International Innovative Research Journal of Engineering and Technology
ISSN: 2456-1983 Vol: 2 No: 3 March 2017

Lateral Computing Models and Study of OpenMP Enhancement and Xeon Processors

R. Sujith
Department of Computer Science Engineering
Cihan University – Duhok, Kuridsitan Region Iraq

Abstract: With the expanding information sizes and intricacy of calculations, and halt came to in processor clock recurrence because of energy limitations, multi center and numerous center CPUs and GPUs have been produced for parallel registering. This has turned into an inescapable approach for high volume information handling, for example, picture preparing. There have been a few APIS for parallel preparing created with included benefits and possibilities, for example, OpenACC, OpenMP, OpenCL and CUDA. Among these, the CUDA is implementable on NDVIDIA's GP-GPUs. While others are implementable on multicore and numerous center CPUs and GPUs, incorporate Intel Xeon Phi co-processors. Here we audit both the equipment and programming structures of the gadgets and API. At that point, look at the execution of OpenMP 2.x API when utilized on Intel Quad Core i7 (8 strings) processor with double Intel Xeon 12 center (48 strings) CPUs by enhancing a picture preparing code on bunching in multispectral include space utilizing remote detecting information. The most extreme speedup 5x is accomplished on Intel i7 center CPU and speedup of 13x is accomplished on Intel Xeon CPU by conjuring dynamic booking when number of strings sent are expansive. Least and most extreme stack estimate required for various number of strings are likewise investigated.

Keywords: Intel i7, Xeon, OpenACC, OpenMP, OpenCL, CUDA, Image Processing, K-Means Clustering

1. INTRODUCTION

The Remote Sensing applications and numerous different applications like Medical imaging, Multimedia Technology and progressed and quick designs utilized as a part of gaming programming and so on manage extensive volume of information which require time viable parallel information preparing calculations and procedures. Presently a day, the rapid PCs are accessible which contains numerous center and multi center processors and coprocessors with high arrangement for parallel calculation. Parallel Computing has turned into an unavoidable approach for high volume information preparing, for example, picture handling and furthermore expanding interest of continuous preparing of pictures. More established methodologies of multi center programming have been deplored and thus there is a need to create more up to date approaches that radically increment the speed and execution.

There are diverse sorts of parallelism to accomplish parallel processing as far as programming. Initial one is the guideline level of parallelism, which removes the parallelism from a solitary direction stream taking a shot at a solitary stream of information which give low level of parallelism. Second is the processor level parallelism supporting in excess of one processor utilized for exceedingly parallel application in which general application is separated into subtasks and after that registered on multi-processor at the same time. Because of this, an application can use every one of the processors boosting the application execution. While there are superior PCs which contain expansive number of centers and multi-processors that can be at the same time utilized for figuring to get elite and speedier speed. For this, parallel application need to execute on numerous center and multi-processor engineering and require programming models that naturally scale with the quantity of processors or centers accessible and furthermore give synchronization between them.

1.1 Parallel Processing on CPU/GPU

In Remote detecting picture preparing, vast number of pictures are handled and tedious tasks are performed on pixels utilizing SIMD execution demonstrate on multi-processor in parallel to show signs of improvement execution. There are numerous equipment and programming approaches for abusing abnormal state parallelism. Sorts of parallelism in equipment are Vector processor, SIMD direction, GPUs and multi-processors. While the Vector processor and SIMD guideline confine their parallelization highlights to a particular application. The GPU and multi-processor parallelization capacity is subject to programming model.

GPU which is normally utilized for exceptionally parallel applications like PC illustrations and picture preparing, their very parallel structures make them more effective than General Propose CPU.GPU has an enormously parallel engineering comprising of numerous quantities of centers intended for taking care of various errands at the same time while CPU comprises of a couple of centers upgraded for consecutive/serial handling. Rather than utilizing capacities of CPU and GPU alone, it is more useful to utilize both the CPU and GPU converged on single incorporated circuit to expand better information trade rates and low power usage. The registering utilizing both the CPU and the GPU co-processors together is called Heterogeneous Computing. To Implements this framework, there are a couple of programming models like CUDA, OpenACC, OpenCL and OpenMP are proposed with their confinements and qualities.

1.2 Heterogeneous Computing

Heterogeneous figuring is a framework which contains in excess of one sort of processor. They are multi-center or numerous center processor frameworks which pick up execution not simply by including the comparable processors but rather by including the diverse sort of co-processors to use specific preparing capacities for taking care of undertakings [2].

Presently a day, information measure is constantly expanding and systems or strategies have been created for preparing. In any case, the calculation times goes past the time furthest reaches of their utility esteem. Along these lines, the Architectures which are quick as well as quicken the execution are should have been utilized. Consequently the attention is on CPU and GPU blend. It is one of the heterogeneous Architecture where the best highlights of both can be joined to accomplish significantly assist calculation pick up and low power utilization. To process expansive number of Remote detecting pictures in heterogeneous framework (CPU+GPU), it can give superior calculation power and lessening calculation time as opposed to utilizing CPU and GPU alone on the grounds that both CPU and GPU have unmistakable engineering highlight.

In Heterogeneous Architecture of CPU+GPU based handling, the CPU is by and large referred to as Host and GPU as a Device and they can be depicted as the ace slave connection. GPU gadget is overseen by CPU. Multi-centers CPUs have centers up to couple of tens and Many-centers GPU have vast number of centers up to a couple of thousands at any rate. Design utilizes Flynn's Single Program Multiple Data (SPMD) and new Single Program Multiple Task (SPMT) execution models. Because of altogether different Architecture and programming models of CPU and GPU based heterogeneous registering, it introduces a few difficulties like run time stack on Processors(Pus)GPUs and accomplishing load adjusting amongst CPU and GPU on the grounds that the diverse number of centers among them. The appropriation of kind of work among the host and gadgets ought to be to such an extent that it uses the computational capacities maximally. It is watched that diverse measure of work-divisions to CPU and GPU can prompt unfathomably extraordinary execution. Numerous tests have been accounted for before for creating strategies for workload conveyance, programmed booking of calculation errands over heterogeneous figuring system(HCS)etc., and oversee information situation to accomplish better use of both processor forces to get quick handling, elite, less overhead required for correspondence between CPU-GPU and less power consumption[3].

Distinctive Programming dialects can be utilized in view of simplicity of programming, capacity to compose advanced code, capacity to focus on numerous Pus and item from various sellers and so on. The CUDA programming works just on NVIDIA's GPUs and utilizations the ACML (AMD Core Math Library). The OpenMP is most generally utilized because of its convenientce and relative simplicity of programming in view of compiler orders. OpenMP chips away at both Intel Xeon Processor and Intel Xeon phi co-processor to accomplish heterogeneous registering [3]. Because of the multifaceted nature of programming on GPU, porting a logical application to the heterogeneous parallel framework is a testing assignment. Accordingly, some most recent research, for example, MPtoStream, a compiler for broadened OpenMP on AMD's High Performance GPUs has been created. [4]

1.2.1 CUDA Programming

NVIDIA CUDA (Compute Unified Device Architecture) is an API for parallel figuring. It is a Programming model gives multi-strung Single Instruction Multiple Data (SIMD) display for actualizing calculation on GPUs. The CUDA exploits huge calculation energy of GPU by using expansive number of co-processor centers to the software engineer. Its registering stage empowers conveying fine-grained (string level) and coarse-grained (square level) information and assignment parallelism utilizing the expanded C and C++ dialects. In fine grained approach, after every guideline cycle, the exchanging between multithreads is finished with slack in the execution of each string. What's more, interestingly, in coarse grained

approach, the exchanging between the multithreads happens when the current string causes some long inertness occasion. CUDA Programmer can likewise pick an abnormal state programming dialect, for example, C, C++ or FORTRAN for parallel programming and these dialects additionally bolster programming structures, for example, OpenACC and OpenCL which give compiler orders used to parallel programming in both the homogeneous and heterogeneous programming with CPU and GPUs. The CUDA gives abnormal state of parallelism and is the most noticeable strategy for GPGPU increasing speed, despite the fact that it is just upheld by NVidia GPU's design [1].

1.2.2 OpenCL Programming

The Open Computing Language, OpenCL, is a structure for composing parallel projects that execute crosswise over Heterogeneous stages. It has been intended to be utilized with GPUs as well as in different stages like multi-center CPUs. Likewise, it stretches out help to AMD, NVidia and Intel GPUs similarly. The fundamental outline objective of OpenCL is to utilize every computational asset of the framework by proficient utilization of parallel programming model in view of C99 augmentations and it additionally characterizes a multilevel memory show. In parallel application, the OpenCL executes serial code on have (both mono center and multi-center CPUs) strings utilizing errand parallelism and parallel code in numerous gadget (GPU) strings utilizing information parallelism over different handling component. Like CUDA, the OpenCL is appropriate for SPMD parallel outline design. Presently a day, CUDA and OpenCL are most unmistakable for GPGPU systems however CUDA is constrained to NVidia Framework. Consequently, it doesn't cover more extensive scope of kinds of utilizations as OpenCL [1].

1.2.3 OpenMP Programming

OpenMP is Programming Model stands for Open Specification for Multi-Processing. OpenMP is called mandate based programming model for shared memory multi-processor utilizing multithreading to create parallel application in C, C++ and Fortran Programming Languages. The OpenMP Application Program Interface (API) furnishes a software engineer with a basic and adaptable interface for building convenient parallel applications. It permits parallel execution on multi-center or many-center co-Processor by applying OpenMP Directives in client characterized code locale. It is the software engineer obligation to take focal points of string parallelism utilizing OpenMP Directives. In Directive based Programming model endeavors are very little required to alter existing code composed for homogeneous CPUs. It is anything but difficult to get advanced code with OpenMP Programming Model without the misfortune in execution.

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 accomplish parallelism. The idea of ace slave connection where the ace string keeps running on have CPU and the slave strings keep running on GPU device(s) to quicken the execution. Numerous to a great degree parallel code squares can contain information reliance. To accomplish parallelism, it is required to identify such reliance, group its sort of reliance and expel the reliance. It is insufficient to apply the OpenMP Directives for upgrading execution, a few different components are likewise influencing like parallel overhead and circle booking. In parallel programming applications, for the circle level parallelism the OpenMP is more proficient. Along these lines, the usage of GPU is more proficient in parallel registering [5, 7].

1.2.4 OpenACC Programming [20]

OpenACC, which remains for Open Accelerators, is one of the programming principles or API for parallel and heterogeneous figuring created by Cray, Caps, NVidia and PGI. The OpenACC is an order based programming model (like OpenMP) which is an accumulation of compiler orders to distinguish circles and locales of code in standard C,C++ and Fortran from have CPU to a joined numerous center gadgets. Along these lines, it doesn't require all the more programming endeavors to quickening agents. The OpenACC mandates give transportability crosswise over multi-center CPUs and also quickening agents (GPUs) of different sorts, not simply NVidia GPUs, including many-center 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 endeavors required for parallelization is higher in contrast with Cuda and OpenCL, while OpenMP and OpenACC are same as far as exertion required for Programming. Be that as it may, they are very extraordinary regarding usage. OpenACC was produced by a portion of the OpenMP individuals because of which it got advantages of wide range quickened frameworks. A portion of the highlights of OpenMP, for example, information orders were first created in OpenACC. The principle contrast between them is that the OpenACC targets versatile parallelism by indicating that a circle as a parallel circle. Presently its compiler's obligation to run this as quick as conceivable on the equipment. The OpenMP targets more broad parallelism at assignment level, which is innately not adaptable and furthermore, it is prescriptive which is particularly coordinated by software engineer it might be quality and in addition shortcoming moreover. OpenMP have a bigger number of highlights than OpenACC.

However, the GPGPU gives high parallelism and quick calculation speed for parallel applications, yet its programming multifaceted nature displays a noteworthy test for engineer and has been extraordinarily streamlined by presenting enhanced library capacities for better memory administration [21]. Despite the fact that the CUDA Programming model was produced particularly for NVIDIA GPU, yet programming GPU is as yet intricate when contrasted with programming to General Purpose CPU and Intel Xeon phi co-processor/Processor utilizing parallel programming model, for example, OpenMP. Subsequently, there has been inquire about endeavors on improvement of Techniques in view of Compiler system for programmed source to source interpretation of standard OpenMP application into CUDA-based GPU[7]. The OpenMP v4.0 [6] has Directives to program quickening agent and new Directives to address issues like the administration of a common memory numerous center quickening agent. OpenMP v4.0 centers around most recent Intel Xeon phi co-processor and processor advancements. OpenMP v4.0 contains some key mandates like "target" which order and load for the execution onto a gadget and the "guide" proviso for determination of information thing to be exchanged to and from the gadget. The "objective information" mandate permits apportioning and exchange information before the real offload happens and the "gadget" condition permits indicating the correct gadget to be utilized if more than one is available in the framework. [18]

2. DYNAMIC SCHEDULING IN OPENMP

The same number of parallel programming strategies are accessible, we have checked on their individual advantages and confinements which thus influence how well they perform for various applications. Here, we utilized OpenMP API, since it is order based convenient programming. The compiler naturally overlooks the mandates on the off chance that they don't bolster OpenMP. The mandates are perceived and prepared by a compiler; they likewise offer open doors for compiler-based improvements [5].

We assessed the execution of K-Means grouping calculation on Intel® Quad Core™ i7-4790@3.60 GHz processor and furthermore on two Intel® Xeon® 12-center processor E5-2680 v3@ 2.50 having 16 GB Primary Memory. The Intel® Core™ i7-4790 being Quad Core Processor gives most extreme 8 coherent strings and

International Innovative Research Journal of Engineering and Technology
ISSN: 2456-1983 Vol: 2 No: 3 March 2017

Intel® Core™ i7-4790 have 2 processors with 12 centers each giving greatest 48 sensible strings (2 intelligent strings for every center). We have utilized the Microsoft Visual Studio extreme 2012, which bolsters OpenMP 2.0 standard. We have utilized "OMP_GET_WTIME" work for ascertaining execution time per cycle in millisecond. The OpenMP is extremely helpful on the off chance that we utilize the compiler mandates at correct place in the application. It gives productive execution and pick up application execution.

We watched that the execution on both the processors execution time get decreased as we increment the quantity of strings. A few conditions of OpenMP like schedule(dynamic) are just compelling when countless are sent. In Fig 5 and 6, it is demonstrated that the impact of number of strings on various orders. The execution time gets decreased from 12.8 milliseconds by utilizing orders "#pragma omp parallel for" with plan (dynamic) for 48 strings 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.

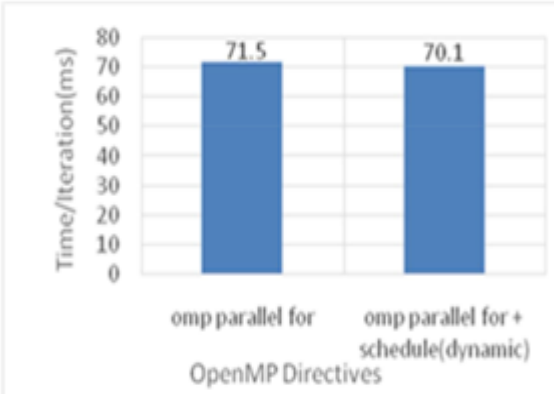

Figure 1. Comparision of OpenMP Directives for 8 Threads.

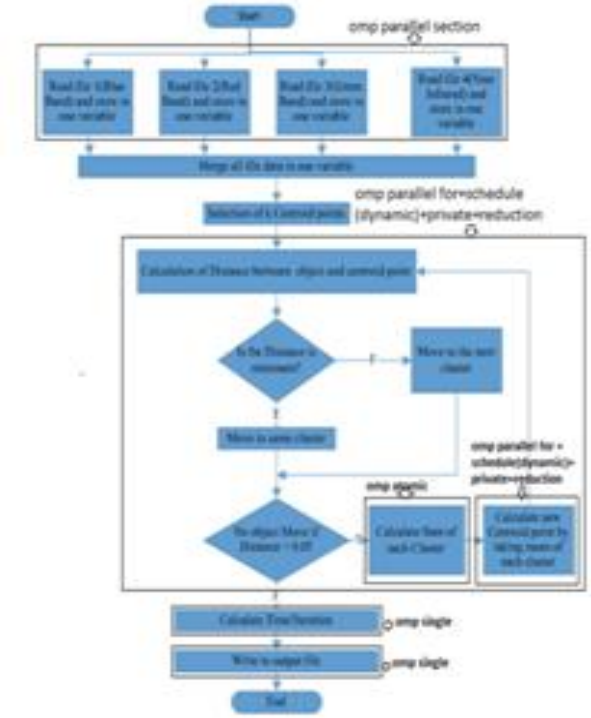

Figure 4. Optimize the K-Means algorithm using OpenMP directives.

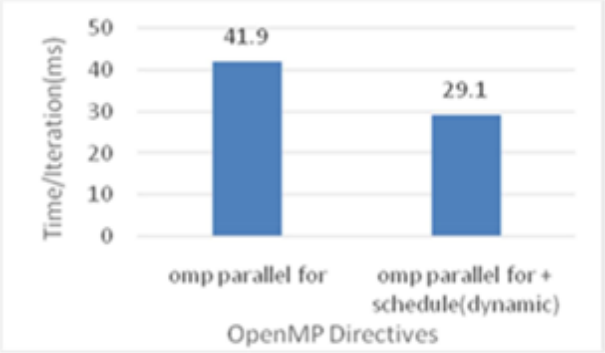

Figure 6. Comparision of OpenMP Directives for 48 threads

Here, We have likewise ascertained the speedup factors for Intel® Quad Core™ i7-4790@3.60 Processor (greatest 8 sensible strings) and Intel® two 12 Core Xeon® Processor E5-2680 v3@2.50 Processors (most extreme 24 coherent strings) with OpenMP and without OpenMP and thought about in Fig 9 and Fig 10, separately. If there should be an occurrence of Intel® Core™ i7-4790@3.60 Processor with 8 strings, we have accomplish most extreme speedup of 4.3x while if there should be an occurrence of Intel® Xeon® Processor E5-2680 v3@ 2.50 Processor with 48 strings, the speedup accomplished is14.2x. This high speedup is accomplished regardless of the Xeon processor is slower in speed

International Innovative Research Journal of Engineering and Technology
ISSN: 2456-1983 Vol: 2 No: 3 March 2017

contrasted with i7 processor. This is ascribed for the most part to the substantial number of strings accessible on framework with Xeon processors and furthermore because of the dynamic booking of the strings.

In the processors of accomplishing the streamlining utilizing OpenMP, the stack measure allocation assumes an imperative part. Contingent upon the extent of information being taken care of, there is least size prerequisite and furthermore greatest stack measure required is dictated by the quantity of strings. Greatest stack estimate conceivable is 2 GB with Windows OS. On the off chance that the stack estimate lower than least, the whole information can't be held in memory. On the off chance that stack is substantial, it is discovered that the framework invests more energy is information exchanges between the essential memory (RAM) and the store memory, bringing about the expanding in time of information handling and loss of preferred standpoint increased through OpenMP advancement. We have broke down the passable min and max stack sizes for accomplishing streamlining.

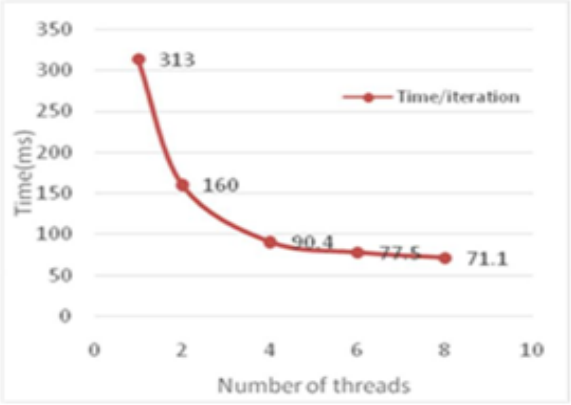

Figure 7. Performance wrt Number of threads on Intel® nCore™1 i7-4790 Processor

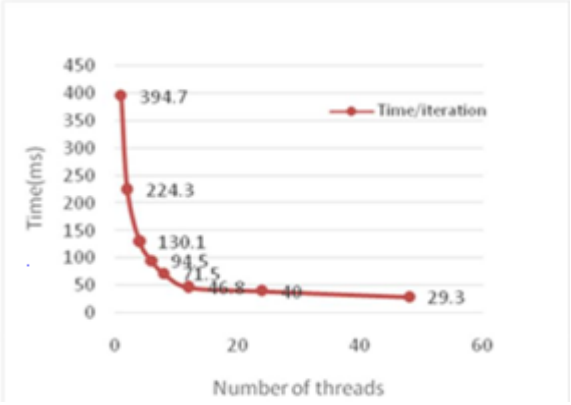

Figure 8. Performance wrt Number of threads on Intel® Xeon® Processor E5-2680 v3

Fig.11 demonstrates the variety of the base Stack measure prerequisite with number of strings when the code advancement. We get the diminishment in least stack examine to 2MB from 60 to 62 MB by advancing the code utilizing OpenMP. The base stack measure necessities relies upon the volume of the information being prepared. With the expansion volume of information, it increments directly in extent to the information volume. This reliance is appeared in Fig. 12.

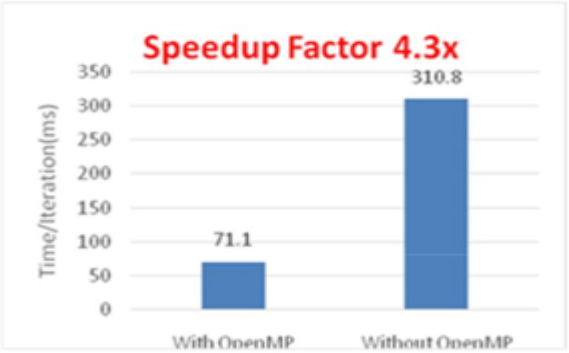

Figure 9. Speedup on intel®™ i7-4790@3.60 Proeessor

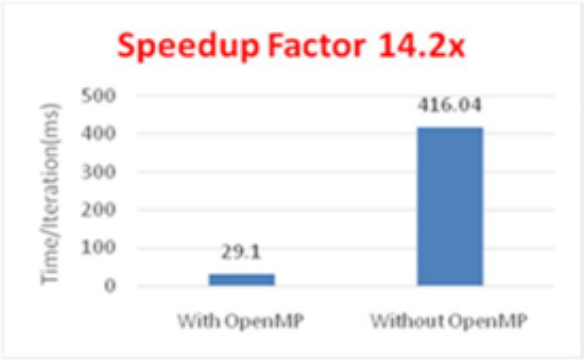

Figure 10. Speedup on intel® Xeon® Processor E5-2680v3@2.50 Processor

The execution time per emphasis stays close consistent up to some particular stack measure and from that point, it increments in by a factor of more than 2. We have watched that particular stack estimate likewise shifts with number of strings as it is appeared in Fig 12. Past the Maximum Stack measure for a given number of strings, the time shaved in code advancement is overpowered by CPU-RAM information exchange times. We watched that the greatest stack measure allowed per string is almost conversely corresponding to the quantity of strings. The item the maximum size and number of strings is about consistent at 1.8 GB. One string can utilize 1.8 GB (the most extreme accessible stack measure) and, as number of strings increment, the maximum stack estimate

permitted diminishes and at last 48 strings can upgrade even with stack size of around 40 MB. The result of number of strings and max stack measure stays consistent at around 1.8GB. In our program executions for streamlining, the stack estimate set at close mean esteem contingent upon the most extreme number of strings utilized.

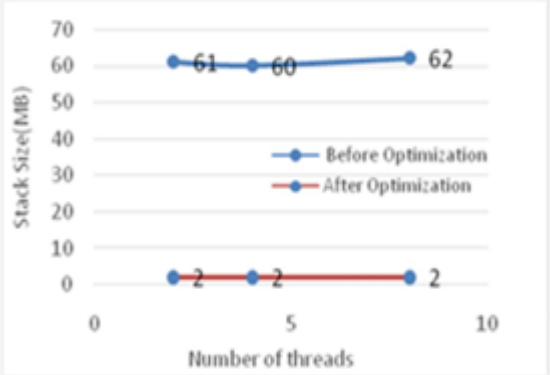

Figure 11. Minimum Stack size requirement before and after code optimization for different number of threads.

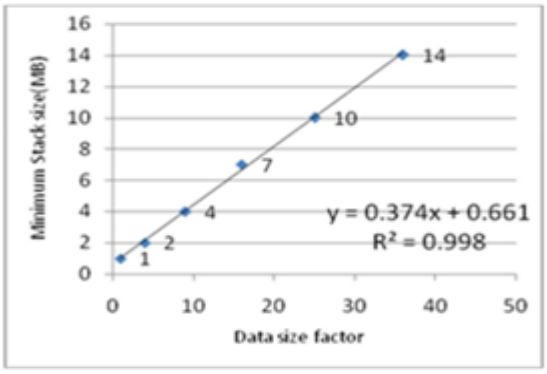

Figure 12. Minimum Stack size requirement for different number of data size factors wrt to 256*256 image size.

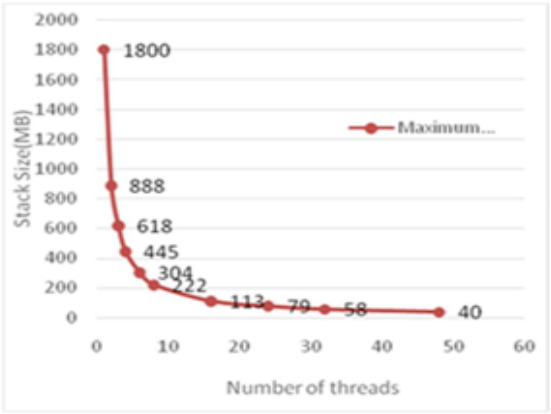

Figure 13. Maximum Stack size variation with increasing number of threads.

We additionally assessed the Stack measure required to accomplish appropriate improvement with OpenMP regarding number of strings when code enhancement. Least stack measure relies upon the volume of information being broke down. The greatest stack estimate considered a set estimation of number of strings diminishes with expanding number of strings. Add up to accessible stack memory of around 2GB is shared similarly among the number strings summoned.

3. CONCLUSIONS

Productive Parallel Programming Technique which are quick as well as quicken the execution that completely profits by frameworks by using the two processors and co-processors is a testing issue. In this Paper, we give an examination of OpenMP, CUDA, OpenCL and OpenACC Parallel Programming Techniques and we have look into their individual advantages and confinements, which progressively affect however well they perform for different applications and Hardware. CUDA and OpenCL are more unmistakable for GPGPU Programming and OpenMP most recent variant 4.5 gave Offloading orders to accomplish heterogeneous registering by using both CPU + GPU in Intel Xeon phi coprocessor. We additionally done the code advancement of k-mean bunching calculation utilizing OpenMP 2.0 and dissected it on Intel® Core™ i7-4790@3.60 Processor and Intel® Xeon® Processor E5-2680 v3@ 2.50 and the trial result demonstrates that OpenMP order gives proficient outcome, if orders are embedded in perfect place and more number of strings are utilized. Our work additionally demonstrates the Performance and Speedup of Processors Intel® Xeon® Processor E5-2680 v3@ 2.50 is high contrast with Intel® Core™i7-4790@3.60.

ACKNOWLEDGEMENT

We are grateful to Shri T. P. Singh, Director, BISAG, for giving framework and consolation, and Special expresses gratitude toward Dr. C.K. Bhensdadia, Dharmsinh Desai University, Nadiad for allowing to complete this venture 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 and Manycore Programming Model, https://www.hpcwire.com/2015/03/0/a-comparison-of-heterogeneous-and-manycoreprogramming-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

