International Journal of Computer Science and Software Engineering (IJCSSE), Volume 6, Issue 12, December 2017
ISSN (Online): 2409-4285 www.IJCSSE.org Page: 315-322

Parallel Computing Models and Analysis of OpenMP Optimization on Intel i7 and Xeon Processors

Kajal Chauhan[1], Dr. C. K. Bhensdadia[2] and Dr. M. B. Potdar[3]

[1] M.Tech Student, Dharmsinh Desai University, Nadiad-387001, Gujarat, India

[2] Head of Dept. of Computer Engineering, Dharmsinh Desai University, Nadiad-387001, Gujarat, India

[3] Project Director, Bhaskaracharya Institute for Space Applications and Geo-Informatics, Gandhinagar 382007, India

[1]okajalchauhan261@gmail.com, [2]ckbhensdadia@ddu.ac.in, [3]mbpotdar11@gmail.com

ABSTRACT

With the increasing data sizes and complexity of algorithms, and dead lock reached in processor clock frequency due to power constraints, multi core and many core CPUs and GPUs have been developed for parallel computing. This has become an inevitable approach for high volume data processing such as image processing. There have been several APIS for parallel processing developed with added merits and potentials, such as OpenACC, OpenMP, OpenCL and CUDA. Among these, the CUDA is implementable on NDVIDIA's GP-GPUs. Whereas others are implementable on multicore and many core CPUs and GPUs, include Intel Xeon Phi co-processors. Here we review both the hardware and software architectures of the devices and API. Then, compare the performance of OpenMP 2.x API when used on Intel Quad Core i7 (8 threads) processor with dual Intel Xeon 12 core (48 threads) CPUs by optimizing an image processing code on clustering in multispectral feature space using remote sensing data. The maximum speedup 5x is achieved on Intel i7 core CPU and speedup of 13x is achieved on Intel Xeon CPU by invoking dynamic scheduling when number of threads deployed are large. Minimum and maximum stack size required for different number of threads are also explored.

Keywords: *Intel i7, Xeon, OpenACC, OpenMP, OpenCL, CUDA, Image Processing, K-Means Clustering Algorithm, Code Optimization.*

1. INTRODUCTION

The Remote Sensing applications and many other applications like Medical imaging, Multimedia Technology and advanced and fast graphics used in gaming software etc. deal with large volume of data which require time effective parallel data processing algorithms and techniques. Now a day, the high speed computers are available which contains many core and multi core processors and coprocessors with high configuration for parallel computation. Parallel Computing has become an inevitable approach for high volume data processing such as image processing and also increasing demand of real time processing of images. Older approaches of multi core programming have been deprecated and hence there is a need to develop newer approaches that drastically increase the speed and performance.

There are different types of parallelism to achieve parallel computing in terms of software. First one is the instruction level of parallelism, which extracts the parallelism from a single instruction stream working on a single stream of data which provide low level of parallelism. Second is the processor level parallelism supporting more than one processor used for highly parallel application in which overall application is divided into subtasks and then computed on multi-processor simultaneously. Due to this, an application can utilize all the processors boosting the application performance. While there are high performance computers which contain large number of cores and multi-processors that can be simultaneously used for computing to get high performance and faster speed. For this, parallel application need to execute on many core and multi-processor architecture and require programming models that automatically scale with the number of processors or cores available and also provide synchronization between them.

1.1 Parallel Processing on CPU/GPU

In Remote sensing image processing, large number of images are processed and repetitive operations are performed on pixels using SIMD execution model on multi-processor in parallel to get better performance. There are many hardware and software approaches for

exploiting high level parallelism. Types of parallelism in hardware are Vector processor, SIMD instruction, GPUs and multi-processors. Whereas the Vector processor and SIMD instruction limit their parallelization features to a specific application. The GPU and multi-processor parallelization capability is dependent on programming model.

GPU which is usually used for highly parallel applications like computer graphics and image processing, their highly parallel structures make them more efficient than General Propose CPU.GPU has a massively parallel architecture consisting of many numbers of cores designed for handling multiple tasks simultaneously while CPU consists of a few cores optimized for sequential/serial processing. Instead of using capabilities of CPU and GPU alone, it is more beneficial to use both the CPU and GPU merged on single integrated circuit to increase better data exchange rates and low power utilization. The computing using both the CPU and the GPU co-processors together is called Heterogeneous Computing. To Implements this system, there are a few programming models like CUDA, OpenACC, OpenCL and OpenMP are proposed with their limitations and strengths.

1.2 Heterogeneous Computing

Heterogeneous computing is a system which contains more than one type of processor. They are multi-core or many core processor systems which gain performance not just by adding the similar processors but by adding the different type of co-processors to utilize specialized processing capabilities for handling tasks [2].

Now a day, data size is continuously increasing and techniques or methods have been developed for processing. However, the computation times goes beyond the time limit of their utility value. Therefore, the Architectures which are not only fast but also accelerate the performance are needed to be used. Hence the focus is on CPU and GPU combination. It is one of the heterogeneous Architecture where the best features of both can be combined to achieve even further computation gain and low power consumption. To process large number of Remote sensing images in heterogeneous system (CPU+GPU), it can provide high performance computation power and reducing computation time instead of using CPU and GPU alone because both CPU and GPU have distinct architecture feature.

In Heterogeneous Architecture of CPU+GPU based processing, the CPU is generally known as Host and GPU as a Device and they can be described as the master-slave relation. GPU device is managed by CPU. Multi-cores CPUs have cores up to few tens and Many-cores GPU have large number of cores up to a few thousands at least. Architecture uses Flynn's Single Program Multiple Data (SPMD) and new Single Program Multiple Task (SPMT) execution models. Due to very Different Architecture and programming models of CPU and GPU based heterogeneous computing, it presents several challenges like run time load on Processors(Pus)GPUs and achieving load balancing between CPU and GPU because the different number of cores among them. The distribution of type of work among the host and devices should be such that it utilizes the computational capabilities maximally. It is observed that different amount of work-divisions to CPU and GPU can lead to vastly different performance. Many experiments have been reported earlier for developing techniques for workload distribution, automatic scheduling of computation tasks over heterogeneous computing system(HCS)etc., and manage data placement to achieve better utilization of both processor powers to get fast processing, high performance, less overhead required for communication between CPU-GPU and less power consumption[3].

Different Programming languages can be used based on ease of programming, ability to write optimized code, ability to target multiple Pus and product from different vendors etc. The CUDA programming works only on NVIDIA's GPUs and uses the ACML (AMD Core Math Library). The OpenMP is most widely used due to its portability and relative ease of programming based on compiler directives. OpenMP works on both Intel Xeon Processor and Intel Xeon phi co-processor to achieve heterogeneous computing [3]. Due to the complexity of programming on GPU, porting a scientific application to the heterogeneous parallel system is a challenging task. Therefore, some latest research, such as MPtoStream, a compiler for extended OpenMP on AMD's High Performance GPUs has been developed. [4]

1.2.1 CUDA Programming

NVIDIA CUDA (Compute Unified Device Architecture) is an API for parallel computing. It is a Programming model provides multi-threaded Single Instruction Multiple Data (SIMD) model for implementing computation on GPUs. The CUDA takes advantage of massive computation power of GPU by utilizing large number of co-processor cores to the programmer. Its computing platform enables communicating fine-grained (thread level) and coarse-grained (block level) data and task parallelism using the extended C and C++ languages. In fine grained approach, after each instruction cycle, the switching between multithreads is done with lag in the execution of each thread. And in contrast, in coarse grained approach, the switching between the multithreads happens when the existing thread causes some long latency event. CUDA

Programmer can also choose a high-level programming language such as C, C++ or FORTRAN for parallel programming and these languages also support programming frameworks such as OpenACC and OpenCL which provide compiler directives used for parallel programming in both the homogeneous and heterogeneous programming with CPU and GPUs. The CUDA provides high level of parallelism and is the most prominent method for GPGPU acceleration, although it is only supported by NVidia GPU's architecture [1].

1.2.2 OpenCL Programming

The Open Computing Language, OpenCL, is a framework for writing parallel programs that execute across Heterogeneous platforms. It has been designed to be used not only with GPUs but also in other platforms like multi-core CPUs. Also, it extends support to AMD, NVidia and Intel GPUs equally. The main design goal of OpenCL is to use all computational resources of the system by efficient use of parallel programming model based on C99 extensions and it also defines a multilevel memory model. In parallel application, the OpenCL executes serial code on host (both mono core and multi-core CPUs) threads using task parallelism and parallel code in many device (GPU) threads using data parallelism across multiple processing element.
Like CUDA, the OpenCL is well suited for SPMD parallel design pattern. Now a day, CUDA and OpenCL are most prominent for GPGPU frameworks but CUDA is limited to NVidia Framework. Therefore, it does not cover wider range of types of applications as OpenCL [1].

1.2.3 OpenMP Programming

OpenMP is Programming Model stands for Open Specification for Multi-Processing. OpenMP is called directive based programming model for shared memory multi-processor using multithreading to develop parallel application in C, C++ and Fortran Programming Languages. The OpenMP Application Program Interface (API) provides a programmer with a simple and flexible interface for building portable parallel applications. It allows parallel execution on multi-core or many-core co-Processor by applying OpenMP Directives in user defined code region. It is the programmer responsibility to take advantages of thread parallelism using OpenMP Directives. In Directive based Programming model efforts are not much required to modify existing code written for homogeneous CPUs. It is easy to get optimized code with OpenMP Programming Model without the loss in performance.

Table 1: Comparison of OpenCL, OpenMP and CUDA [18]

	OpenCL	**OpenMP**	**CUDA**
Type	Heterogeneous CPU-GPU Computing	Heterogeneous CPU-GPU Computing	GPGPU Computing
Parallelism	Data Parallelism and Task Parallelism	Data Parallelism and Task Parallelism	Data Parallelism
Language or library	C99 and C++11 extensions	Directives for C,C++AND FORTRAN	C,C++ extensions
offloading	clEnqueue	Target device	Kernel <<<…>>>
Explicit data mapping and movement	bufferWrite function	Map (to/from/ tofrom/alloc)	cudaMemcpy function
Mutual Exclusion	atomic	Locks, critical, atomic, single.	atomic
Error Handling	exception	omp cancel	–

OpenMP Implements the fork-join model to achieve parallelism. The concept of master-slave relation where the master thread runs on host CPU and the slave threads run on GPU device(s) to accelerate the performance. Many extremely parallel code blocks can contain data dependency. To achieve parallelism, it is required to detect such dependency, classify its type of dependency and remove the dependency. It is inadequate to apply the OpenMP Directives for enhancing performance, several other factors are also affecting like parallel overhead and loop scheduling. In parallel programming applications, for the loop level parallelism the OpenMP is more efficient. Because of this, the utilization of GPU is more efficient in parallel computing [5, 7].

1.2.4 OpenACC Programming [20]

OpenACC, which stands for Open Accelerators, is one of the programming standards or API for parallel and heterogeneous computing developed by Cray, Caps, NVidia and PGI. The OpenACC is a directive-based programming model (like OpenMP) which is a collection of compiler directives to identify loops and regions of code in standard C,C++ and Fortran from host CPU to an attached many core devices. Because of this, it doesn't require more programming efforts to accelerators. The OpenACC directives provide

portability across multi-core CPUs as well as accelerators (GPUs) of various kinds, not just NVidia GPUs, including many-core processors like Intel Xeon phi chips from Intel.

Table 2: Comparison of directive based Programming Models[20].

	OpenACC	**OpenMP**
Target	Focused on Accelerating Computing	Focused on general purpose Computing
Approach	Descriptive	Prescriptive
Interoperability	Extensive interoperability	Limited interoperability
Mutual exclusion	Atomic	Locks, critical, atomic, single and master
Join	wait	Task wait
	More mature for accelerators	More mature for multi-core

With the OpenACC Directives, the programming efforts required for parallelization is higher in comparison to Cuda and OpenCL, while OpenMP and OpenACC are same in terms of effort required for Programming. But they are quite different in terms of implementation. OpenACC was developed by some of the OpenMP members due to which it got benefits of wide range accelerated systems. Some of the features of OpenMP such as data directives were first developed in OpenACC. The main difference between them is that the OpenACC targets scalable parallelism by specifying that a loop as a parallel loop. Now its compiler's responsibility to run this as fast as possible on the hardware. The OpenMP targets more general parallelism at task level, which is inherently not scalable and also, it is prescriptive which is very much directed by programmer it may be strength as well as weakness also. OpenMP have more features than OpenACC.

Though, the GPGPU provides high parallelism and fast computation speed for parallel applications, but its programming complexity presents a significant challenge for developer and has been greatly simplified by introducing improved library functions for better memory management [21]. Even though the CUDA Programming model was developed specifically for NVIDIA GPU, but programming GPU is still complex as compared to programming to General Purpose CPU and Intel Xeon phi co-processor/Processor using parallel programming model such as OpenMP. Hence, there has been research efforts on development of Techniques based on Compiler framework for automatic source to source translation of standard OpenMP application into CUDA-based GPU[7]. The OpenMP v4.0 [6] has Directives to program accelerator and new Directives to address issues like the management of a shared-memory many core accelerator. OpenMP v4.0 focuses on latest Intel Xeon phi co-processor and processor technologies. OpenMP v4.0 contains some key directives like “target” which compile and load for the execution onto a device and the “map” clause for selection of data item to be transferred to and from the device. The “target data” directive allows allocating and transfer data before the actual offload takes place and the “device” clause allows specifying the exact device to be used if more than one is present in the system. [18]

2. DYNAMIC SCHEDULING IN OPENMP

As many parallel programming techniques are available, we have reviewed their individual benefits and limitations which in turn affect how well they perform for different applications. Here, we used OpenMP API, because it is directive based portable programming. The compiler automatically ignores the directives if they do not support OpenMP. The directives are recognized and processed by a compiler; they also offer opportunities for compiler-based optimizations [5].

We evaluated the performance of K-Means clustering algorithm on Intel® Quad Core™ i7-4790@3.60 GHz processor and also on two Intel® Xeon® 12-core processor E5-2680 v3@ 2.50 having 16 GB Primary Memory. The Intel® Core™ i7-4790 being Quad Core Processor provides maximum 8 logical threads and Intel® Core™ i7-4790 have 2 processors with 12 cores each providing maximum 48 logical threads (2 logical threads per core). We have used the Microsoft Visual Studio ultimate 2012, which supports OpenMP 2.0 standard. We have used “OMP_GET_WTIME” function for calculating execution time per iteration in millisecond. The OpenMP is really beneficial if we use the compiler directives at right place in the application. It gives efficient performance and gain application performance.

We observed that the performance on both the processors execution time get reduced as we increase the number of threads. Some clauses of OpenMP like schedule(dynamic) are only effective when a large number of threads are deployed. In Fig 5 and 6, it is shown that the effect of number of threads on different directives. The execution time gets reduced from 12.8 milliseconds by using directives "#pragma omp parallel for" with schedule (dynamic) for 48 threads and it is

reduced to 1.4 millisecond for 8 threads. Therefore, here we can observe that it is not enough to increase number of threads but also it is required to insert appropriate OpenMP directives in parallel code to enhance performance. We have compared the k-means clustering algorithm on Intel® Core™ i7-4790@3.60 GHz Processors and Intel® Xeon® Processor E5-2680 v3@ 2.50 GHz processors with 16 GB Primary Memory in Fig 7 and Fig 8, respectively. We have taken different Numbers of threads and execution time per iteration in both processors. We have used time per iteration rather than total time, which depend on the number of iterations required for same set convergence criterion. The total time for same convergence depends on the initial cluster centres chosen. By utilizing all 48 logical threads available on two 12-Core Xeon processors, we have got 2x more speedup as compared to Intel® Core™ i7-4790 processor having 8 logical threads.

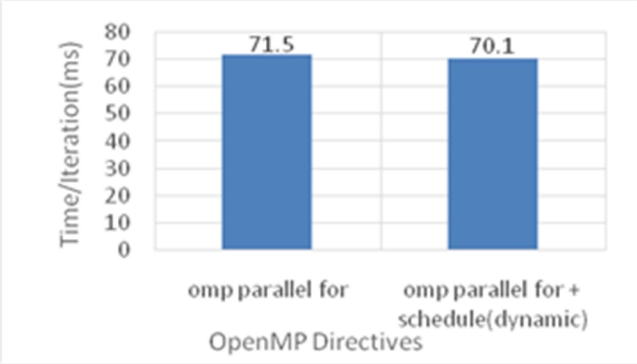

Fig 5.Comparision of OpenMP Directives for 8 Threads.

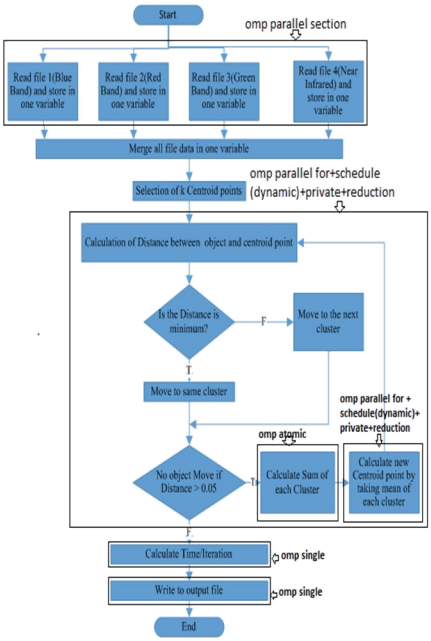

Fig. 4. Optimize the K-Means algorithm using OpenMP directives.

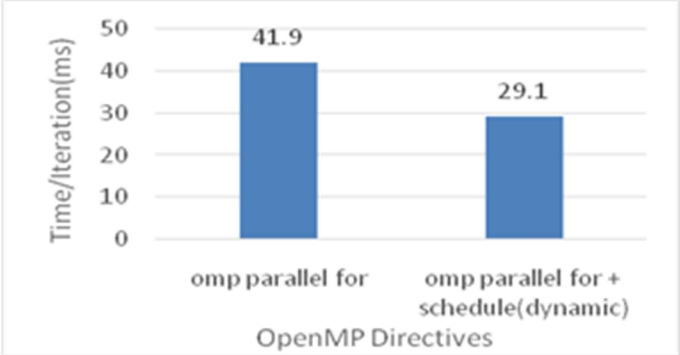

Fig. 6.Comparision of OpenMP Directives for 48 threads.

Here, We have also calculated the speedup factors for Intel® Quad Core™ i7-4790@3.60 Processor (maximum 8 logical threads) and Intel® two 12 Core Xeon® Processor E5-2680 v3@2.50 Processors (maximum 24 logical threads) with OpenMP and without OpenMP and compared in Fig 9 and Fig 10, respectively. In case of Intel® Core™ i7-4790@3.60 Processor with 8 threads, we have achieve maximum speedup of 4.3x while in case of Intel® Xeon® Processor E5-2680 v3@ 2.50 Processor with 48 threads, the speedup achieved is14.2x. This high speedup is achieved in spite of the Xeon processor is slower in speed compared to i7 processor. This is attributed mainly to the large number of threads available on system with Xeon processors and additionally due to the dynamic scheduling of the threads.

In the processors of achieving the optimization using OpenMP, the stack size allotment plays an important role. Depending on the size of data being handled, there is minimum size requirement and also maximum stack size required is determined by the number of threads. Maximum stack size possible is 2 GB with Windows OS. If the stack size lower than minimum, the entire data cannot be held in memory. If stack is very large, it is found that the system spends more time is data transfers between the primary memory (RAM) and the cache memory, resulting in the increasing in time of data processing and loss of advantage gained through OpenMP optimization. We have analyzed the allowable min and max stack sizes for achieving optimization.

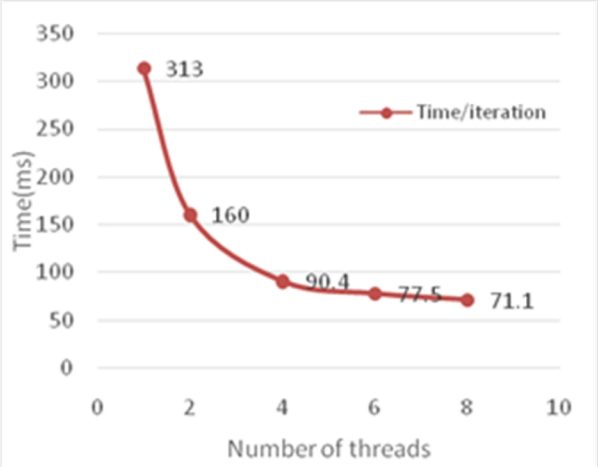

Fig. 7. Performance wrt Number of threads on Intel® nCore™I i7-4790 Processor

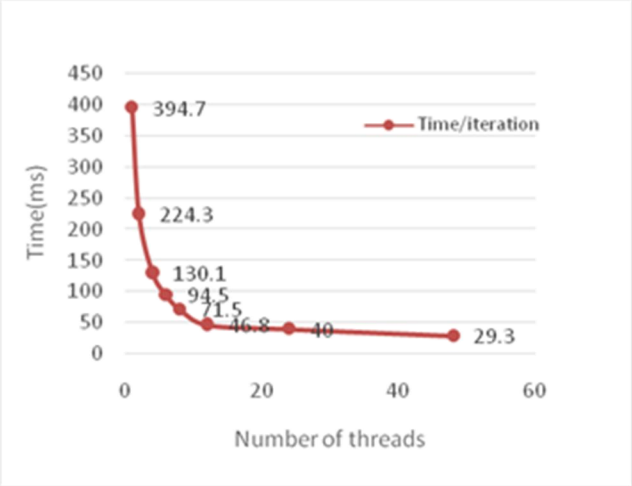

Fig. 8. Performance wrt Number of threads on Intel® Xeon® Processor E5-2680 v3

Fig.11 shows the variation of the minimum Stack size requirement with number of threads before and after the code optimization. We get the reduction in minimum stack size up to 2MB from 60 to 62 MB by optimizing the code using OpenMP. The minimum stack size requirements depends on the volume of the data being processed. With the increase volume of data, it increases linearly in proportion to the data volume. This dependence is shown in Fig. 12.

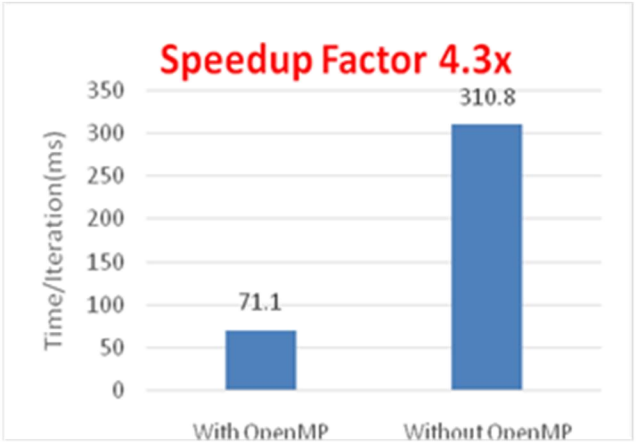

Fig. 9.Speedup on Intel® Core™ i7-4790@3.60 Processor

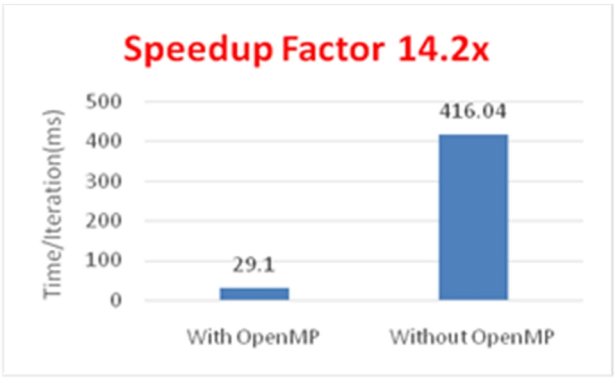

Fig. 10.Speedup on Intel® Xeon® Processor E5-2680 v3@ 2.50 Processor

The execution time per iteration remains near constant up to some specific stack size and thereafter, it increases in by a factor of more than 2. We have observed that specific stack size also varies with number of threads as it is shown in Fig 12. Beyond the Maximum Stack size for a given number of threads, the time shaved in code optimization is overwhelmed by CPU-RAM data transfer times. We observed that the maximum stack size permitted per thread is nearly inversely proportional to the number of threads. The product the max size and number of threads is nearly constant at 1.8 GB. One thread can use 1.8 GB (the maximum available stack size) and, as number of threads increase, the max stack size allowed decreases and ultimately 48 threads can optimize even with stack size of about 40 MB. The product of number of threads and max stack size remains constant at about 1.8GB. In our program executions for optimization, the stack size set at near mean value depending on the maximum number of threads used.

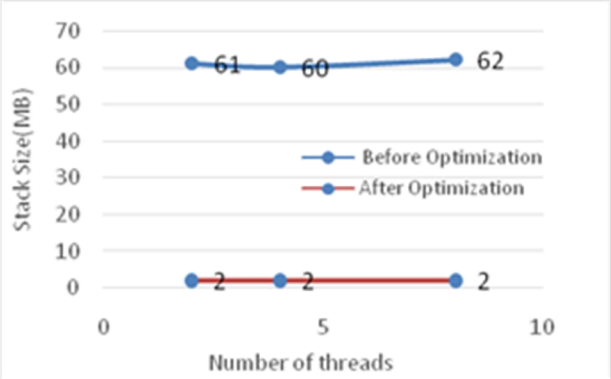

Fig. 11. Minimum Stack size requirement before and after code optimization for different number of threads.

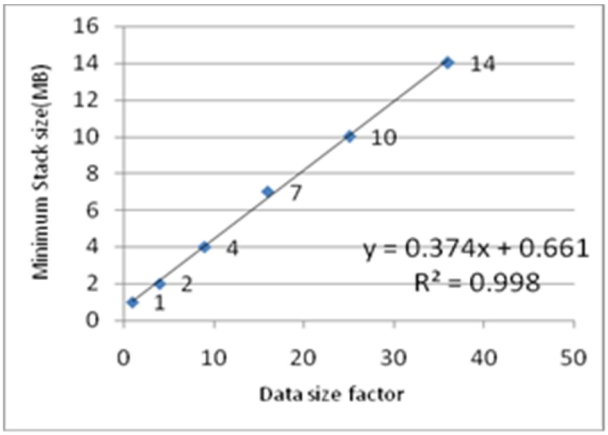

*Fig.12 Minimum Stack size requirement for different number of data size factors wrt to 256*256 image size.*

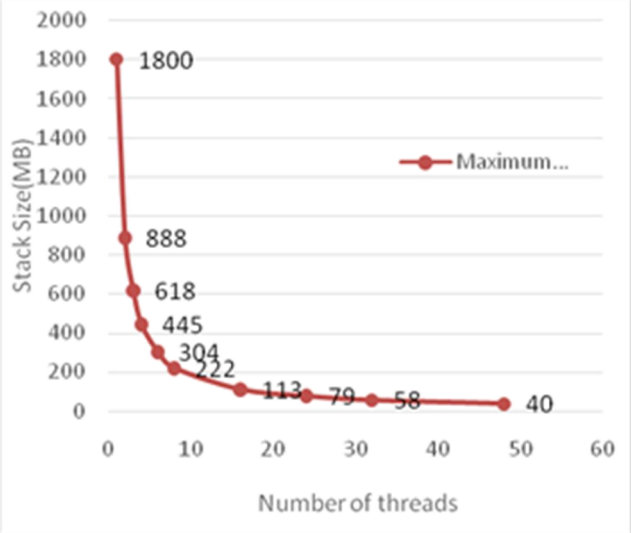

Fig. 13. Maximum Stack size variation with increasing number of threads.

We also evaluated the Stack size required to achieve proper optimization with OpenMP with respect to number of threads before and after code optimization. Minimum stack size depends on the volume of data being analyzed. The maximum stack size allowed for a set value of number of threads decreases with increasing number of threads. Total available stack memory of about 2GB is shared equally among the number threads invoked.

3. CONCLUSIONS

Efficient Parallel Programming Technique which are not only fast but also accelerate the performance that fully benefits from systems by utilizing both processors and co-processors is a challenging problem. In this Paper, we provide a comparison of OpenMP, CUDA, OpenCL and OpenACC Parallel Programming Techniques and we have research their individual benefits and limitations, which successively have an effect on however well they perform for various applications and Hardware. CUDA and OpenCL are more prominent for GPGPU Programming and OpenMP latest version 4.5 provided Offloading directives to achieve heterogeneous computing by utilizing both CPU + GPU in Intel Xeon phi coprocessor. We also done the code optimization of k-mean clustering algorithm using OpenMP 2.0 and analyzed it on Intel® Core™ i7-4790@3.60 Processor and Intel® Xeon® Processor E5-2680 v3@ 2.50 and the experimental result shows that OpenMP directive gives efficient result, if directives are inserted in right place and more number of threads are used. Our work also shows the Performance and Speedup of Processors Intel® Xeon® Processor E5-2680 v3@ 2.50 is high compare to Intel® Core™i7-4790@3.60.

ACKNOWLEDGMENTS

We are thankful to Shri T. P. Singh, Director, BISAG, for providing infrastructure and encouragement, and Special thanks Dr. C.K. Bhensdadia, Dharmsinh Desai University, Nadiad for permitting to carry out this project at BISAG.

REFERENCES

[1] Culler, David, et al., "LogP: Towards a realistic model of parallel computation." ACM Sigplan Notices. Vol. 28. No. 7. ACM, 1993.
[2] AMD-What is Heterogeneous Computing? http://developer.amd.com/resources/heterogeneous-computing/what-is-heterogeneous-computing/

[3] Mittal, Sparsh, and Jeffrey S. Vetter, "A survey of CPU-GPU heterogeneous computing techniques." ACM Computing Surveys (CSUR) 47.4 (2015): 69.
[4] Yang, XueJun, et al., "MPtostream: An OpenMP compiler for CPU-GPU heterogeneous parallel systems." Science China Information Sciences(2012): 1-11.
[5] Rohit Chandra, Leonardo Dagum, Dave Kohr, DrorMaydan, Jeff McDonald, Ramesh Menon, "Exploiting Loop-Level Parallelism," in Parallel Programming in OpenMP ,San Francisco, USA,2000
[6] Newburn, Chris J. et al., "Offload compiler runtime for the Intel® Xeon Phi coprocessor." Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), 2013 IEEE 27th International. IEEE, 2013.
[7] Lee, Seyong, Seung-Jai Min, and Rudolf Eigenmann, "OpenMP to GPGPU: a compiler framework for automatic translation and optimization" ,ACM Sigplan Notices 44.4 (2009): 101-110.
[8] Capotondi, Alessandro, and Andrea Marongiu, "On the effectiveness of OpenMP teams for cluster-based many-core accelerators", in High Performance Computing & Simulation (HPCS), 2016 International Conference on. IEEE, 2016.
[9] Intel Xeon Phi Product Family, https://www.intel.com/content/www/us/en/products/processors/xeon-phi/xeon-phi-processors.html
[10] Cramer, Tim, et al., "Openmp programming on Intel r Xeon phi tm coprocessors: An early performance comparison", in Proc. Many Core Appl. Res. Community (MARC) Symposium, 2012.
[11] Intel Corporation, "Intel RXeonPhiTM Coprocessor Instruction Set Architecture Reference Manual," September 2012, reference number 327364-001.
[12] Intel Pentium Processor, https://www.intel.com/content/www/us/en/products/processors/pentium.html
[13] Intel Xeon phi Programming Environment, https://software.intel.com/en-us/articles/intel-xeon-phi-programming-environment
[14] Kowalik, Janusz, Piotr Arłukowicz, and Erika Parsons. "Speeding Up Computers." arXiv preprint arXiv:1603.05487 (2016).
[15] Hybrid Computing – Coprocessors/Accelerators Power-Aware Computing – Performance of Applications Kernels https://www.cdac.in/index.aspx?id=pdf_xeon-phi-prog-overview-hypack
[16] CUDA vs. Phi: Phi Programming for CUDA Developers, http://www.drdobbs.com/parallel/cuda-vs-phi-phi-programming-for-cuda- dev/240144545
[17] James Jeffers James Reinders, "Introduction" in Intel Xeon Phi Coprocessor High Performance Programming,2013
[18] A comparison of heterogeneous andManycore Programming Model, https://www.hpcwire.com/2015/03/02/a-comparison-of-heterogeneous-and-manycore-programming-models/
[19] NVIDIA Tesla GPU Architecture AND CUDA Environment,https://code.msdn.microsoft.com/windows desktop/NVIDIA-GPU-Architecture-45c11e6d
[20] Is OpenACC The Best Thing To Happen To OpenMP?,https://www.nextplatform.com/2015/11/30/is-openacc-the-best-thing-to-happen-to-openmp/
[21] Wilt, Nicholas. The cuda handbook: A comprehensive guide to gpu programming. Pearson Education, 2013.