Comparative Study of Parallel Programming Models to Compute Complex Algorithm

Mukul Sharma (Research Scholar)
Department of Computer Science & Engineering,
SBCET, Jaipur Rajasthan, India

Pradeep Soni (Research Scholar)
Department of Computer Science & Engineering,
SBCET, Jaipur, Rajasthan, India

ABSTRACT
The main goal of this research is to use OpenMP, Posix Threads and Microsoft Parallel Patterns libraries to design an algorithm to compute Matrix Multiplication effectively. By using the libraries of OpenMP, Posix Threads and Microsoft Parallel Patterns Libraries, one can optimize the speedup of the algorithm. First step is to write simple program which calculates a predetermined Matrix and gives the results, after compilation and execution of the code. In this stage only single core processor is used to calculate the Matrix multiplication.
Later on, in this research OpenMP, Posix Threads and Microsoft Parallel Patterns libraries are added separately and use some functions in the code to parallelize the computation, by using those functions multi-cores of a processor are allowed. Then execute the program and check its run time, then a timer function is added to the code which periodically checks the time it took for the computer to do the parallelization. First the program is run without the Parallel libraries, and then with the OpenMP, Posix Threads and with Microsoft Parallel Patterns libraries code.
Then program is executed for each input Matrix size and result is collected. Maximum 5 trials for each input size are conducted and record the time it took for the computer to parallelize the Matrix multiplication.
Finally comparison of the performance in terms of execution time and speed up for OpenMP, Posix Threads and Microsoft Parallel Patterns libraries is done using different Matrix Dimensions and different number of processors.

***Keywords*:** Parallel Computing, Parallel Programming models, Open MP, PThreads, Microsoft Parallel Patterns Libraries.

1. INTRODUCTION
Parallel computing includes all the architectures and software terms which are related to the applications that run concurrently [1]. It means that the use of more than one resource at a time to compute a problem to get the result is parallel computing.
A processor is typically a unit of 32 or 64 bits or variable-length chunks of data that can read and run program instructions. The data in the instruction guide the processor that how to do the work. The work of instructions are to read data from memory or send data to the user display as an output, but this work is done so fast and take very less time that one can get the results smoothly.
A part of the processor that performs reading and executing the instruction is 'Core'. Single core processors means one instruction at a time can be executed. However as the name implies, Multi-core processors are collection of more than one core, that can run multiple instruction at a time. A very common example would be a dual core processor [2].

The advantage of using dual core processor in the place of single core is that it can either use both its cores to complete a single task or it can span threads which divided tasks between both its cores, so that it takes twice the amount of time it would take to execute the task than it would on a single core processor. Also more than one task can be performed by Multi- core processors at a particular time. A common example would be listening to music on windows media player while dual-core processor is running a background virus check. As Multi-core processor is using the shared memory so both cores share the same memory address, and on the same chip on a multi-core architecture. So memory is shared in multi-core processors, so any change in one core is reflected to other one. Following are the Parallel Programming Models used in this research:

1.1 OpenMP
OpenMP (Open Multiprocessing) can be defined as an Application Programming Interface which can operate on different platforms for shared memory multiprocessing programming in C, C++ and Fortran languages, on different processor architectures and operating system environments, including Windows, AIX, HP-UX, GNU/Linux, Mac OS X, and Solaris platforms. It includes a set of compiler directives, library routines, and environment variables that affects the dynamic behavior [5]. Management for OpenMP is done by the nonprofit technical consortium OpenMP Architecture Review Board (or OpenMP ARB), jointly defined by a group of major computer hardware and software organizations, including AMD, IBM, Intel, Cray, HP, Fujitsu, Nvidia, NEC, Microsoft, Texas Instruments, Oracle Corporation, and other organizations.
OpenMP implements a portable and scalable model which provides developers a simple and efficient interface for creating parallel programs and applications for various platforms including the standard desktop systems to the supercomputers. An application built with the hybrid model of parallel computing can execute upon a computer cluster using both OpenMP and Message Passing Interface (MPI) programming Models, or more clearly by the use of OpenMP extensions of non-shared Memory systems.

1.2 Posix Threads
The Pthreads api is basically POSIX C API thread library that has standardized inbuilt functions for using threads across different platforms. Posix Threads Provides a functionality to take full advantage of the capabilities provided by threads, it is a standardized programming Model. For UNIX systems, this interface has been specified by the IEEE POSIX 1003.1c standard (1995). Implementations that adhere to this standard are referred to as POSIX threads, or Pthreads. Most hardware vendors now offer Pthreads in addition to their proprietary API's. Pthreads can be defined as a collection of C language

programming types and procedure calls. Vendors usually provide a Pthreads implementation in the form of a header or include file and a library that can be easily link with any program [8].

1.3 Microsoft Parallel Patterns Libraries

Microsoft's Parallel Patterns Library (PPL) gives an imperative programming model that improves scalability as well as user interaction in the development of Parallel applications. The PPL is developed on the scheduling and resource management components of the Concurrency Runtime. It increases the level of abstraction between the application program and the underlying threading mechanism by providing generic, type-safe algorithms and containers that work on data concurrently. The PPL also provides a way to develop applications that scale by giving solutions to shared state.

The PPL make the following features available:

- ***Task Parallelism*:** it provides a way to perform several tasks in parallel
- ***Parallel algorithms*:** they are the generic algorithms which works on collections of data in parallel
- ***Parallel containers and objects*:** the generic container types which gives safe concurrent access to their elements

2. MOTIVATION

The high complexity of Matrix multiplication when performed with large dimensions motivates us to explore a new algorithm for fast computation, using different libraries to solve problem with multiple threads simultaneously. Even the system does not allow to declare the matrices with more than one thousand dimensions.

The computation of multiplication of large matrices can involve billions of steps. By using this approach, one can reduce the memory latency and can be handled efficiently with multiple threads by OpenMP runtime library, Posix Threads and Microsoft Parallel Patterns libraries.

3. PROPOSED APPROACH

Matrix multiplication when performed with large dimension matrices consists the great complexity. Then first write the algorithm for Matrix multiplication and then modify it by adding the OpenMP, Posix Threads and Microsoft Parallel Patterns libraries and functions. Then a timer function to calculate the execution time for 2, 4, 6, and 8 processors using OpenMP, Posix Threads and Microsoft Parallel Patterns API, is added separately.

A problem is faced then, that one cannot be able to define Matrix with more than 1000x1000 dimensions. While defining the Matrix with much size the "Segmentation fault" occurred.

So dynamic memory allocation for the input Matrix is performed. This approach is important for this research because with large dimension input matrices the differences between the execution time and speedup is more clear for algorithm with OpenMP, Posix Threads and Microsoft Parallel Patterns API's and without these parallel API's.

4. EXPERIMENTAL SETUP

In this research, all the tests are performed under following specifications:

a) Host System: Intel i5 processor with 6 GB RAM and 500 GB Hard disk.

b) Operating Environment: Ubuntu 12.04 is used for Posix Threads/Open MP and Windows 8 is used for Microsoft Parallel Patterns libraries for all the tests.

c) Posix Threads api: API with gcc++ 4.7 is used.

d) OpenMP api: Intel® OpenMPx api 4.0

e) Microsoft Parallel Patterns api: Visual Studio 2013 is used for Task Parallelism.

4.1 Performance Metrics of Parallel Systems

a) Speedup: Speedup Sp can be defined in terms of the ratio of the execution time using single core of the sequential algorithm in order to solve a problem to the time used by the algorithm implementing parallel computing while solving the same problem on $\boldsymbol{p}$ processor [7]. All the processors used by the parallel algorithm must be having same characteristics with the processor which is used in the sequential algorithm.

$$\textbf{Speedup } \boldsymbol{Sp} = \frac{\textbf{Sequential time for Scaled}-\textbf{up workload}}{\textbf{Parallel time for Scaled}-\textbf{up workload}}$$

b) Execution Time: Execution time can be defined in terms of time consumed by an algorithm in order to solve a problem using processor p.

c) Serial runtime: The total time consumed in the overall execution of the serial calculations. Generally denoted by TS.

d) Parallel runtime: Total time consumed in the overall execution of parallel calculations, computations and processing elements (PE). Generally denoted by TP.

5. RESULTS AND ANALYSIS

1) **Experiemnt-1: (Matrix Dimensions (500x500), (500x500)** Execution time without using Parallel libraries is 0.91 Seconds.

Table 1
Execution Time and Speed Up for Matrix Dimensions (500x500), (500x500)

Programming Model	Execution Time	Speedup (in%)
OpenMP	0.64	42.18
PThread	0.49	85.71
Parallel Patterns Lib.	0.84	8.33

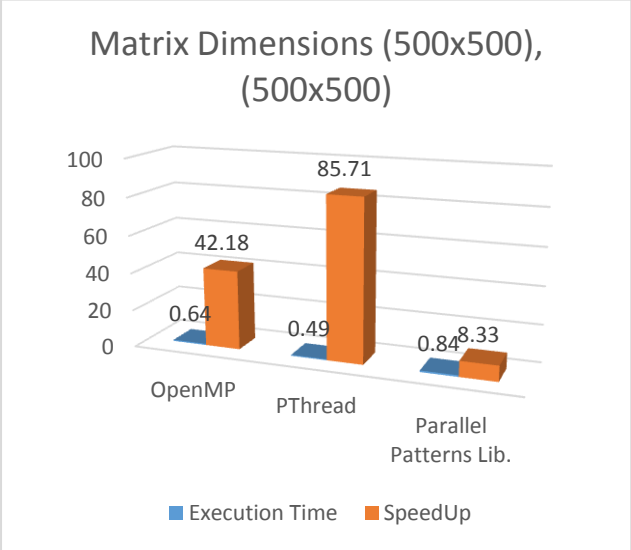

Figure 1: Execution time and Speedup for Matrix Dimensions (500x500), (500x500)

From the figure 1 it is clear that the speedup in Posix Threads is maximum for Matrix Dimensions (500x500), (500x500).

2) **Experiment-2:- (Matrix Dimensions (1000x1000), (1000x1000))** Execution time without using Parallel libraries is 9.34 Seconds.

Table 2
Execution Time and Speed Up for Matrix Dimensions (1000x1000), (1000x1000)

Programming Model	Execution Time	SpeedUp
OpenMP	6.01	55.40
PThread	5.93	57.50
Parallel Patterns Lib.	7.869	18.69

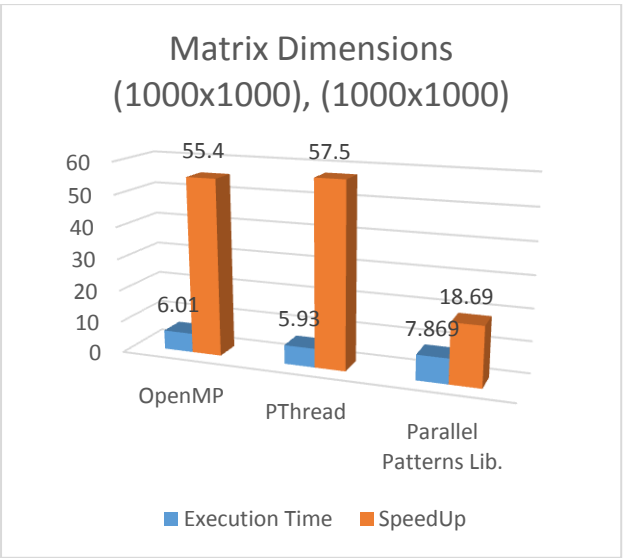

Figure 2: Execution time and Speedup for Matrix Dimensions (1000x1000), (1000x1000)

From the figure 2 it is clear that the speedup in Posix Threads is maximum for Matrix Dimensions (1000x1000), (1000x1000).

3) **Experiment-3:- (Matrix Dimensions (1500x1500), (1500x1500))** Execution time without using Parallel libraries is 45.89 Seconds.

Table 3
Execution Time and Speed Up for Matrix Dimensions (1500x1500), (1500x1500)

Programming Model	Execution Time	Speedup (in %)
OpenMP	26.63	72.32
PThread	28.68	60.00
Parallel Patterns Lib.	32.091	42.99

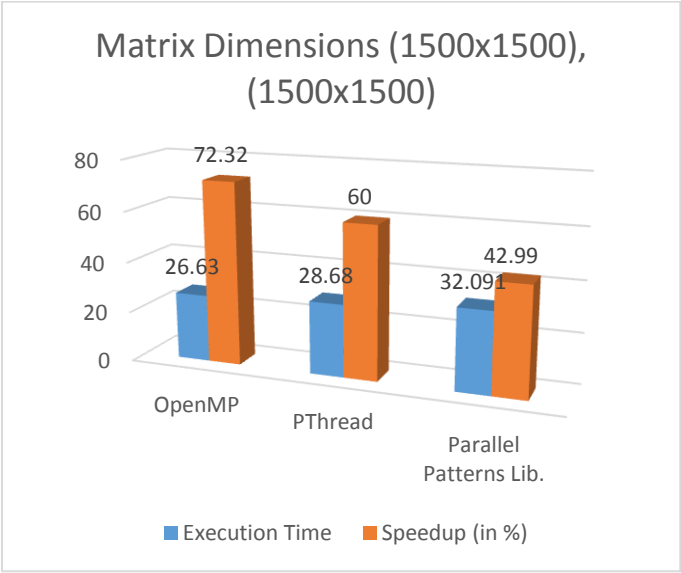

Figure 3: Execution time and Speedup for Matrix Dimensions (1500x1500), (1500x1500)

From the figure 3 it is clear that the speedup in OpenMP is maximum for Matrix Dimensions (1500x1500), (1500x1500).

6. CONCLUSION

As the dimensions of matrices increased, the difference between the execution time taken by the computer for the computation of matrices with OpenMP, PThread and Microsoft Parallel Patterns APIs increases significantly.

For the relatively smaller dimensions of matrices Pthreads provides the best outcomes in terms of speedup but as the matrix dimensions increases OpenMP provides better outcomes comparatively PThreads and Parallel Patterns Libraries.

As the number of Matrix Dimensions increased the speedup using Microsoft Parallel Patterns libraries doesn't increase significantly because of the problem of overhead while in OpenMP and PThread speedup increases significantly.

Speedup of this algorithm with OpenMP in comparison to PThread and Microsoft Parallel Patterns libraries would approach better in all the different experiments.

7. FUTURE SCOPE

The future of parallel computing is bright, but with great opportunities it includes challenges.

Generally, in a compiler implementation, the compiler can infer the type of the data element of the intermediate representation of the guarded function invocation. Therefore, it is not necessary to ask the programmer to specify it again in the API. So, the API does not need to have this field, which would make the API simpler and cleaner. There are other similar issues in the current design which can be optimized in a decent compiler implementation in all Programming Models. Therefore, a final decision on the design of the API needs a comprehensive discussion to decide which fields need to be kept in the API to make it simpler.

The implementation of much complex algorithms such as Tower of Hanoi which is not possible using OpenMP API because of the recursion and waiting problems can be done.

8. REFERENCES

[1] Blaise Barney, "Introduction to Parallel Computing", Lawrence Livermore National Laboratory, January 2009.

[2] Anshul Gupta,"Introduction to Parallel Computing", IBM T.J. Watson Research Center, Yorktown Heights, 2003.

[3] George Karypis,"Parallel Algorithms and Applications", University of Minnesota, Minneapolis, March 2012.

[4] George Mozdzynski, "Concepts of Parallel Computing",European Centre for Medium-Range Weather Forecasts, March 2012.

[5] S. Salvini, Unlocking the Power of OpenMP, Invited lecture at 5th European Workshop on OpenMP (EWOMP '03), September 2003.

[6] Dheeraj Bhardwaj, "Parallel Computing- A Key to Performance", Department of Computer Science & Engineering, Indian Institute of Technology Delhi, August 2011.

[7] R. Parikh,"Accelerating quicksort on the intel Pentium 4 processor with hyper–threading technology",Software Community Intel, October 2007.

[8] Werner Backes, Sussane Wetzel, "A Parallel LLL using Posix Threads", Department of computer science, Stevens Institute of Technology.

[9] J. Balart, A. Duran, M. Gonz`alez, X. Martorell, E. Ayguad´e, and J. Labarta. Nanos Mercurium,"A Research Compiler for OpenMP", 6th European Workshopon OpenMP (EWOMP '04), pages 103–109, September 2004.

[10] D. an Mey,"Two OpenMP programming patterns", Proceedings of the Fifth European Workshop on OpenMP - EWOMP'03, September 2003.

9. AUTHOR'S PROFILE

Mr. Mukul Sharma is a Microsoft Certified Professional and Microsoft certified Technology Specialist. He has two years' experience in development with Microsoft Technologies. He is Pursuing his Master of Technology Degree in Computer Science. His area of research involves Ad Hoc Networks, Parallel Programming and Scalable Computing.

Mr. Pradeep Soni is a Microsoft Certified Technology Specialist. He has two years' experience in development with Microsoft Technologies. He is Pursuing his Master of Technology Degree in Computer Science. His area of research are Parallel Programming, Networking and DSA.