IJARSCT
ISSN (Online) 2581-9429
International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)
Volume 2, Issue 7, May 2022
Impact Factor: 6.252

Performance Analysis of Serial Computing Vs. Parallel Computing in MPI Environment

Anurag[1], Arjun V P[2], Akash Chauhan[3], Manoj Kumar Yadav[4]
Students, Department of Computer Science And Engineering[1,2,3],
Assistant Professor, Department of Computer Science and Engineering[4]
Dronacharya Group of Institutions, Greater Noida, UP, India

Abstract: *Parallel computing has been known for years, but it has only suddenly grown in popularity as a result of the multi core processors and machine learning to general public at reasonable cost. The goal of this paper is to compare the performance of serial versus parallel algorithm in MPI(Message Passing Interface) environment.*

Keywords: Central Processing Unit (CPU); Message Passing Interface (MPI); Parallel Computing;

I. INTRODUCTION

Parallel computing is the process of running an application or a calculation on many processors at the same time. It's a type of computing architecture in which enormous problems are split into smaller parts which can compute in one go. Multiple CPUs communicate via shared memory to complete the task, which then combines the results.

1.1 History of Parallel Computing

In April 1958, Stanley Gill (Ferranti) discussed parallel programming and the need for branching and waiting.[1] Also in 1958, IBM researchers John Cocke and Daniel Slotnick discussed the use of parallelism in numerical calculations for the first time.[2]

Burroughs Corporation introduced the D825 in 1962, a four-processor computer that accessed up to 16 memory modules through a crossbar switch.[3] In 1967, Amdahl and Slotnick published a debate about the feasibility of parallel processing at American Federation of Information Processing Societies Conference.[2] It was during this debate that Amdahl's law was coined to define the limit of speed-up due to parallelism.

In 1969, Honeywell introduced its first Multics system, a symmetric multiprocessor system capable of running up to eight processors in parallel.[2] C.mmp, a multi-processor project at Carnegie Mellon University in the 1970s, was among the first multiprocessors with more than a few processors. The first bus-connected multiprocessor with snooping caches was the Synapse N+1 in 1984.[4]

1.2 Message Passing Interface (MPI)

Message Parsing Interface (MPI) is a specification for the developers and users of message passing libraries. By itself, it is NOT a library - but rather the specification of what such a library should be.[5]

In MPI model the data is transferred from one address of a process to another process with in a co-operative manner to prevent data loss. The main objective of the MPI model is to provide developers a standard for writing message passing programs. The goal of this interface are:

- Flexible
- Portable
- Efficient
- Practical

1.2.1 History of MPI

MPI has resulted from the efforts of numerous individuals and groups that began in 1992.

IJARSCT
ISSN (Online) 2581-9429
International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)
Volume 2, Issue 7, May 2022
Impact Factor: 6.252

Some history:

1980s - early 1990s: Distributed memory, parallel computing develops, as do a number of incompatible software tools for writing such programs - usually with tradeoffs between portability, performance, functionality and price. Recognition of the need for a standard arose. [5]

Apr 1992: Workshop on Standards for Message Passing in a Distributed Memory Environment, sponsored by the Center for Research on Parallel Computing, Williamsburg, Virginia. The basic features essential to a standard message passing interface were discussed, and a working group established to continue the standardization process. Preliminary draft proposal developed subsequently.[5]

Nov 1992: Working group meets in Minneapolis. MPI draft proposal (MPI1) from ORNL presented. Group adopts procedures and organization to form the MPI Forum. It eventually comprised of about 175 individuals from 40 organizations including parallel computer vendors, software writers, academia and application scientists.[5]

- Nov 1993: Supercomputing 93 conference - draft MPI standard presented.
- May 1994: Final version of MPI-1.0 released
- MPI-1.1 (Jun 1995)
- MPI-1.2 (Jul 1997)
- MPI-1.3 (May 2008).
- 1998: MPI-2 picked up where the first MPI specification left off, and addressed topics which went far beyond the MPI-1 specification.
- MPI-2.1 (Sep 2008)
- MPI-2.2 (Sep 2009)
- Sep 2012: The MPI-3.0 standard was approved.
- MPI-3.1 (Jun 2015)
- Current: The MPI-4.0 standard is under development.[5]

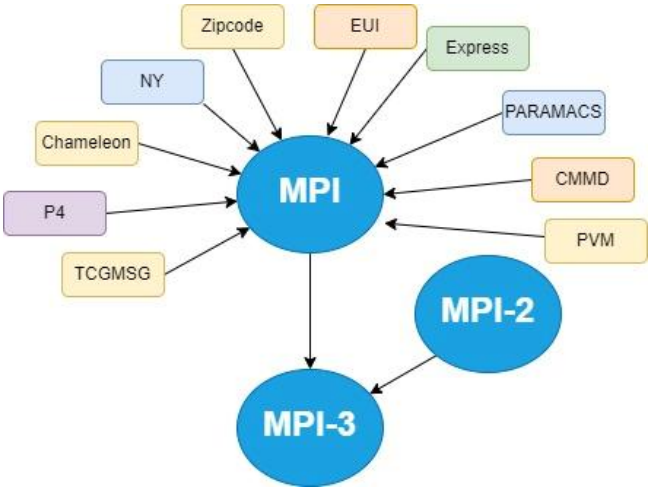

Fig. 1. Development of MPI

1.3 Open MPI

The Open MPI Project is an open source Message Passing Interface implementation that is developed and maintained by a consortium of academic, research, and industry partners.[5]

Open MPI is therefore able to combine the expertise, technologies, and resources from all across the High-Performance Computing community in order to build the best MPI library available. Open MPI offers advantages for system and software vendors, application developers and computer science researchers.[5]

It is available in different languages such as C, C++ and Fortan. It provides easy syntaxes to implement and easily and also enables powerful command over CPU scheduling to optimize our code very efficiently. In this paper we use Open MPI to demonstrate the advantages of parallel computing.

IJARSCT
ISSN (Online) 2581-9429
International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)
Volume 2, Issue 7, May 2022
Impact Factor: 6.252

1.4 **Serial Vs. Parallel**

1.4.1 Serial Programming: In serial programming the problem is divided into tasks(instruction) and the executed consecutively on a single core. This means that CPU only runs the next task until pervious job has not finished.

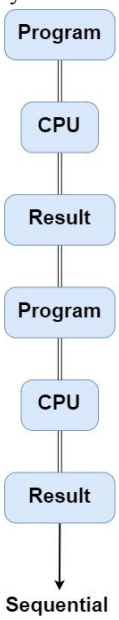

Fig. 2. Serial programming schema

This approach arose many limitations some of them are as follows:

- Lower Speed of execution:- The computation with this approach is much slower because in multi core CPU only one core is utilized and rest of the core stays idle
- Limits to miniaturization:- To increase the processor computing power large number of transistor are required which are very closed and placed extremely close (in nanometres). So a limit will be reached for how small can component would be.

1.4.2 Parallel Programming

In parallel computing the problem is divided into smaller task and distributed among all the cores of the CPU and combine output of each of core and shows result to the user.

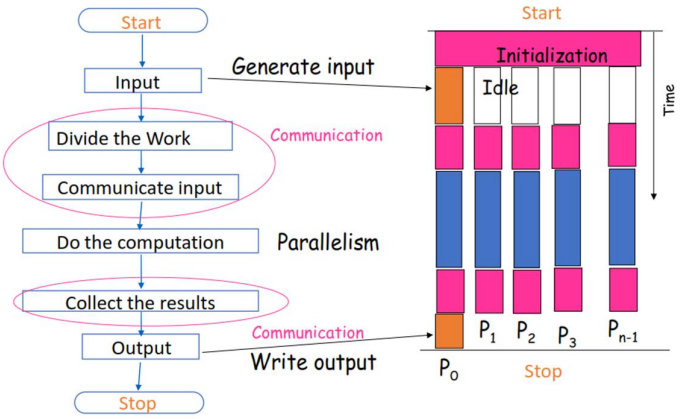

Fig. 3. Parallel programming schema

IJARSCT
ISSN (Online) 2581-9429
International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)
Volume 2, Issue 7, May 2022
Impact Factor: 6.252

This approach is way better than serial and have some major advantages some of them are as follows:

- High Speed of execution:- All the tasks are distributed equally to each of the core which decrease the computation cost and increase speed of execution.
- Ability to solve larger problems:- Many issues are so huge and/or complicated that solving them on a single computer is impractical or impossible, particularly given computer memory constraints.

II. OBJECTIVE

The main objective of this performance comparison is to analyse the performance difference of code written in parallel as compared to serial of same algorithm on the same machine and push the limits of each of the programming approach to see the worst case conditions.

III. EXPERIMENT

We took the matrix multiplication as problem statement to solve. The reason behind choosing this problem because matrix multiplication is widely used in many complex and large problems. Matrix multiplication is one of the problems that can be parallelized with high efficiency, which is best suited to demonstrate the advantage of MPI. Hence we took this problem to benchmark each of the programming paradigm.

3.1 Serial Matrix Multiplication

Here is the code snippet of Matrix multiplication of serial version :

```
for(i=0;i<r;i++)
    {
        for(j=0;j<c;j++)
        {
            mul[i][j]=0;
            for(k=0;k<c;k++)
            {
                mul[i][j]+=a[i][k]*b[k][j];
            }
        }
    }
```

Figure 4: Serial code algorithm for matrix multiplication

I tested this algorithm for matrix multiplication in Ryzen 5 2500u(4 Core CPU) and below are my findings:

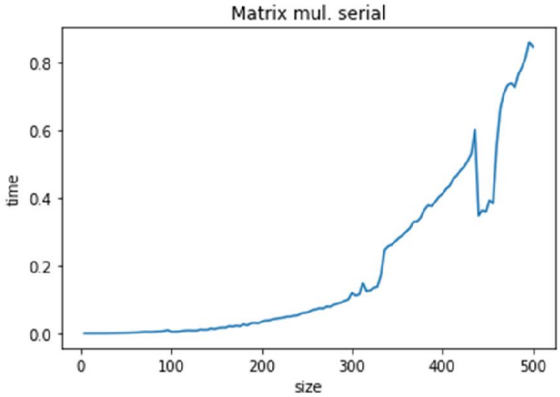

Fig. 5. Serial code performance over size of array

IJARSCT
ISSN (Online) 2581-9429
International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)
Volume 2, Issue 7, May 2022
Impact Factor: 6.252

As we can see that the performance of serial code degrades over time as the size of the matrix increases because it give instruction to CPU one by one and hence more work has to be done by single core as size increase.

3.2 Parallel Matrix Multiplication

Here is the code snippet of Matrix multiplication of parallel version :

```
MPI_Scatter(a, N*N/size, MPI_INT, aa, N*N/size, MPI_INT,0,
                MPI_COMM_WORLD);


    MPI_Bcast(b, N*N, MPI_INT, 0, MPI_COMM_WORLD);


          for (i = 0; i < N; i++)
            {
                    for (j = 0; j < N; j++)
                    {
                            sum = sum + aa[j] * b[j][i];
                    }
                    cc[i] = sum;
                    sum = 0;
            }

    MPI_Gather(cc, N*N/size, MPI_INT, mul, N*N/size, MPI_INT,0,
                MPI_COMM_WORLD);
```

Fig. 1. Parallel code of matrix multiplication

In this code we first split the matrix A as strip and send each strip to each core and broadcast B matrix to all the cores. Then each core compute the answer for a row then we collect all the calculated strips at one place (i.e. at core 0) and display the output. I tested this algorithm with 4 cores on my PC with different sizes of matrices which is shown below:

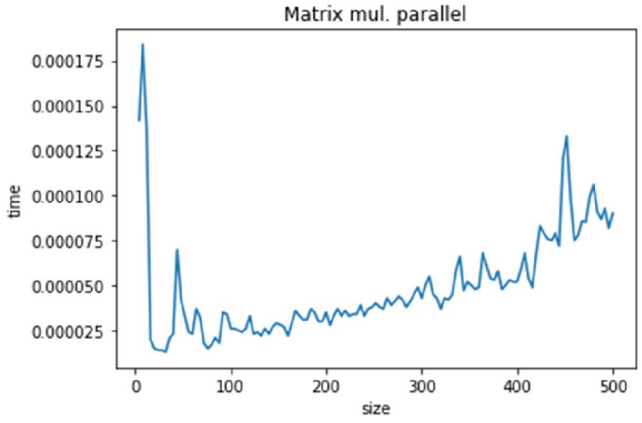

Fig. 1. Parallel code performance over size

As we can see that parallel version of code didn't perform well for small value of size of matrix because of communication overhead, but when the size increases that communication overhead become negligible and we get good performance.
We plot efficiency graph to compare better below:

IJARSCT
ISSN (Online) 2581-9429
International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)
Volume 2, Issue 7, May 2022
Impact Factor: 6.252

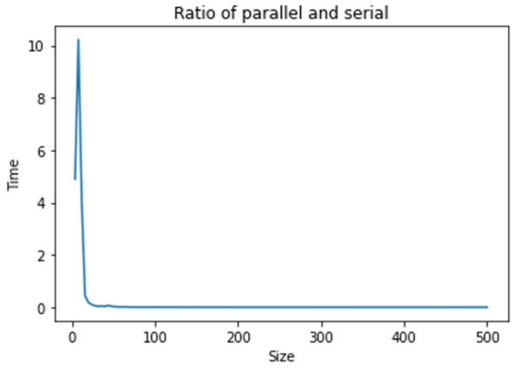

Fig. 1. Efficiency graph parallel vs. serial

We can clearly see that the serial version is good in less computation related task but when complexity increase it starts to struggle. In case of parallel version, we see more computation time in less intense task because communication time but when in complex task this communication time becomes negligible and give consistent performance in higher matrix sizes.

IV. RESULT

From our experiment we finds that the higher computational resource task are much easily computed when we code it into the parallel version as compared to serial version but for less complex task the communication time between the cores becomes significantly larger and hence performance degrades.

V. CONCLUSION

This parallel programming paradigm are meant for those tasks which are very complex in nature but it is not necessary to implement in each of the program because it may degrade its performance. With the advent of multi-core processors and the universal expectation that the number of cores on a chip will increase exponentially over the next years (in line with the original prediction of Moore's law [7])

VI. ACKNOWLEDGMENT

This work would not have been possible without the cooperation of many others, some of whom contributed directly and others indirectly, such as the various resources available online including and not limited to various other chat applications, libraries, research papers, theoretical notes etc which allowed us to recognise this notion. We owe a debt of gratitude and respect to our Project mentor, Manoj Kumar Yadav (Assistant Professor-CSE), for his invaluable guidance, opinion, encouragement, and support during the semester. We appreciate the college's assistance in providing the required infrastructure and technical support for the project's completion. Finally, we want to express our gratitude to our family and friends for their unwavering support.

REFERENCES

[1]. "Parallel Programming", S. Gill, The Computer Journal Vol. 1 #1, pp2-10, British Computer Society, April 1958.
[2]. Wilson, Gregory V. (1994). "The History of the Development of Parallel Computing". Virginia Tech/Norfolk State University, Interactive Learning with a Digital Library in Computer Science. Retrieved 2008-01-08.
[3]. Anthes, Gry (November 19, 2001). "The Power of Parallelism". Computerworld. Archived from the original on January 31, 2008. Retrieved 2008-01-08.
[4]. Patterson and Hennessy, p. 753.
[5]. Lawrence Livermore, & Blaise Barney. (n.d.). *What is MPI*. LLNL HPC Tutorials. Retrieved May 27, 2022, from https://hpc-tutorials.llnl.gov/mpi/what_is_mpi/
[6]. Open-mpi.org. 2022. Open MPI: Open Source High Performance Computing. [online] Available at: <https://www.open-mpi.org/> [Accessed 27 May 2022].
[7]. Jesper Larsson Träff," What the parallel-processing community has (failed) to offer the multi/many-core generation", Elsevier Inc., pp. 807-812, 2009.