International Journal of Linguistics and Computational Applications (IJLCA) ISSN 2394-6385 (Print)
Volume 2, Issue 1, January - March 2015 ISSN 2394-6393 (Online)

Performance Optimization of Parallel Programming

Athira Anil K. E. #1, N. Arivazhagan*2

[1]Dept. of Information Technology, SRM University, Chennai, India.
athiraanilke@gmail.com
[2]Assistant Professor, Dept. of Information Technology, SRM University, Chennai, India.
arivazhagan.n@ktr.srmuniv.ac.in

Abstract— A well organized parallel application can accomplish better performance over sequential execution on existing and forthcoming parallel PC architecture. This paper depicts the issues that can limit the performance of parallel programs that restricts them from providing the expected performance improvement compared to sequential programming, test assessment of certain parallel application with sequential programs. Before depicting the exploratory assessment, this paper depicts a few systems pertinent of parallel programming. The assessment of parallel applications has been carried out by exploratory result assessment and execution estimations. Proper utilization of all available cores and resources, in addition to allocating balanced amount of load among all computing units, can lead to improved performance.

Keywords— *Multi-core, Multithreading, OpenMP, Parallel Programming, Performance Analysis, Processor Architecture.*

1. Introduction

Parallel processing is a way of dividing a large complex task among multiple processing units which operates simultaneously for achieving a common goal. The main purpose of parallelism is to decrease the execution time by maximum uses of CPU resources. Large instructions are decomposed into different parts and different smaller parts are distributed to different processors to be performed simultaneously. Its increases efficiency, speed and performance. Immense performance gain has been accomplished by doing operations in parallel. Speed up realization for speedier execution of a program requires essentially three aspects:

- Algorithm must include numerous independent operations
- Programming language must permit parallel operation specification and automatic recognition
- Hardware should have such an architecture which is capable of executing different operations at the same time.

Program must coordinate the needs of algorithm with the abilities of basic hardware. Execution of parallel application can be accomplished utilizing Multi-core innovation. The element spurred the outline of parallel calculation for multi-core framework is the performance. The performance of parallel algorithm is sensitive to number of cores available to the framework, core to core latencies, memory design, and synchronization costs.

2. Methodology

The sequential execution model is bad and wasteful in multi-core environment, while the normal parallel processing may be suitable. A standout amongst the most imperative numerical issues is solution of system of linear equations. Multi-core advancements backings multithreading to executing numerous threads in parallel and consequently the execution of the applications can be made improved. To accomplish the high performance in the application, we have to build up the right parallel algorithm, needs the hardware and the programming language like OpenMP. The OpenMP has the backing of multithreading. The system can be created so that all the processor can be occupied to enhance the performance.

Architectural advancements can enhance the measure of work performed per instruction. Technological enhancements can diminish the time obliged per instruction cycle. Numerous algorithms with inherent parallelism have computational intricacy than sequential. Algorithms suitable for single processors may not be useful for multi-processors. Really rebuilding of operation must be carried out to uncover the hidden parallelism. Hardware supported multithreading is one of the most adaptable procedure to hide latency, as it doesn't require any special software analysis and support. Besides, as it can be invoked dynamically it can deal with eccentric circumstances. Parallel programming supports fine grained parallelism.

Every parallel programming contains a parallel section and a serial section. Serial sections limit the parallel effectiveness. If there exists lot of serial computations in your program then there won’t be good speed up in your program. Serial work doesn’t allow perfect speedup. This is well explained by Amdahl’s law. Amdahl’s law is very essential rule for establishing theoretical basis for achieving maximum speed up of a parallel program.

Amdahl’s law places a strict limit on the speed up that can be understandable by using multiprocessors. The Amdahl’s law states that a small portion of the program which can’t be parallelized will limit the overall speedup available from parallelization. Amdahl’s Law can be shortly expressed by using the following equation, where f_s is the serial fraction of code, f_p is the parallel fraction of code, N is the number of processors:

Effect of multiple processors on runtime

$$t_p = (f_p / N + f_s)\, t_s$$

Effect of multiple processors on speedup

$$S = \frac{t_s}{t_p} = \frac{1}{(f_p / N + f_s)}$$

From the above speedup equation we can see that, the maximum potential speedup for a parallel program depends on how much a program can be parallelized.

Representation of Amdahl's law:

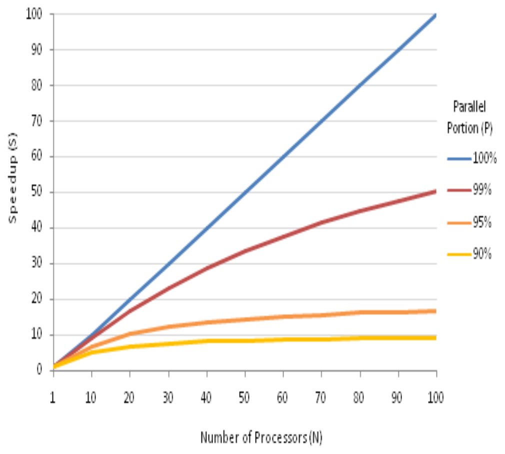

Fig. 1: Representation of Amdahl's law

But, in reality the expected result is not obtained because of degradation in performance due to various factors. The most important performance degradation contributor is communication.

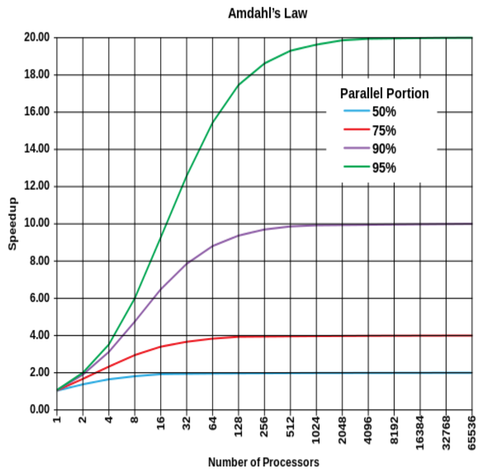

Fig.2: Reality of Amdahl's law

Various factors can limit the performance of parallel programs:

- Serial time can dominate
  - ❖ Poor Single Processor Performance
  - ❖ Typically due to Memory Performance.
- Parallel Overhead
  - ❖ Synchronization and Communication overhead
- Improper load balancing
  - ❖ Differing Amounts of Work Assigned to Processors
  - ❖ Different Speed of Processor
- Managing complex data dependency is Non-Trivial
- Managing data access conflicts without full access control
- Managing multiple core executions driving one insane
- Lack of Parallel library for general use
- Interacting With Hardware
- Legacy software is often used as the entry point to parallelization

Solutions for Load Imbalance

- ❖ Better Initial Assignment of Tasks
- ❖ Dynamic Load Balancing of Tasks

We need to consider whether the CPU requirements will justify parallelization to be done. Moreover check whether the code is to be used just once or more. This is necessary because parallel programming has a steep learning curve and is effort-intensive. Parallel computing environments are unstable and unpredictable. They don't even respond to the serial debugging and tuning techniques. Moreover, they may not yield a result you expect in spite of the large amount of time you invest on them.

3. Experimental Result Evaluation & Performance

Performance is the primary concern on parallel programs. In parallel applications, we expect the better performance than the traditional sequential programming. In general sense the expected performance depends on the available computer architecture.

3.1 System of linear equations

Number of Equations (n)	Execution time of sequential (in seconds)	Execution time of parallel (in seconds)
1000	3	3
2000	22	27
3000	85.999	65.528
4000	211	152.0882

Table 1: Tabulation of Performance Calculation of Linear Equations

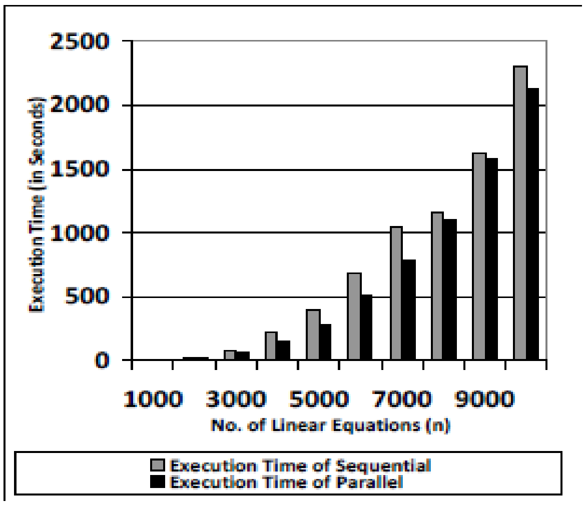

Fig. 3: Performance graph

Though in most cases we can see parallel execution in producing very less execution time compared to parallel, in when number equations were 2000 serial execution was faster than parallel which is contradictory to our assumption. This indicates the chances of exceptional behaviour at times which is unappreciable.

3.2 *Image Convolutions*

The equation for image convolution is given by

$$Out(i,j)=\sum_{m=0}^{M-1}\sum_{n=0}^{N-1}In(m,n)Mask(i-m,j-n)$$

Where, In is the input image, Mask is the convolution mask, and Out is the output image. The dimension of the image is M x N

The convolution algorithm will generate results that are greater than the range of original values of the input image. For this, scaling operation is performed to restore the result to same gray level range of original picture.

Program	Sequential	OpenMP	OpenCL
Time (in Sec)	0.51	0.05	0.96

Table 2: Time elapsed in image convolution

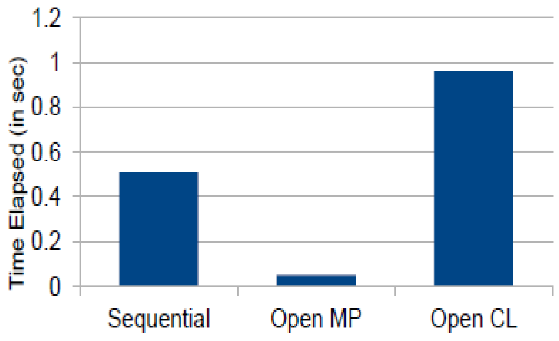

Fig. 4: Image convolution

Performance of OpenMP is better compared to the OpenCL, as evident from the figure. The speed-up achieved is (Seq/MP) = 0.51/0.05 = 10.2 whereas no speed-up is achieved w.r.t OpenCL as (Seq/CL) = 0.05/0.96 = 0.53. As a result, OpenMP is much faster compared to OpenCL, as OpenCL is busy in doing background of kernel creation and other things, than actual execution. The actual GPU device execution time can be found by profiling [14] the gpu device which will be very less as compared to OpenMP.

3.2 *String Reversal*

For the comparison purpose, we have taken a string reversal problem. We have considered a huge file in mega-bytes and tried to reverse it using OpenCL.

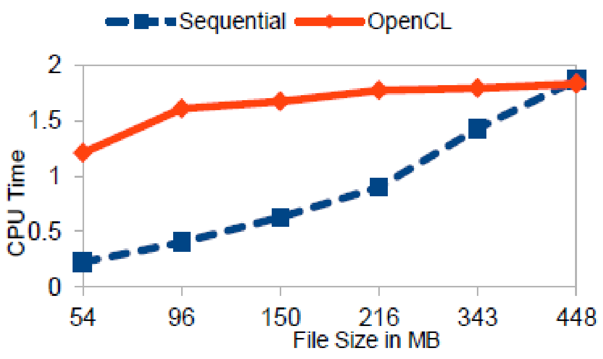

Fig.5: String Reversal performance comparison graph

File Size	Sequential	OpenCL
50 MB	0.22	1.23
94MB	0.44	1.68
155MB	0.58	1.64
216MB	0.91	1.76

Table 3: Tabulation of string Reversal

String reversal problem is straight forward. Just read the entire file and start copying values from end of the file. As there are no dependencies in this operation, we have not considered OpenMP Programming model. Even if we take OpenMP into consideration, performance will be same as reversing of read string falls in critical section. From Figure and Table, it can be concluded that, OpenCL Programming model is not suitable for this kind of applications.

All these test analysis proves that performance of parallel programs largely depend on the problem and the underlying architecture. Diverse machine architectures require different algorithms for productive resource use and operation. The type of parallelism relies upon the kind of architecture. Henceforth new parallel algorithms have to be designed. It is convenient to consider the parallelism in the algorithm instead of finding diverse parallel programs for different architectures. Pipelining and information parallelism are the two imperative procedures that assistance to enhance concurrency.

4. Discussion

As indicated by a few analysis, parallelism shows better performance over conventional sequential execution. In any cases sometimes the parallel loops meets expectations slower then sequential execution. In parallel execution there is so much unpredictability than in sequential. Alongside the accomplishment of great execution there are some terrible encounters also. Some of the issues occurred that if properly taken care to enhance the execution of parallel applications are:

4.1 Inappropriate employments of threading

Creating, pulverizing and scheduling between threads are cost variable. Threading utilize the CPU resources. Due to inappropriate usage, at some point, multithreading will not show the better execution. In the event where there is substantial I/O holding up then numerous worker thread can't work all the while.

4.2 Over utilization of threads

Over utilization of threads over CPU resources can sometimes ease off the execution. The profits of parallelization rely on the number of processors.

4.3 Unbalanced workload

Suppose the framework has enough parallel power, but total number of data or work is not all that high then each CPU won't have such a great amount of work to do. For vast workload the time of synchronization between threads is unimportant. However for small amount of data the synchronization expense of CPU can be so costly (moderately) than the primary execution. So the performance may be down.

4.3 Proper CPU utilize and working framework reliance

At long last, our applications on a working framework are not ready to 100% utilization of the CPU assets. It is the operating system which is capable of scheduling distinctive current threads in a framework. At the point when our program makes a few threads for execution in the meantime there may be some different threads in line of operating system. So it is calm difficult to captivate the CPU asset 100%.

Hypothetically the velocity up of execution in parallel ought to be linear. But all things considered we don't get it. Above talked about issues and other minor issues we don't get the pace up as direct. It is essential to perceive that the overhead from a given number of threads is an impression of the effectiveness with which threading has been applied to the given software (however it can never be equivalent to zero). When threading has been brought into the application, the tuning methodology must recognize bottlenecks that speak to threading overhead.

Most components of threading overhead fall into the following classifications:

Thread creation/destructin: Thread pooling (making reusable threads) reduces the need to make and resign worker threads.

Synchronization and lock administration: Threads lock the critical section of the code to shield data they are utilizing from being overwritten by different threads in a manner that could create unexpected results. Lock forces threads to hold on for one another to release the lock in order to access the critical section of the code.

Load imbalance: Consider the compelling situation where an application generates two threads and one performs substantial computation, while the other just composes the results of that processing to the screen. The second thread would be idle a great part of the time, diminishing the application's general productivity. Such issues, which are often very subtle, are critical factors in threaded application tuning.

As the quantity of processor cores accessible to the workload expands, so should the quantity of threads. More number of threads can build the unpredictability and complexity effects of each of the above overheads, which in turn brings down the effectiveness of the general code on an every core premise. This impact is integral for programming designers to incorporate great threading practices into their products for the anticipated increments in the quantity of cores per processor.

5. Conclusion

Put in the context of real-world contemplations about the overheads connected with programming multi-threading, they enlighten the potential outcomes that multi-core hardware bears individual applications that have been appropriately parallelized. The performance of parallel algorithm is sensitive to number of cores available to the framework, core to core latencies, memory design, and synchronization costs. The software development tools must conceptual these varieties so that the software performance keeps on gaining the profits of the Moore's law. Best practices and instruments from various renowned computer hardware design vendors are a key method for empowering designers to fabricate robust and adaptable threading into their applications. Various software development products are planned on account of threading, including libraries of pre-threaded capacities that lighten programming troubles, and in addition apparatuses that help engineers recognize threading errors and execution issues, accelerating the advancement process and expanding the nature of threaded applications.

It is an awesome test for us to beat the issue of parallel computing making proficient parallel applications in multiprocessor environment and tackle those world complex issues in an extremely effective manner. Performance is a principle issue of parallel applications. The learning of proper partitioning, staying away from over utilization, legitimate load balancing, proper memory sharing and so on makes your parallel application execution high. In this paper we have attempted to quantify the execution of parallel applications with distinctive parallelism criteria. On the premise of our exploratory result we have examined some imperative execution issues also. So we surmise that our Paper will be useful for the educated community as well as experts for further research or making huge parallel applications in a commonsense field. This Paper work will help as an establishment for further research for tackling the forthcoming complex issues on the planet.

Acknowledgement

This experimental paper work is developed as a result of full support and assistance of my guide Assistant Professor (Sr.G) N. Arivazhagan of SRM University. I express my sincere gratitude to him for his guidance. I also thank God Almighty, my family, other instructors for giving me support for doing the work.

References

[1] Posix threads programming. https://computing. llnl.gov/tutorials/pthreads.

[2] [2] G. J. Barbara Chapman and R. van der Pas. *Using OpenMP: Portable Shared Memory Parallel Programming*. The MIT Press, 2007.

[3] [3] M. Curtis-Maury, J. Dzierwa, C. D. Antonopoulos, and D. S. Nikolopoulos. Online power-performance adaptation of multithreaded programs using hardware event-based prediction. In *ICS*, pages 157–166, 2006.

[4] [4] M. Frigo, C. E. Leiserson, and K. H. Randall. The implementation of the Cilk-5 multithreaded language. In *PLDI*, pages 212–223, 1998.

[5] [5] J. Lee, H.Wu, M. Ravichandran, and N. Clark. Thread tailor: Dynamically weaving threads together for efficient, adaptive parallel applications.

[6] [6] Girbal, S., N. Vasilache, C. Bastoul, A. Cohen, D. Parello,M. Sigler, and O. Temam (2006). Semi-automatic composition of loop transformations for deep parallelismand memory hierarchies. International Journal of Parallel Programming, 34, 261–317.

[7] https://computing.llnl.gov/tutorials/parallel_comp/

[8] https://www2.cisl.ucar.edu/docs/parallel_concepts

[9] https://computing.llnl.gov/tutorials/openMP/

Athira Anil K.E., currently pursuing MTech in Department of Information Technology from SRM University, Chennai. I hold a Bachelor Degree in Information Technology from KMCT College of Engineering (Under Calicut University).

N.Arivazhagan is Asst.Professor of Department of information technology, SRM University, Kattankulathur. He has been serving more than 25 years of teaching and 1 year industry experience. He is doing research in Content Based Image Retrieval. He holds a M.S Degree from Birla Institute of Technology, Pilani and M.B.A from Madurai Kamaraj University.