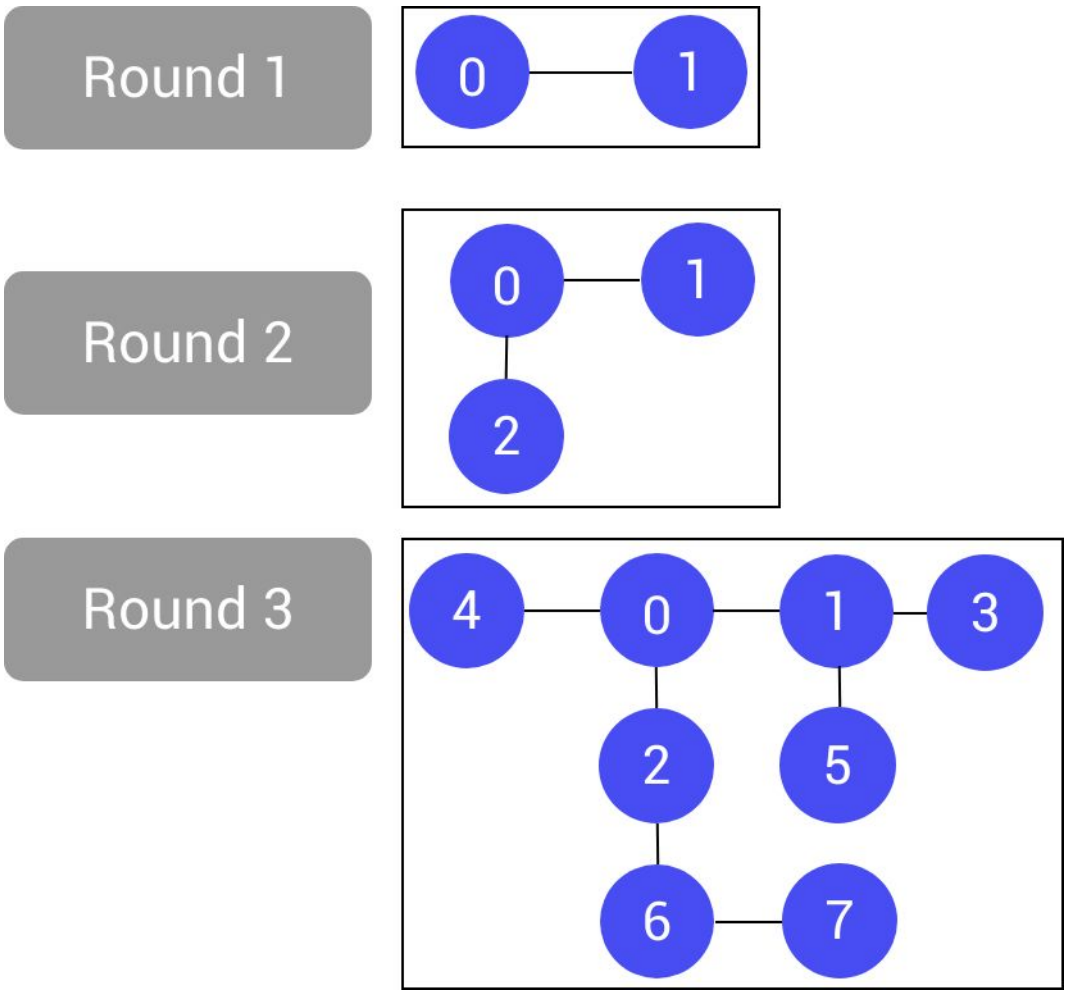


Figure 5.3 – Binomial tree

In RMABcastBinomial perform sending data at Algorithm 2:

1. Compute shifted rank $srank$ Algorithm 4.this rank is related to root (Line 1).
2. The process start loop with $log_2(P) from (3 - 16)$
3. Begin synchronization epoch for each process
4. if $srank$ in Line (2) is not set compute $rank$ Line (7) then assign the snd_buf to rcv_buf with the rank Line (9).otherwise $break$ the loop Line (13).
5. synchronizes the private and public window copies of win line (10)

Algorithm 1 RMABcastBinomial(snd_buf , rcv_buf , win , $comm$)

Require:

snd_buf- origin buffer

rcv_buf- received buffer

$win-$ RMA window

$comm-$ logical group of MPI processes

Ensure:

for $r = 0$ **to** p **do**

$descr.root = r$

$send_loop(snd\_buf, rcv\_buf, rank, descr, win, comm)$

end for

Figure 5.4 – Algorithm of Binomial Tree for broadcast messages across the communication processes

Algorithm 3 move_data(snd_buf , rcv_buf , put_rank , win)

Require:

snd_buf- origin buffer

rcv_buf- received buffer

$win-$ RMA window

put_rank- the destination rank

Ensure:

$MPI\_Win\_lock(MPI\_LOCK\_SHARED, win)$

for $i = 0$ **to** $sizeof(snd\_buf)$ **do**

$rcv\_buf[i + rank] = snd\_buf[i]$

end for

$MPI\_Win\_sync(win)$

$MPI\_Win\_unlock(rank, win)$

Figure 5.6 – Algorithm shows how to move data from the current location of memory to the destination processes location

Algorithm 2 send_loop(snd_buf, rcv_buf, win, $comm$)

Require:
snd_buf- origin buffer
rcv_buf- received buffer
$win-$ RMA window
$comm-$ logical group of MPI processes
Ensure:
$srank = comp\_srank(my_rank, descr.root, p)$
$mask = 1$;
while $mask < p$ **do**
  if $(srank \& mask) == 0$ **then**
    $rank = srank | mask$
    if $rank < p$ **then**
      $rank = comp\_rank(rank, root, p)$
      $check\_received\_data$
      $move\_data(snd\_buf, rcv\_buf, rank, win)$
    else
      $break$
    end if
    $mask = mask << 1$
  end if
end while

Figure 5.5 – Algorithm shows hot to send data across the processes on RMA one sided shared memory model

Algorithm 4 comp_srank(my_rank, $root$, p)

Require:
my_rank- current process's rank
$root-$ root process
$p-$ number of MPI processes
Ensure:
$(my\_rank - root + p)\%p$

Figure 5.7 – Algorithm illustrate how to calculate Shifted rank

In Comp_srank :

1. Compute shifted rank $srank$ A.this rank is related to root (Line 1).

Algorithm 5 comp_rank(my_rank, $root$, p)

Require:

$srank-$ the computed rank in 4

$root-$ root process

$p-$ number of MPI processes

Ensure:

$((srank + root)\%p$

Figure 5.8 – Algorithm illustrate how to calculate put rank

5.0.2 Binary Tree

At the first round in Figure 5.9 P_0 sends a message to P_1 and P_2, at the second round P_1 sends to P_3 and P_4, for the third round P_2 sends messages to P_5 and P_6 .Finally, at the Fourth round, P_3 sends messages to P_7 [28]. The height of the binary tree is equal to 5.2

$$T_{Total} = T * log_2(P) \tag{5.2}$$

at each round the maximum number is 2^i for i is the round number. With each round being shown a doubling of the number of nodes broadcasting and receiving. As a result, there are fewer transmission stages [29].To finish the task, the final step (for instance, rank 7 in Figure 5.9) requires $log_2(P)$ communication rounds. In RMABcastBinary perform moving data at Algorithm 6:

1. Compute shifted rank $srank$ Algorithm 4.this rank is related to root (Line 1).
2. The process start loop with $log_2(P) from(3 - 19)$
3. compute the $rank$ according to the $root$ and $child1$ Line (4)
4. calculate $child1$ and $child2$
5. if $child1$ is less than number of processes
6. Begin synchronization epoch for each process Line (8) in the $move_data$ algorithm 3
7. compute $child1$ Line (7) then assign the snd_buf to rcv_buf with the rank Line (9).
8. otherwise repeat same steps for $child2$

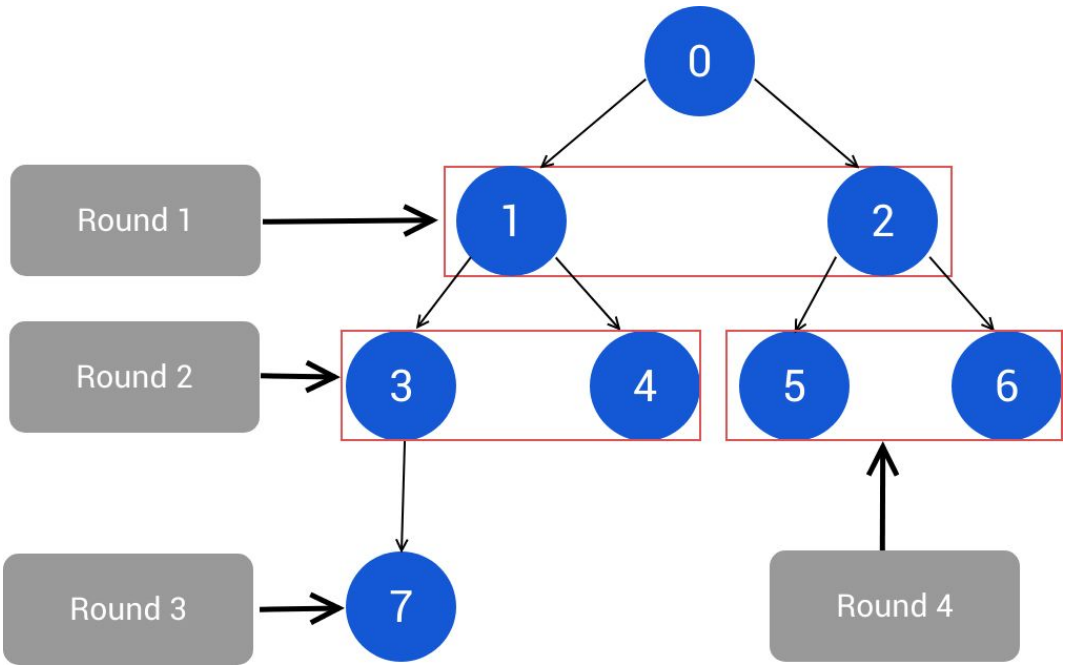


Figure 5.9 – Binary tree

Algorithm 6 RMABcastBinary(snd_buf , rcv_buf , win , $comm$)

Require:

snd_buf – origin buffer
rcv_buf – received buffer
win – RMA window
$comm$ – logical group of MPI processes

Ensure:

$srank = comp\_srank(my_rank, descr.root, p)$
for $r = 0$ **to** p **do**
$rank = comp\_rank(rank, root, p)$
$child1 = 2 * rank + 1$
$child2 = 2 * rank + 2$;
if $child1 < p$ **then**
$child1 = comp\_rank(rank, root, p)$
$move\_data(snd\_buf, rcv\_buf, child1, win)$
end if
if $child2 < p$ **then**
$child2 = comp\_rank(rank, root, p)$
$move\_data(snd\_buf, rcv\_buf, child2, win)$
end if
end for

Figure 5.10 – Algorithm of Binary Tree for broadcast messages across the communication processes

5.0.3 Linear sequential Algorithm

the message transfer from the root to all other ranks they send each process after the last process is finished P_0 send to $P_1, P_2, P_3 ... P_7$ in the case of with size of 8 the 5.3 the number of transferring time [28, 30].

$$T_{Total} = T * (P - 1) \tag{5.3}$$

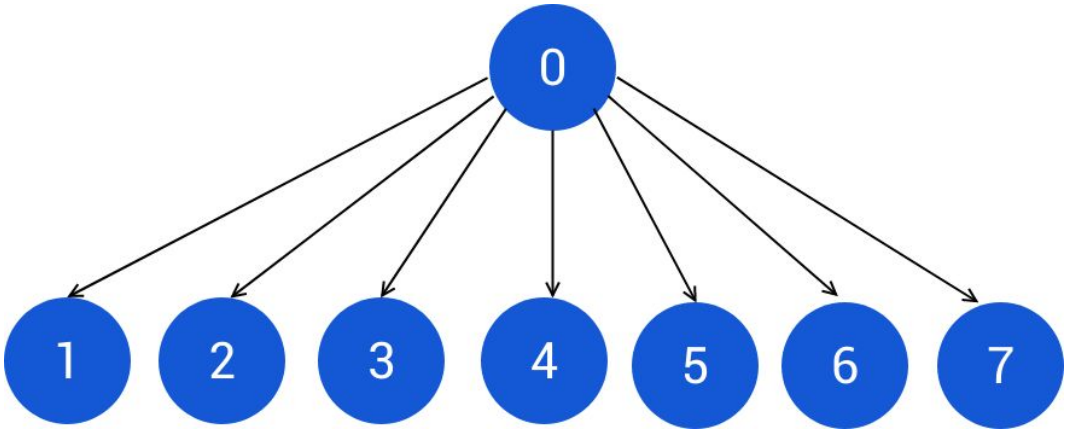


Figure 5.11 – Linear sequential tree

Here we just start passive target synchronization epoch for all processes and assign the data to the all processes' locations, corresponding the window win .then we synchronize our window win Algorithm 7.

Algorithm 7 RMABcastLinear(snd_buf, rcv_buf, win, $comm$)

Require:

snd_buf- origin buffer
$length-$ message length
$win-$ RMA window
$comm-$ logical group of MPI processes

Ensure:

for $r = 0$ **to** p **do**
$MPI\_Win\_lock(MPI\_LOCK\_SHARED, win)$
$MPI\_Put(snd\_buf, length, datatype, rank, win)$;
$MPI\_Win\_sync(win)$
$MPI\_Win\_unlock(rank, win)$
end for

Figure 5.12 – Algorithm of Linear sequential Algorithm for broadcast messages across the communication processes

6 THE RESULTS OF THE RESEARCH (DEVELOPMENT)

6.1 Methodology of the experiment

The experiment with shared Broadcast have been carried out on HP XL250a Gen9 equipped with Two 12-core Intel Xeon E5-2680v3 processors 2500 MHz,192 GB RAM We have tested with 1000 messages and 8 packages starting from 16,128,..,33554432 Bytes with Open MPI 4.0.1. As a comparison, shared memory was set up for MPI.we have used Shared memory to implement collective algorithm (Broadcast)

6.2 Results of the experiment

6.2.1 Open MPI

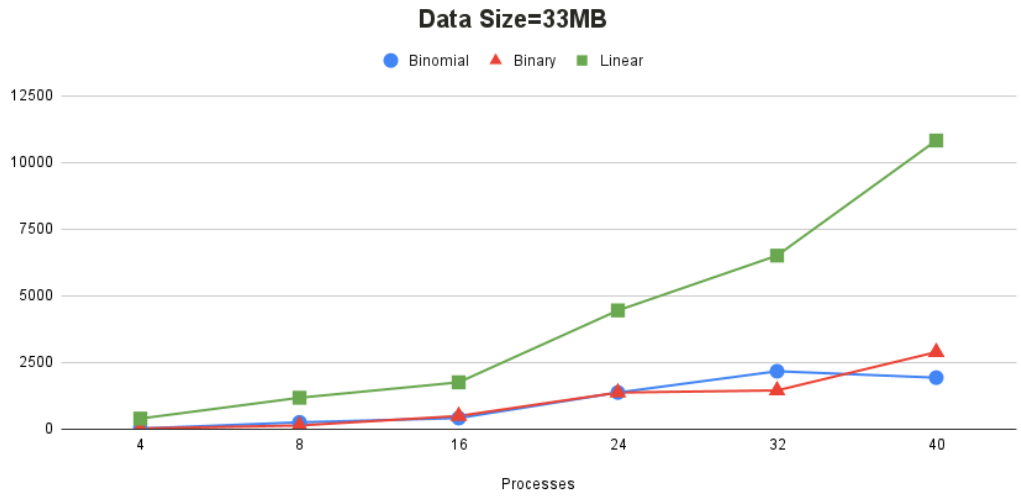


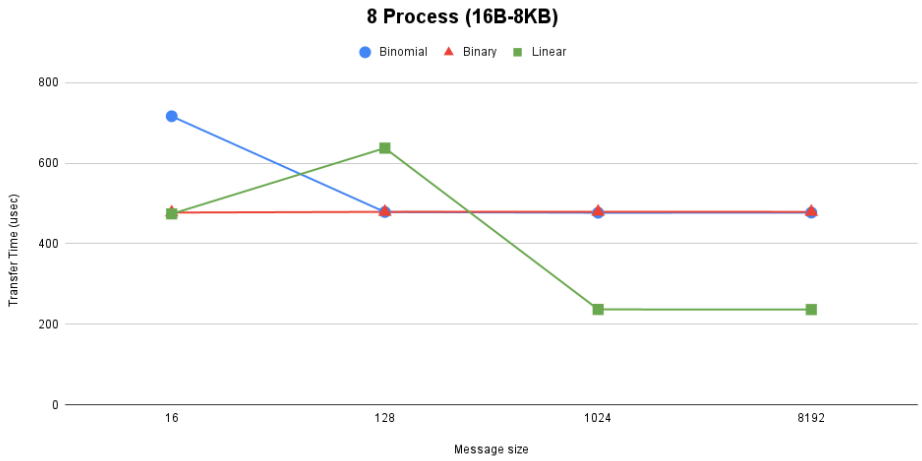
Figure 6.1 – Performance of Binomial Tree, Linear, and Binary Tree for Data size 33MB with different number of processes

Figure 6.1 demonstrates an interesting pattern when it comes to the transmission of large messages exceeding 33MB. Within the context of the Shared Memory model, the Binomial tree algorithm exhibits superior performance compared to the other algorithms. The Binary tree closely follows this, exhibiting a transfer time that's almost on par with the Binomial tree, highlighting its potential efficiency for large data transfers.

However, the situation is distinctly different for the Linear algorithm. For large data message transfers, it is found to be the slowest among the three algorithms. Its per-

formance lags notably behind both the Binary and Binomial trees, indicating that it may not be the most efficient choice for transferring larger volumes of data.

(a) – Small Data



(b) – Medium Data

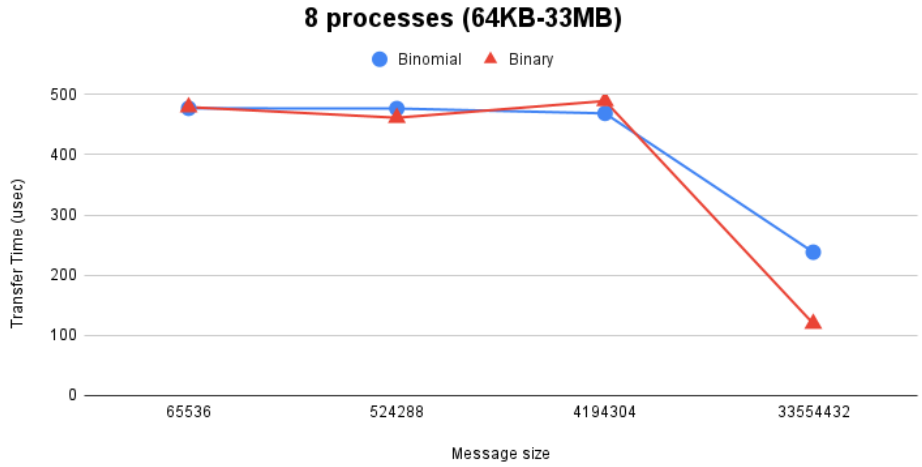


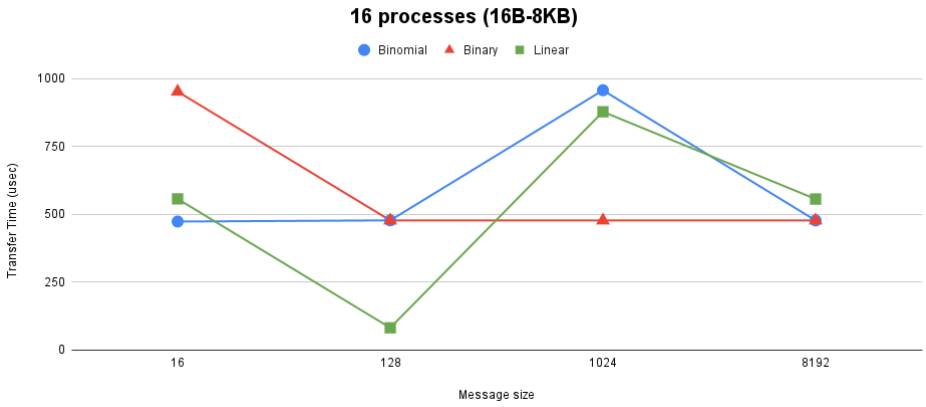
Figure 6.2 – The broadcast operation's latency varies depending on the number of processes 8 and the size of the buffers.

The Linear method turns out to be the most effective for the job of transmitting data with a size range from 16B to 8KB, as illustrated in Figure 6.2a. When compared to its competitors, it performs better when it comes to data transmission.

The circumstance changes, as shown in Figure 6.2b, nevertheless. When compared to the Linear method, the Binary and Binomial trees both show a significant decrease in transfer time. This demonstrates their ability to better manage data flow in this environment.

However, the Binary tree method excels at handling even higher message sizes. It is advised for sending data weighing in at 65KB to 33MB. This means that it has the special capacity to manage large amounts of data more effectively.

(a) – Small Data



(b) – Medium Data

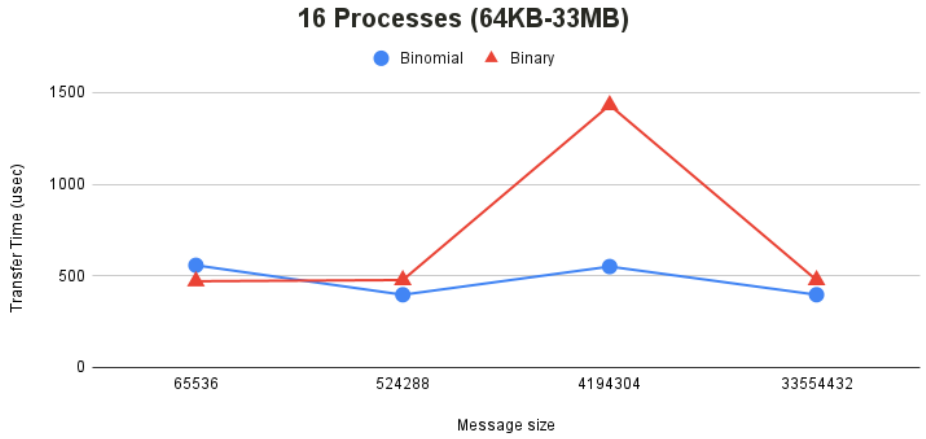
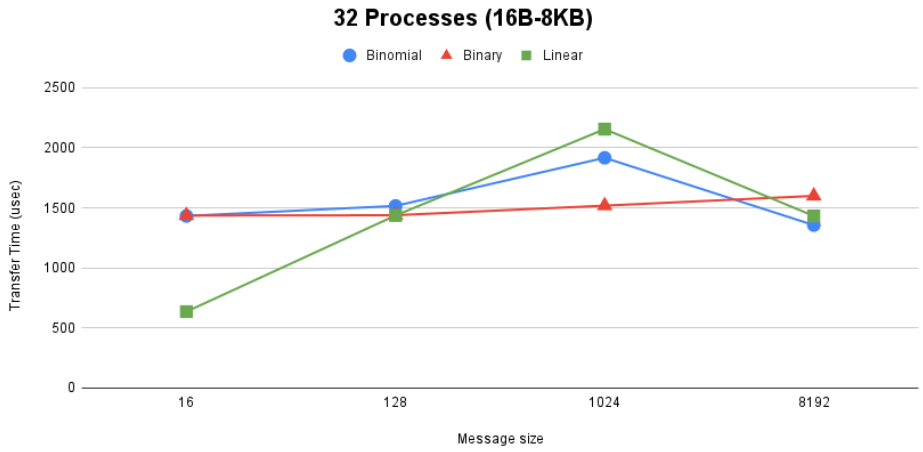


Figure 6.3 – The broadcast operation's latency varies depending on the number of processes 16 and the size of the buffers

The Binary tree demonstrated outstanding effectiveness in lowering transfer time during the transmission of messages using 16 processes. This was especially clear for smaller data sets, as Figure 6.3a illustrates. This performance puts the Binary tree ahead of the Binomial and Linear trees, giving it the top recommendation for applications involving smaller amounts of data.

On the other hand, the Binomial tree becomes more prominent as the amount of the data grows. When working with bigger data sets, it performs better than the Binary tree, as seen in Figure 6.3b. This shows that the Binomial tree is the best choice in scenarios requiring large amounts of data.

(a) – Small Data



(b) – Medium Data

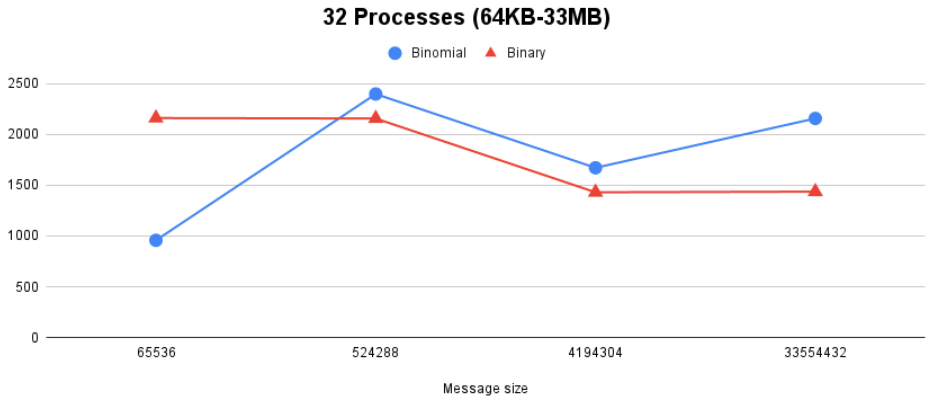
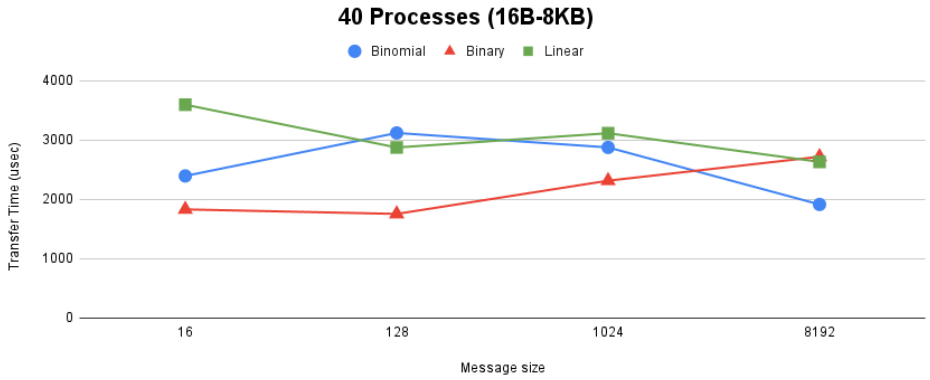


Figure 6.4 – The broadcast operation's latency varies depending on the number of processes 32 and the size of the buffers

In the process of transmitting messages utilizing 32 processes, the Linear tree emerged as a more effective method, particularly for smaller data. As illustrated in Figure 6.4a, it outperformed both the Binomial and Binary trees in terms of transfer time. This makes it the most recommended choice when dealing with compact data.

Conversely, when handling larger data, the Binary tree takes the lead in terms of performance. As depicted in Figure 6.4b, it demonstrates superior efficiency compared to the Binomial tree. Therefore, for substantial volumes of data, the Binary tree proves to be a more suitable option.

(a) – Small Data



(b) – Medium Data

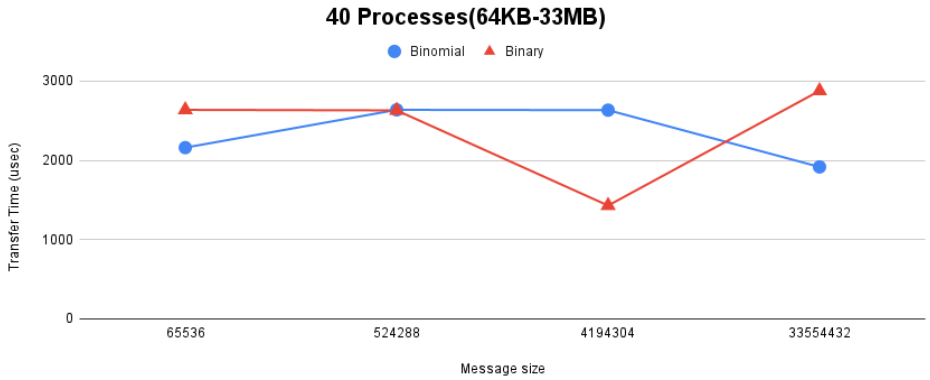


Figure 6.5 – The broadcast operation's latency varies depending on the number of processes 40 and the size of the buffers

The Binary tree showed a better transfer time in the job of sending messages utilizing 40 processes, especially for smaller data. It performs better than both the Binomial and Linear trees, as seen in Figure 6.5a, making it the favored option when working with lesser amounts of data.

However, the situation is different when dealing with bigger data . The Binomial tree exceeds the Binary tree in terms of efficiency, as illustrated in Figure 6.5b. This suggests that the Binomial tree becomes a better option for managing increasing amounts of data.

BUSINESS PLAN

1 Executive Summary

The name of my project is Algorithms for Implementing Collective Operations in the Remote Memory Access Model for Distributed Computing Systems.We have analysed the MPI_Bcast algorithms that most library's used Point to point technique.We have introduced One sided technique for MPI_Bcast techniques using Shared model.Since all communication is done via shared memory and just the shared memory needs to be set up once, it offers the fastest possible calculation performance over point to point.Our knowledge will help to compute large files sizes and broadcasts among all processes by using the connected machines to access there memories and processes and we can transfer it among machines.Our understanding of parallel computing may be applied to typical high-performance server and cluster architectures. And for many businesses or people who cannot afford servers, we may provide it in the form of cloud services. These services are accessible online.

Currently, there are not many cloud computing and high-performance computing companies in Russia like Yandex Cloud [31].

6,000,000 is allocated as a one-time investment, mostly for the first several months' worth of costs and equipment acquisitions. Five years are allotted for the operational time. In order to better serve customers, we will grow our business after five years. After five years, a revision of the business plan is required.

2 Description of the object and the company carrying out the project

Table 1 – Overview for Product Description

Product name	Parallel computing services
Purpose of the product	We are a cloud computing business that gives customers access to powerful hardware.

Main product feature	– Our deals often occur on a monthly basis, and of course we give loyal clients discounts. – Due to the utilization of cutting-edge virtualization technology, one computer may be utilized concurrently by hundreds of users with a minimum purchase need of only two processing cores. – Customers do not need to hire qualified maintenance staff or purchase expensive servers. This significantly lowers their expenditures. – Our products may be used by customers to do a variety of computations, including scientific calculations. – Users in the business world may use it to create corporate websites and application servers. – Customers may rent a high-performance computer remotely. Individual clients are hesitant to purchase pricey computers because they sometimes have restricted budgets or sporadic demands. They can execute huge program on our device via a remote desktop – It's as simple to purchase our services as internet purchasing. Customers don't need to worry about anything when they purchase, upgrade, suspend, or cancel computer resources.
Consumer properties of products	Consumer properties of products.The safe environment with 2 factor authentication Safe network by integrating with best tools in the market to avoid cypher attack Fast service with reliable performances

Product competitive advantages	– The emergence of new competitors – High performance – Upgrade system constantly – Support for multiple environment
The main consumers and uses of products	– Monitor status of product with the requirements – schedule tasks – multi clusters for palatalization tasks – fast support available for any problem with our server – Daily reports for the status of tasks and computing files
Legal protection of products	Licenses for the system with trademarks for database environment and environment trademark
Service	The client will sign a contract for using the product as he agreed in the contract after the trail phase ,warrant is included as the agreement in the contract,any change in the plan he purchased will be done at the update versions with extra fees

3 Marketing Plan

Market analysis

The size of the global cloud computing market, estimated at USD 405.65 billion in 2021, is expected to increase at a CAGR of 19.9% from 2022 to 2029, from USD 480.04 billion to USD 1,712.44 billion. According to our research, the worldwide market grew by 13.8% in 2020 compared to 2019. Cloud computing has shown higher-than-anticipated demand across all areas compared to pre-pandemic levels because to the unprecedented and overwhelming COVID-19 pandemic. The global cloud computing market is projected to grow from $480.04 billion in 2022 to $1,712.44 billion by 2029, at a CAGR

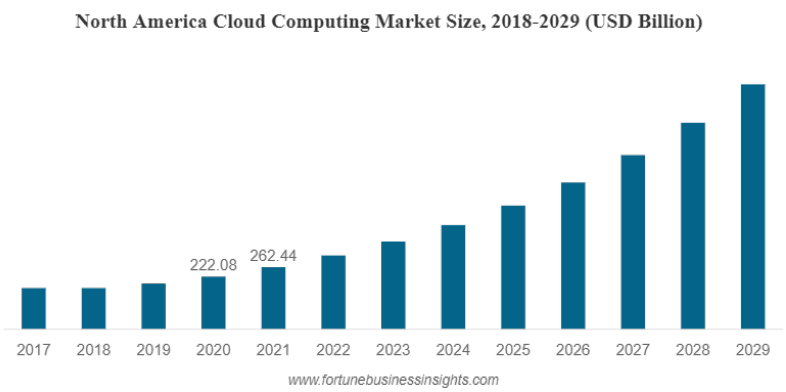


Figure 6.6 – the growth of cloud market size

of 19.9% in forecast period. The healthcare sector is anticipated to have the greatest

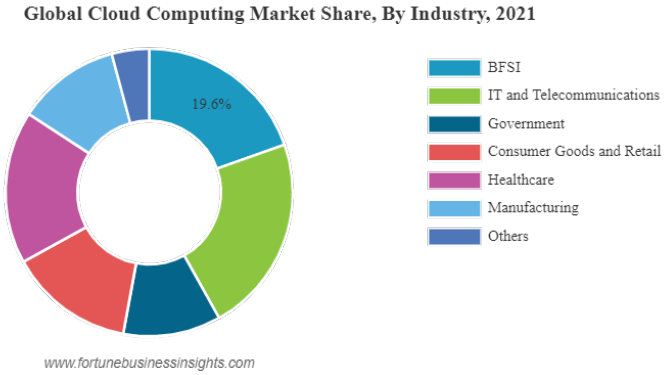


Figure 6.7 – Cloud market share by industry

Compound Annual Growth Rate over the research period (CAGR). The expanded use of cloud-based software, mobile apps, wearable medical equipment, and smart medical equipment, among other things, is responsible for this expansion. Due to growing government and cloud provider activities, as well as investment plans to boost cloud adoption among start-ups, other sectors, including retail and consumer goods, BFSI, government, manufacturing, and others, are anticipated to develop at a large pace.

Trade Policy

Product characteristics

– We support the development of a welcoming atmosphere for knowledge exchange. Universities and educational institutions may address each student and researcher on an individual basis, provide continuous access to their programs, and complete other activities related to digital education by using cloud technology.

Table 2 – SWOT analysis 1

Strengths	**Weaknesses**
Once the data is in the cloud, utilising the cloud to backup and recover the data is simpler.	You cannot access this data if the clients' internet connection is poor. We do not, however, have any other means of gaining access to cloud-based data.
By enabling groups of people to swiftly and easily exchange information on the cloud through shared storage and services, cloud apps promote collaboration.	Transferring an organization's services from one vendor to another might provide challenges. Moving from one cloud to another might be challenging since different providers offer various platforms.
Using the cloud and an internet connection, we can quickly and easily access information stored anywhere, at any time. By guaranteeing that our data is constantly available, an internet cloud architecture boosts organisational effectiveness and productivity.	
Cloud computing reduces both hardware and software maintenance costs for organizations	

Table 3 – SWOT analysis 2

Opportunities	**Threats**
one of the oportunity that the clint could have is digitizing his work to organize his work anywhere	The service could be disadvantage if the cloud hacked or the software fail
The brands offered, like software, have opportunity to become more competitive in the market.	After cloud servicing, there can be job losses since most human labour will be replaced by automation.
The chance exists to train personnel who are capable of managing cloud servicing so the operation could go independent	When a system malfunctions or is upgraded, service delivery could be impacted.
the potential to increase revenue from cloud services by lowering labour expenses to complete	

– Customer may construct and manage virtual machines using our compute cloud, which offers the scalable processing power you need. Preemptible virtual machines and fault-tolerant instance groups are supported by the service.

– Customers may connect to disks from the marketplace that include Linux images. To provide dependable data storage, each disk is automatically duplicated in its availability zone. Compute Cloud also enables disk snapshots to further simplify the process of transferring data from one storage device to another.
– Budgeting is possible for the customer, and cost optimization in our cloud is simple. Our flexible billing system allows you to simply keep track of your spending and only pay for what you really use.

Characteristics of the breadth

Before customer use Our cloud.He must go into this stages :-

1. Customer go to our pricing section
2. choose the specification that he needed (HPC,hosting..etc)
3. if the user have a question he can talk to our support.
4. Once a customer creates a billing account, they can test our platform out for two months.
5. Customer create Virtual machine
6. choose the operating system
7. if the customer have VM on our cloud he can extend his storage size with extra fee
8. Choose access policy if the customer wants to work with team
9. then connect to his OS and manage it using his administration panel

Packaging, packaging requirements

The customer after choosing his service and paying for it he will receive electronic contract and bill for what he chooses with links of tutorial of how to use the chosen services with documentation.

Contents of instruction for the product

1. customer log in his account.
2. go to administration panel
3. manage the service
4. register team and roles
5. connect to customer service and use his functionality

Brand concept

Our brand logo is in all of our products and pages it is registered as a copyright it is like a water mark that ensure the product from our company with license and signature

Distributive policy

Planning sales territory

Our company targets startup companies,researcher,students and Individuals located in Russia to compute there work or hosting there website ,database and application.There is lot of startup and researchers that's need our services.

Characteristics of distribution channels

We have a direct sales through our companies office and the head courter to sell our project with full support and discounts depends on the scale of our project. In addition to that we have an exclusive distribution channels with dealers we don't have an office in this place that's have a right to sell our products with also full support from our company

Distribution channel promotion strategy

Our push strategy is available after the finalization of the finance year there will be a lot of discounts for our services at this time .We also include our sales expert to put what is the percent of our discount .Our pull strategy is at the finance year we will advertise before a month that will be the end of our discount time so the customer can catch the discount with the last time at our office or traders that can sell our services

Communication policy

Table 4 – Communication policy

Tools	Channel	Concepts

Advertising	Newsletter	Sending offers to the e-mail addresses of wholesale companies,researcher and students specifies the purpose of the services and technical characteristics, offers and discounts
	Magazine advertising	Photos of our services in office that's used by different employees with photo of example of reports that's the system export with our logo and specification of our services with contact information[32]
	TV Channel Advertises	Photo section of our services with a video in the companies,student and researchers with our services and full feedback from their experiences with contact information and specification of our services and company background with our logo[32]
	Youtube	Youtube advertising with reviews from multiple youtubers that are expert in technologies of cloud services and applications [33]
Sales promoting	Clients	Our client will have free services like upgrading there storage or uses another services for a period of time if they give our contact to a needed company that's have or have not have a relation with
PR	Review articles in scientific journals	Reflect new properties of this category of goods and provide statistics on reliability
Exhibits	Participation in the exhibition	our customer and our representative will participate in Exhibits to win the prizes and promote our cloud

Internet representation	– company's website – Company's Social media [34]	Our cloud photos, specifications and multiple services like support,hosting and High Parallel Computing and advantages

calculation of the promotion budget

Table 5 – calculation of the promotion budget

Tools	**Name of the event**	**Period**	**Place promotion**	**Cost calculation**	**Total amount**
Magazine advertising	First page advertising	May 2024	Magazines and newspapers	136000	544000
TV Channel Advertises	After evening advertising 5 times	Mar 2024	Scientific channels and popular channels	78350	235050
Youtube advertising	2 minutes advertising	Jun 2024	At scientific and related videos	40000	320000
Social media advertising	10000 views and 1000 participants	Feb 2024	Scientific pages and groups that is related to our professional with capable users	200000	200000

Promotion schedule

Table 6 – Promotion schedule

Events	1quarter	2quarter	3quarter	4quarter	1quarter	2quarter	3quarter	4quarter
Magazine advertising								
Exhibit								
Social media								
Youtube advertising								
TV Channel Advertises								

Price policy

pricing strategy

our strategy is our cloud have a constant price even in time of rising prices maybe our price can be raised depends on the customer additions in the services and storage if it is customized or add feature or add a new period of support there is price discounts for startup companies and large period ,seasonal discount at the financial years and when releasing of new services our systems (1):

$$CostPrices = C + C * R/100 = C * (1 + R/100) \quad (1)$$

where $C = 2000$ – per month for computing cloud.
$R = 25\%$ – our profit

4 Production Plan

CHARACTERISTICS OF INDUSTRIAL BUILDINGS AND STRUCTURES

Table 7 – INDUSTRIAL BUILDINGS AND STRUCTURES

No	**type**	**Address**	**Square**	**status**	**Cost**
1	Development Center	Pulkolskvaya ST.Peter	60	Owned	5000000
2	Warehouse	Veteranov St.Peter	140	Owned	3890000

THE NEED FOR PRODUCTION EQUIPMENT

Table 8 – Main equipment

No	Name	Amount	Price	Price per unit	Cost without tax	Cost with tax	Provider
1	Laptop	50	2000000	40000	30000	40000	Lenovo distribution center
2	Server	5	800000	160000	120000	160000	Lenovo distribution center
3	Fingerprint Devices	3	48000	16000	12000	16000	Fingertec distribution center

Table 9 – Auxiliary equipment

No	Name	Amount	Price	Price per unit	Cost without tax	Cost with tax	Provider
1	Laptop	10	400000	40000	30000	40000	Lenovo distribution center
2	Server	2	320000	160000	120000	160000	Lenovo distribution center

Insurance 8%

The economic result

The difference between the present value of cash inflows and withdrawals over a period of time is known as **net present value**. It is used to evaluate a project's or investment's profitability. (4):

$$NPV = \sum [\frac{C_T}{(1+r)^t}] - C_0 \quad (2)$$

Table 10 – Programs license

No	Name	Amount	Price	Price per unit	Cost without tax	Cost with tax	Provider
1	Oracle Database Enter-prise Edition	1	3800000	3800000	2850000	3800000	Oracle distri-bution center
2	Mongo Database Dedi-cated Edition	3	164160	54720	41040	54720	Mongo Database Dedi-cated Edition
3	Security Frame-work	5	1200000	240000	180000	240000	Norton distri-bution center
4	**Total**		8732160		3372240		

Table 11 – NUMBER AND REMUNERATION OF MANAGERS AND OTHER CATEGORIES OF WORKERS

No	Position	Number of people	Position salary	Salary per month	Salary per year
1	General manager	1	130000	130000	1560000
2	Financial manager	1	91055	91055	1092660
3	Chief accounting officer	1	86298	86298	1035576
4	Marketing manager	1	88395	88395	1060740
5	Development Manger	1	112705	112705	1352460
6	Team leader	3	86512	259536	3114432
7	Senior Developer	6	80312	481872	5782464
8	Junior Developer	12	67645	81740	9740880

where C_T the net cash inflow during the period t , r is the discount rate and t is the number of periods and C_0 is the initial investment .We have:

– Initial investment: 6,000,000 rubles

– Monthly service per month 2000 rubles

– production cost 3050000 rubles

– fixed cost 17700 rubles

Table 12 – Production cost

Name costs	**Amount**
Main equipment, Auxiliary equipment and Programs license	8732160
Basic salary of production workers	24739212
Insurance payments	1979136.96
Business expenses	800000
General production expenses	450000
Total gross unit PRODUCTION COST	3050000

lets assume we will have 2000 customer per month .Therefore, your monthly revenue would be 4,000,000 rubles. the monthly production cost and the fixed cost, our monthly profit would be 932,300 rubles (4000000 rubles - 3050000 rubles - 17700 rubles). Over a year, your total profit would be 11187600 rubles.

$$NPV = \sum_{x=1}^{5} [\frac{11187600}{(1+0.1)^x}] - 6000000 \tag{3}$$

our NPV for 5 years will be 36409806 rubles or 7281961 rubles per year

Internal Rate of Return

$$0 = \sum [\frac{C_T}{(1+irr)^t}] - C_0 \tag{4}$$

$$0 = \sum_{x=1}^{5} [\frac{11187600}{(1+irr)^x}] - 6000000 \tag{5}$$

$IRR = 185\%$ a positive NPV indicates a profitable project, and an IRR greater than the cost of capital also indicates a worthwhile project.

5 Organizational Plan

Characteristics of the organization implementing the project

Our company is Limited liability company:

1. Writing the company contract and a summary of the contract through a lawyer
2. Heading to the commercial registry office to which the company's headquarters belongs: to review the legal terms of the contract presented by the lawyer,and the commercial registry seals the contract "valid for registration"
3. He goes to the court to which the company's headquarters belongs in order to register the contract and its summary in the commercial registry of the court and to publish on the bulletin board
4. The lawyer then goes to a daily newspaper for judicial announcements to publish a summary of the contract
5. He goes to the tax office affiliated to the company's headquarters to obtain the tax card
6. He goes to the Chamber of Commerce to which the company's headquarters is affiliated to obtain a license to practice trade or activity
7. He goes to the Chamber of Commerce to which the company's headquarters is affiliated to obtain a license to practice trade or activity
8. Go to the Social Insurance Office to register the company and employees

Table 13 – Information about the developers of the project and the project management team

Position	**Role in the project**	**Seniority**	**Education specialty**	**Experience in project implementation**	**Note**
General manager	Manage all the company	20 years	PHD,Accountant	– 2015,Manger Accountant in Russia – 2018, Business Manger Accountant in Egypt	
Sales manager	Mange Sales Department	8 years	Master, Economist	2016, Sales Manger at multi national company in Egypt	Best sales in 2019 in his old company
HR Manger	Mange HR department	6 years	University of California,Human Resource Management	2019,Senior HR employee at company in Dubai UAE	
Development Manger	Mange development department	15 years	LETI Computer Engineering department , ,Engineer	– 2014 senior developer in Amazon Ireland 2018 – 2018 Team leader Vodaphone England – 2020 Development Manger In Vodaphone England	
IT Manger	Mange IT Department	10 years	Cairo University computer science,Engineer	2017,IT Manger at Yandex Russia	

6 Pricing analysis

Price analysis in marketing refers to the examination of customer reactions to hypothetical pricing in survey research. Price analysis, in broad business terms, is the act of looking at and analyzing a proposed price without looking at its individual cost components and planned profit.

this stages :-

1. We take the requirements from the client and sign a contract of the requirement through the website
2. We take this requirement to solution architecture and database administrator
3. We take this requirement to solution architecture and database administrator
4. And put a plan for supporting the customer
5. Test on production environment before using use it
6. The client uses the services in the trial phase with tutorial of how to use the services

our team contain team leader 2 senior developers ,3 juniors developers , tester and product owner and scrum master so our price of services depend on that,cooling,electricity and license programs we can conclude that

1. 320000 for programs license
2. 400000 for developing and testing
3. 80000 for supporting

Table 14 – Production cost

Total production costs (per Server)	170700
Fixed cost	160000
Estimated renting price(per 10 virtual machine in server)	200000
Revenue per server	40000
Tax on revenue	3200

7 Risks

- The Altair PBS Pro task scheduler should only be used to execute resource-intensive applications on cluster nodes. Applications that need a lot of resources should not be executed on the front-end server; in this case, all user processes are abruptly and immediately stopped. Application launches on the front-end server repeatedly will restrict access.
- The needs of other users of the complex must be considered when seeking resources (processor cores, RAM). In particular, the task's required resources must correspond to those that were actually utilized.
  - You cannot request more than will be utilized since, even if it will be idle, the unused will be shown as busy and unavailable to other users. Running fewer processes than the number of CPU cores required is one example.
  - However, it is also not possible to use more resources than those that were requested because, at the same time, resources that the scheduler considers free and allots to other tasks start to be used. For instance, unless the job occupies the whole node, you cannot execute more than one resource-intensive activity per core. If there are any issues with the way the request for resources is written ("select=..."), one should ask the assistance
- In the event of a user's loss of data, illegal access to it, failure to finish computations by a deadline, or other harm, our firm disclaims all liability.

8 Conclusion

Our services will be an alternative for expensive cloud that's offer High Performance Computing in Russia.It will be a full function services so the customer can uses our services for everything that needs computing ,hosting or database services .Every client can see status and has a daily and monthly reports for monitoring the activity of him or his team.

The safe environment with 2 factor authentication Safe network by integrating with best tools in the market to avoid cyber attack Fast service with reliable performance.

CONCLUSION

This study presents a hybrid solution integrating message forwarding (across-node) and shared memory (on-node) techniques, developed in response to the widespread adoption of multi-core technology. This solution promotes reduced memory consumption, a vital factor in data-intensive operations.

In this paper, we introduced an approach built around the Binomial tree algorithm for MPI_Broadcast operations. By leveraging shared memory, we constructed performance models through a series of communication experiments. We further expanded this approach into a comprehensive method and applied it to Open MPI.

Our findings suggest that the proposed MPI_broadcast approach of RMA Shared memory for both Binomial tree and Binary tree algorithms is effective. The Linear tree algorithm demonstrated superior speed for data volumes ranging from 16 Bytes to 65KB, particularly for a smaller number of processes (fewer than 32).

However, for larger messages up to 33 MB, the Binomial and Binary trees were the fastest algorithms. As the number of processes and data sizes increase, the Binomial tree algorithm with the shared memory approach is strongly recommended.

Moving forward, we plan to continue refining this approach, integrating and testing more algorithms. Our aim is to identify the optimal algorithm for Broadcast messages on MPI within shared memory models. This research is a stepping stone towards more efficient and effective data transfer methods, contributing to the broader goal of optimizing large-scale data operations.

REFERENCES

1. Optimisation and performance evaluation of mechanisms for latency tolerance in remote memory access communication on clusters / J. Nieplocha [et al.] // International Journal of High Performance Computing and Networking. — 2004. — Vol. 2, no. 2–4. — DOI: 10.1504/ijhpcn.2004.008904.
2. Remote memory access programming in MPI-3 / T. Hoefler [et al.] // ACM Transactions on Parallel Computing. — 2015. — Vol. 2, no. 2. — DOI: 10.1145/2780584.
3. MPI: A message-passing interface standard Version 3.0 / E. Lusk [et al.] // International Journal of Supercomputer Applications. — 2009. — Vol. 8, no. 3/4.
4. OpenMPI. Open MPI: Open Source High Performance Computing // OpenMPI. — 2014.
5. Tipparaju V., Nieplocha J., Panda D. Fast collective operations using shared and remote memory access protocols on clusters // Proceedings - International Parallel and Distributed Processing Symposium, IPDPS 2003. — 2003. — DOI: 10.1109/IPDPS.2003.1213188.
6. A high-performance, portable implementation of the MPI message passing interface standard / W. Gropp [et al.] // Parallel Computing. — 1996. — Vol. 22, no. 6. — DOI: 10.1016/0167-8191(96)00024-5.
7. Walker D. W., Dongarra J. J. MPI: A standard message passing interface // Supercomputer. — 1996. — Vol. 12, no. 1.
8. Performance analysis and optimization of MPI collective operations on multi-core clusters / B. Tu [et al.] // Journal of Supercomputing. — 2012. — Vol. 60, no. 1. — DOI: 10.1007/s11227-009-0296-3.
9. Wadsworth D. M., Chen Z. Performance of MPI broadcast algorithms // IPDPS Miami 2008 - Proceedings of the 22nd IEEE International Parallel and Distributed Processing Symposium, Program and CD-ROM. — 2008. — DOI: 10.1109/IPDPS.2008.4536478.
10. Zhou H., Gracia J., Schneider R. MPI collectives for multi-core clusters: Optimized performance of the hybrid MPI+MPI parallel codes // ACM International Conference Proceeding Series. — 2019. — DOI: 10.1145/3339186.3339199.
11. Efficient algorithms for all-to-all communications in multiport message-passing systems / J. Bruck [et al.] // IEEE Transactions on Parallel and Distributed Systems. — 1997. — Vol. 8, no. 11. — DOI: 10.1109/71.642949.
12. A case for standard non-blocking collective operations / T. Hoefler [et al.] // Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). 4757 LNCS. — 2007. — DOI: 10.1007/978-3-540-75416-9{_}22.
13. Hoefler T., Lumsdaine A. Optimizing non-blocking collective operations for InfiniBand // IPDPS Miami 2008 - Proceedings of the 22nd IEEE International Parallel and Distributed Processing Symposium, Program and CD-ROM. — 2008. — DOI: 10.1109/IPDPS.2008.4536138.

14. Hoefler T., Schneider T. Optimization principles for collective neighborhood communications // International Conference for High Performance Computing, Networking, Storage and Analysis, SC. — 2012. — DOI: 10.1109/SC.2012.86.

15. Hoefler T., Moor D. Energy, memory, and runtime tradeoffs for implementing collective communication operations // Supercomputing Frontiers and Innovations. — 2014. — Vol. 1, no. 2. — DOI: 10.14529/jsfi140204.

16. MPI on a million processors / P. Balaji [et al.] // Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). 5759 LNCS. — 2009. — DOI: 10.1007/978-3-642-03770-2{_}9.

17. The Art of Multiprocessor Programming, Second Edition / M. Herlihy [et al.]. — 2020. — DOI: 10.1016/C2011-0-06993-4.

18. Sterling T. Parallel Programming with MPI // Beowulf Cluster Computing with Windows. — 2018. — DOI: 10.7551/mitpress/1557.003.0014.

19. Gottlieb A. Multi-purpose highly parallel computing (after the killer micros take a breather) // Developing a computer science agenda for high-performance computing. — 1994. — DOI: 10.1145/197912.197941.

20. Karniadakis G. E., Kirby R. M. I. Parallel Scientific Computing in C++ and MPI: A Seamless Approach to Parallel Algorithms and Their Implementation // Book. — 2003. — P. 616.

21. Amdahl G. M. Validity of the single processor approach to achieving large scale computing capabilities // AFIPS Conference Proceedings - 1967 Spring Joint Computer Conference, AFIPS 1967. — 1967. — DOI: 10.1145/1465482.1465560.

22. Pacheco P. S., Malensek M. An Introduction to Parallel Programming. — 2021. — DOI: 10.1016/B978-0-12-804605-0.00002-6.

23. Czarnul P. Parallel Programming for Modern High Performance Computing Systems. — 2018. — DOI: 10.1201/b22395.

24. Cappello F., Etiemble D. MPI versus MPI+OpenMP on the IBM SP for the NAS benchmarks // Proceedings of the International Conference on Supercomputing. 2000–November. — 2000. — DOI: 10.1109/SC.2000.10001.

25. Collectives in hybrid MPI+MPI code: Design, practice and performance / H. Zhou [et al.] // Parallel Computing. — 2020. — Vol. 99. — DOI: 10.1016/j.parco.2020.102669.

26. Träff J. L., Rougier A. MPI collectives and datatypes for hierarchical all-to-all communication // ACM International Conference Proceeding Series. 09-12-September–2014. — 2014. — DOI: 10.1145/2642769.2642770.

27. LeBlanc T. J., Markatos E. P. Shared memory vs. Message passing in shared-memory multiprocessors // Proceedings of the 4th IEEE Symposium on Parallel and Distributed Processing, SPDP 1992. — 1992. — DOI: 10.1109/SPDP.1992.242736.

28. Nuriyev E., Rico-Gallego J. A., Lastovetsky A. Model-based selection of optimal MPI broadcast algorithms for multi-core clusters // Journal of Parallel and Distributed Computing. — 2022. — Vol. 165. — DOI: 10.1016/j.jpdc.2022.03.012.

29. Hoefler T., Siebert C., Rehm W. A practically constant-time MPI broadcast algorithm for large-scale InfiniBand clusters with multicast // Proceedings - 21st International Parallel and Distributed Processing Symposium, IPDPS 2007; Abstracts and CD-ROM. — 2007. — DOI: 10.1109/IPDPS.2007.370475.

30. Patarasuk P., Yuan X. Efficient MPI_Bcast across different process arrival patterns // IPDPS Miami 2008 - Proceedings of the 22nd IEEE International Parallel and Distributed Processing Symposium, Program and CD-ROM. — 2008. — DOI: 10.1109/IPDPS.2008.4536308.

31. Yandex. — https://cloud.yandex.com/en/docs.

32. Advertising Rates. — http://www.passportmagazine.ru/468/.

33. growthtactics. YouTubeadscost? — 09/2019. — https://www.demandcurve.com/blog/youtube-ads-cost.

34. Rate(CPM) VK. — https://vk.com/faq11887.