DOI: http://dx.doi.org/10.26483/ijarcs.v9i2.5746

ISSN No. 0976-5697

Volume 9, No. 2, March-April 2018

International Journal of Advanced Research in Computer Science

RESEARCH PAPER

Available Online at www.ijarcs.info

EVALUATION OF OPENMP OPTIMIZATION IN HETEROGENEOUS COMPUTING MODE BY CODE OFFLOADING ON INTEL XEON PHI CO-PROCESSOR

Kajal Chauhan
M. Tech. Student, Dharmsinh Desai University,
Nadiad-387001, Gujarat, India

Dr. C. K. Bhensdadia
Head of Dept. of Computer Engineering
Dharmsinh Desai UniversityNadiad-387001, Gujarat, India

Dr. M. B. Potdar
Project Director, Bhaskaracharya Institute for Space
Applications and Geo-Informatics
Gandhinagar 382007, India

Abstract: As the computing needs are increasing, the utilization of compute powers of multi-processors and co-processors together is an active area of research. This paradigm is known as Heterogeneous computing. 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 used for parallel computing. It has become new approach for high volume data processing in the field of image processing. The present authors had earlier tested on the Intel Quad Core i7 processor with 8 threads and two Intel Xeon 12 core with 48 threads CPUs for optimization of K-Means clustering image processing code using remote sensing data. The speedup of 5x was achieved on Intel i7 core CPU and 13x was obtained on Intel Xeon CPU when dynamic scheduling as threads deployed were large. In continuation of the earlier studies, the present study analyses the Intel Xeon phi coprocessor 7120P(device) HPC accelerator performance with processor base frequency of 1.24 GHz along with OpenMP Parallel computing model. It is observed that the offloading will not give best result with small data size. To get the full benefits of offloading on Intel Xeon phi coprocessor, computation offloading with OpenMP utilizing both processor and coprocessor gains accelerations and increases the performance if communication overhead is less than the computation times which is highly application dependent.

Keywords: Key words: Intel i7, Xeon, Intel Xeon Phi, Code Offloading, OpenMP, Image Processing, K-Means Clustering, Code Optimization.

I. INTRODUCTION

Since the requirement and need of more and more compute power is increasing rapidly, there many new architectures are to fulfill this requirements. One of them is GP-GPU to fulfill this requirement. The GPU manufactured by NVIDIA mainly for gaming systems have found way into parallel computation as GP-GPUs. The GPGPU Provides high parallelism and fast computation speed for parallel applications, but its CUDA programming complexity presents a significant challenge for developer and has been greatly simplified by introducing improved library functions for better memory management [3]. Even though the CUDA Programming model was developed specifically for NVIDIA GPU, the heterogeneous programming of GP-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. The another architecture which can accomplish the requirement of accelerated computing is many integrated core (MIC) architecture of Intel Xeon Phi coprocessor (fig. 1). A program source code written for standard Intel® Xeon® processor(CPU) can be compiled and run on a Intel® MIC Products (Intel Xeon phi). The programming these cores can be with the OpenMP directives in standard C, C++, and

FORTRAN source code[4]. The newer version of OpenMP like OpenMP v4.0 [5] provides directives to program accelerators and also new directives to address the management of a shared-memory. OpenMP v4.0 focuses on latest Intel Xeon phi co-processor and processor technologies. OpenMP v4.0 contains some key directives like “target” which compiles and loads the executable 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 of device and transferring of data to it. Before the actual offload takes place, the “device” clause has provision of allowing a specific device if more than one device is present in the system. [6].

Three modes of offloading are shown in fig. 2. The most common Execution modes in heterogeneous environment is offloading a compute intensive portion of a code to the Device [7, 8]. The mode is known as native mode, wherein the entire code is uploaded on the device for execution .

Offload mode: In this mode, an application starts execution on a host and later some selected highly computationally intensive parallelizable portions of the code are offloaded (i.e. sent) to device(s) for the execution on coprocessors by using all the cores and resources on the device computing system. This mode is used when a program contains largely and highly parallel codes and the concerned data for processing on the device(s) are large in size. The data required for processing on device(s) by the offloaded program for computation is to be transferred from CPU to coprocessor(s) only once without any need of multiple transfers. In this model, the coprocessor acts as an accelerating device similar to GPU. The Offload is achieved

by using Offloading directive available in OpenMP v4.0 and later versions. Using this directive at the beginning of a code region, where parallel computation is accelerates the computation quite significantly.

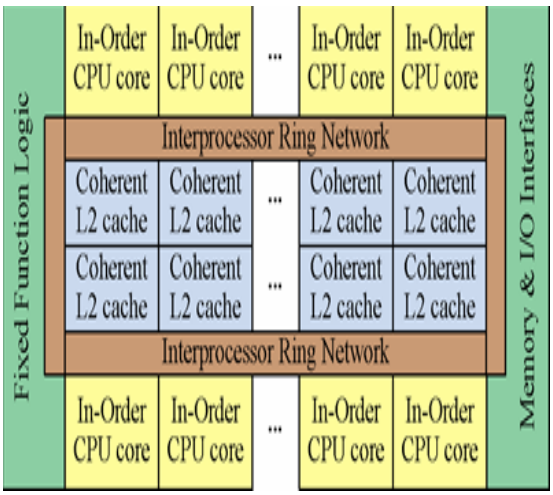

Fig. 1 High level overview of the Intel Xeon Phi Co-processor architecture [2].

This mode is hindered by the issue of the time required for transferring the related data from host memory system to the device(s) memory systems. The larger the volume of the data, more the time required for transfer. Also, the time overhead required for establishing link from host to device(s) is also quite significant.

The loss in time due to data transfer need to be compensated by the processing time of all computing elements of the device(s). In general, the processors on the GPU are slower than those of the host. The offloading to devices becomes economical only when the gain in the processing time even with slower processors surpasses the data transfer time by a large margin.

Native mode: In this mode, the entire application is made to run on coprocessor itself. The program code is compiled for "mmic" execution using a suitable compiler. The executable is transferred to the device. The user set default to the device and thus executes the code in the native mode. Hence, in this mode not all the platform computing power is used. The input data for processing has to be uploaded to the device memory separately. A major drawback is the smaller size of the coprocessor memory than the host processor RAM memory. Therefore, this mode is more beneficial when application contains high or massive amount of computations parallelism without the sequential or minor sequential components. Most of the computations involved in scientific research fall in this category of data processing types.

The OS running on the devices is usually is Linux. Using this native mode of execution is simple when the host is also running on the Linux OS. If the host is running on Windows OS, a secure connection to device need to be established. Such a situation is avoided if Knights landing Intel Xeon Phi processor is deployed in place of host processor(s).

Symmetric mode: In this case, the application program runs on both the host processor and coprocessor(s) with some workload sharing which is possible with Message Passing Interface (MPI). All available cores on host and devices are used. However, a programmer may face two challenges. First, balancing the workload among the different number of cores in CPU and GPU coprocessors. Second, the communication cost through MPI can be usually higher than the computation cost. The second can be minimized when the data when is large. The cost-benefit analysis is required to be carried out in this mode of operation.

II. INTEL XEON PHI COPROCESSOR

Intel Xeon Phi architecture is based on different hardware design and programming principles than its closest contender NVidia Tesla and AMD in HPC market used for acceleration of general purpose computing and highly parallel applications. The applications which benefit in performance with GPU should always benefit from Intel Xeon Phi coprocessor because of the same fundamentals of vectorization, SIMD implementation etc. The flexibility of an Intel Xeon Phi includes support for applications that can't be run on GPUs. Moreover, the efforts required for programming in Intel Xeon Phi is much less than that for CUDA (Table 1). CUDA programming requires specially writing of application kernels. Programs written in OpenMP are can be easily ported for execution on Intel Xeon Phi Coprocessor (Table 2). The Compute kernel can be easily ported to Intel Xeon Phi without much code changes. The efforts of porting applications to CUDA or OpenCL are usually much higher in than those required in case of OpenMP directive based programming model [10, 11].

The Intel® Xeon Phi™ coprocessor can be programmed with standard techniques like C/C++, Fortran using parallelization paradigms like OpenMP, Co-array Fortran, OpenCL and systems MPI, Intel Cilk, Intel TBB and the Intel Math Kernel Library (MKL). The OpenMP is enough to get better result when used with latest offload version of OpenMP v4.0/4.5 to get high performance [12].

Intel Xeon phi coprocessor (named Knights Corner) is powered by one or more processors which act as a host for coprocessor and each host has one or more number of coprocessor. The Intel Xeon Phi coprocessor is connected to an Intel Xeon processor through PCI Express bus. To boost the application performance, both the Intel Xeon processor and Intel Xeon phi coprocessor can be used. Intel Xeon phi coprocessor Architecture, as shown in Fig. 1, contains 61 cores running at 1.24 GHz Pentium cores [13] supporting maximum 4 threads per core. It also includes 32 vector registers with width of 512 bits. Various Execution model as explained earlier offload, native and symmetric have been developed and design that is used for the execution of application on Intel Xeon phi coprocessor in associated with host processor [7, 14].

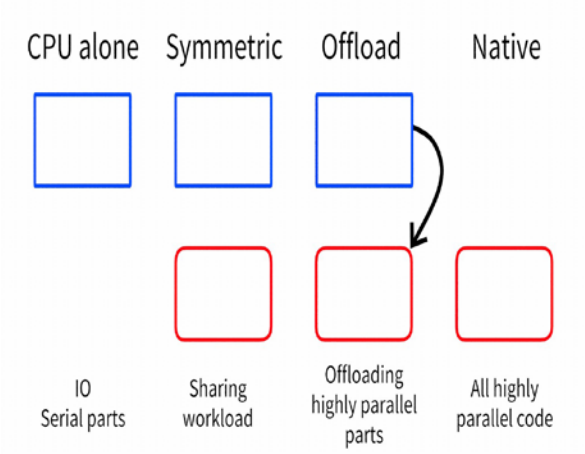

Fig. 2 Execution Model of CPU alone, Symmetric, offload and Native.

The 2nd Generation Intel Xeon phi Processor [5] (named Knights Lander) is the first Intel's bootable host processor that provides vectorization and massive parallelism for HPC applications. Its architecture is based on well-known Intel's standard shared memory architecture, which basically focuses on providing improvement in vector and scalar performance. It contains up to 72 cores and adopts new memory technology, MC-DRAM for large high bandwidth memory transfers and DDR for huge bulk memory transfers. The Knights Landing is upgradation for Knights corner user. The Applications which run are on Knights Corner can easily run on Knights Landing. The programming language that work for Intel Xeon processor and Knights Corner, viz. OpenMP, MPI and TBB, work equally well for Knights Landing. But GPU Programming like CUDA and OpenCL are not possible for this processor.

Intel Xeon phi coprocessor (named Knights Corner) is powered by one or more processors which act as a host for coprocessor and each host has one or more number of coprocessor. The Intel Xeon Phi coprocessor is connected to an Intel Xeon processor through PCI Express bus. For better performance of applications, both the Intel Xeon processor as host and Intel Xeon phi as coprocessor can be deployed. Intel Xeon phi coprocessor Architecture, (Fig. 1) contains 61 cores running at 1.24 GHz Pentium cores [13] supporting maximum 4 threads per core. It also includes 32 vector registers with width of 512 bits. Various Execution model as explained earlier offload, native and symmetric have been developed and design that is used for the execution of application on Intel Xeon phi coprocessor in associated with host processor [7, 14].

The 2nd Generation Intel Xeon phi Processor [5] (named Knights Lander) is the first Intel's bootable host processor that provides vectorization and massive parallelism for HPC applications. Its architecture is based on well-known Intel's standard shared memory architecture, which basically focuses on providing improvement in vector and scalar performance. It contains up to 72 cores and adopts new memory technology, MC-DRAM for large high bandwidth memory transfers and DDR for huge bulk memory transfers. The Knights Landing is upgradation for Knights corner user. The Applications which run are on Knights Corner can easily run on Knights Landing. The programming language that work for Intel Xeon processor and Knights Corner, viz. OpenMP, MPI and TBB, work equally well for Knights Landing. But GPU Programming like CUDA and OpenCL are not possible for this processor.

Table.1 Programming comparison of Intel Xeon Phi co-processor with CUDA Enabled Device [9]

Programming Approach	CUDA Enable Device	Intel Xeon Phi co-processor
Language such as C/C++ /Fortran etc	Only Through the offload programming mode. Many language can be accelerated only by calling CUDA, OpenCL or library methods	Both Native and offload mode but requires the use of a threading model like Pthreads or OpenMP
CUDA, OpenCL acceleration	On device as an offload accelerator	Offload model, OpenCL compiler support is coming. Technically possible for CUDA, but products such as CUDA-x86 do not currently generate code for Intel Xeon Phi coprocessor. Alternate possible path include (1) the CU2CL CUDA-to-OpenCL source translator,(2)LLVM translation and (3)manual translation
Directive-based programming	Via OpenACC as an external accelerator	Via OpenMP natively and in offload mode
Programming with libraries	Both native and offload mode	Both on-device and offload

III. OFFLOADING TO INTEL XEON PHI COPROCESSOR

Intel Xeon Phi coprocessor enables new OpenMP programming directive "offload" that offloads the computation from a host processor to Intel Xeon phi co-processor for parallel processing. Offloading advantage depends on factors such as (i) application characteristics such as the computation part must be higher than the communication for the offloaded portion, (ii) efficiency of host and coprocessor runtimes in transferring data and speed

of code invocation etc. Intel provides a user-level offload library called the "Intel Coprocessor Offload Infrastructure (COI)" for the services to create coprocessor-side process. It creates FIFO pipeline between host and coprocessor, moves data and uploads code, invokes code, manage memory buffer etc. The language pragmas is one of the programming model which provides compiler directives for offloading like "#pragma omp offload", for transfers of data to and from coprocessor for offloading code, with clauses "in, out, inout and nocopy"; of which the "inout" is used implicitly. For the synchronization of variables between host and coprocessor "#pragma offload target(mic)" directive is used to offload. The "#pragma offload_transfer target(mic)'' minimizes the data transfer allocation overhead on device. Therefore, it depends on the programmer how to enhance offload speedup[5]. For using offload directives, the following steps need to be followed:

1) Install Intel's MPSS i.e. Many core Platform Software Stack [15].
2) Set the environment for Intel MIC Architecture software [16] and install the driver successfully and start the coprocessor.
3) Install the Software Development tools [17] and install the latest version of Intel compiler.
4) Configure the Intel Parallel Studio XE with visual studio.
5) Enable all offload and OpenMP options in the project properties.

Table. 2 Comparison of NVDIA GPU with Intel Xeon Phi co-processor

	NVidia GPU	Intel Xeon Phi co-processor
Number of cores	2880(K40 TESLA)	72 (7290F)
Memory size	12GB(K40 TESLA)	384GB(7290F)
Parallelism	Data parallelism	Task Parallelism
Directives	OpenACC	OpenMP + Phi Directives
Tools	Intel Native Compiler	OpenCL
Native Programming Model	CUDA	Vector Intrinsic

IV. OBJECTIVES

The present authors reported in [1] the analysis of OpenMP Directives based optimization of K-Means clustering algorithm on Intel Core™1 i7-4790 3.6 GHz Quad core host processor with 8 logical threads and on twin Intel Xeon Processor E5-2680 v3 2.5 GHz host processors with 12 cores each and 48 logical threads. The Fig. 3(a, b) show the execution times on Intel i7 processors with 8 threads and on Intel Xeon processors with 48 threads, respectively. The speedup factor of 4.3 was obtained with Intel i7 Quad core (8 threads) host and 14.2 with Intel Xeon dual 12 core (48 threads) host. This present work extends the above work to include Intel Xeon Phi co-processor and offload compute intensive code to it and evaluate the advantage gained over the previous configurations. In Offloading we can use both Intel Xeon Processor on host and Intel Xeon Phi coprocessor by using OpenMP directives.

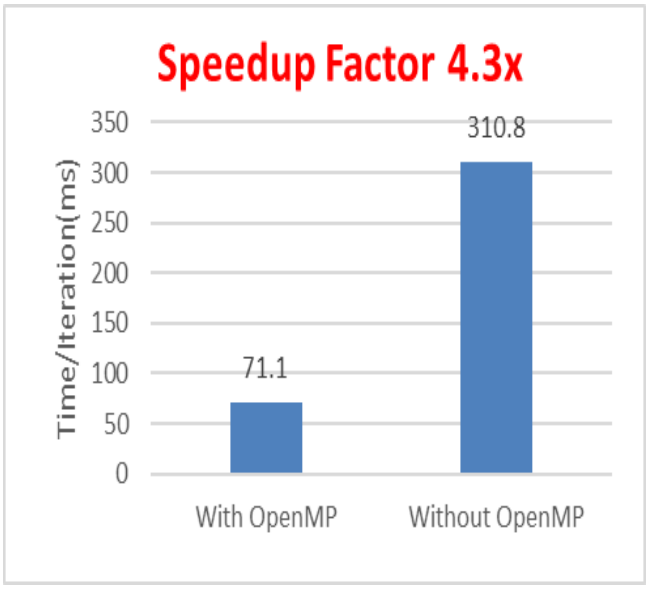

Fig 3a. Speedup on Intel® Core™ i7-4790@3.60

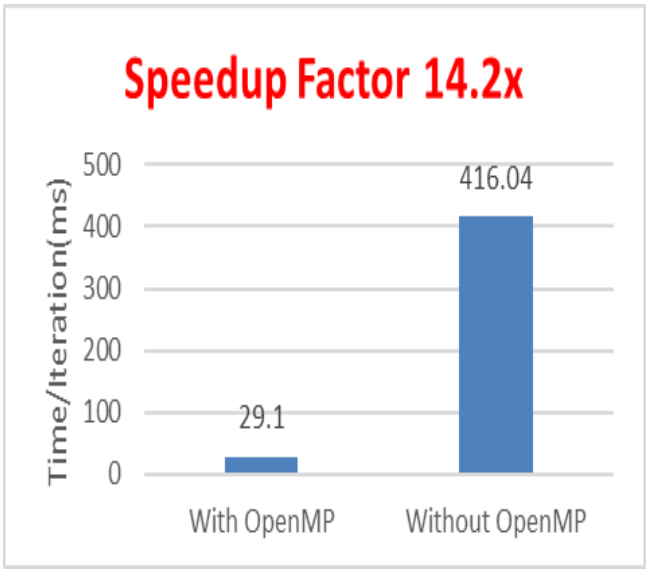

Fig 3b. Speedup on Intel® Xeon® Processor E5-2680 v3@ 2.50 Processor

V. DATA ANALYSIS AND RESULT

We have used the K-Means clustering code that does unsupervised classification of the remotely sensed multispectral data based on the clustering in the spectral feature space. Earlier, the results of the OpenMP optimization of this code on Intel i7 and Intel Xeon Hosts have been reported [1]. The optimization of this code in heterogeneous computing environment comprising of Intel

Xeon two 12 core 2.5 GHz CPUs having 16 GB primary memory as host and Intel Xeon Phi (Knights Corner) 61 cores with processor speed 1.24 GHz co-processor as device by offloading K-Means clustering algorithm is attempted. We have used OpenMPv4.0/v4.5 Directives. The Microsoft Visual Studio ultimate 2012 is integrated with Intel Parallel Studio XE 2017 which provides the Intel compiler C++ 17.

The Performance of OpenMP optimization of K-Means clustering algorithm using four band multispectral data acquired by Landsat 8 Thematic Mapper. The basic 4 band multispectral data of 512 lines by 512 columns were rescaled to sizes 256*256, 768*768, 1024*1024, 1280*1280, 1536*1536 and 1792*1792 lines and columns. The data volume of 1792x1792 images increased by a factor 49 compared to the size of 256x256 lines and columns.

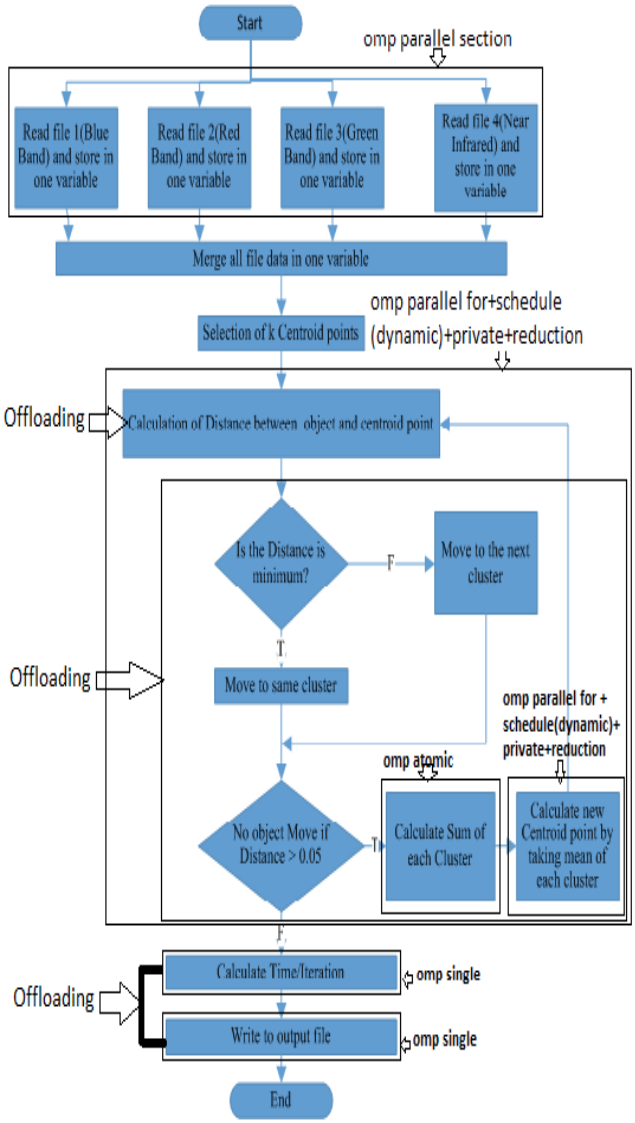

Fig. 4 Optimize the K-Means algorithm using OpenMP directives and apply offloading on Intel Xeon phi coprocessor.

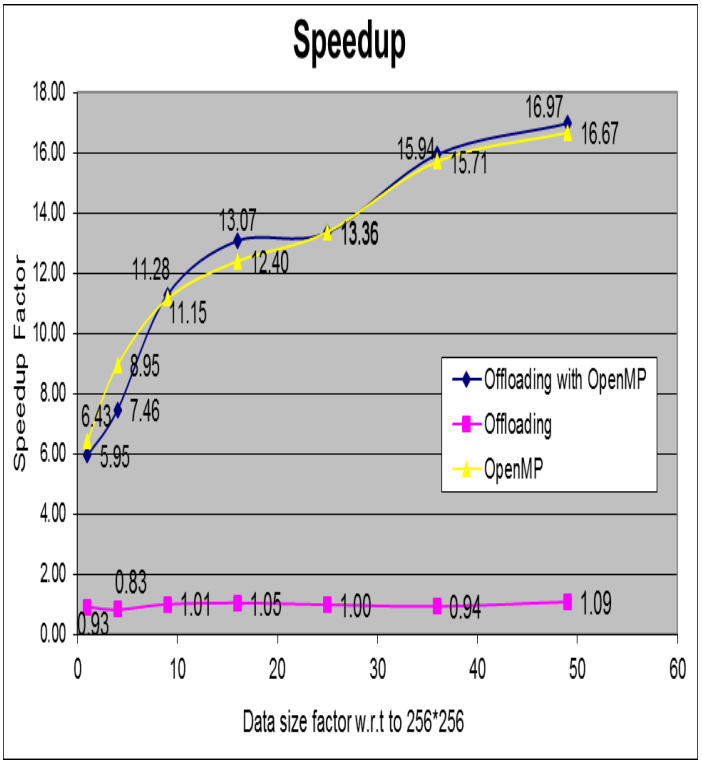

Fig. 5 Speedup factor as a function of Data size factor of offloading, OpenMP and both.

Case 1: The K-means clustering code is run with both Offloading followed by OpenMP. The parallel and compute intensive portions of the code were offloaded to Intel Xeon phi coprocessor on which the code is run in under OpenMP directives. The volume of data size is changed by a factor of49 as described above. This mode utilized all the 48 threads available on host of execution of that part the which is not offloaded to the device. The offload portion of the code is run using all 240 threads. As there are four parallelizable regions in the code, all are offloaded separately. The processor times for each of the offload portions are computed and later summed to compute total offload processing time. Based the total compute times with and without offloads, the speedup factors are computed for each data set whose volume ranges by a factor of 49. These are shown in Fig. 4. The speedup factor ranges from 5.95 to 16.97 as data volume ranges from 1 to 49.

Case 2: In this case, the K-Means clustering code is run without OpenMP optimization directives. The data are transferred to the Xeon Phi device, but processed without OpenMP directives. The data are processed using all the threads available on Xeon Phi device, which is can be up to 240 threads. Based on the total compute times of four offload regions, the speedup factors are computed and shown in Fig. 4. They range from nearly 0.83 to 1.09 as the data volumes ranges by a factor of 49. The compute times do not take into account the data transfer time from host primary memory to the device memory. The data transfer time to Xeon Phi device required by is about 3.3 seconds.

The Fig. 4 shows flow chart of the source code of K-Means clustering in which the three portions of offloading and OpenMP optimizations are indicated.

We have evaluated the performances of OpenMP and offloading case of following 3 options:

Case 3: In this case, the data are not offloaded to the device, but rather processed using only OpenMP directives. In this mode, the code is executed on the host processors Intel Xeon using maximally available 48 threads. The speedup factors are computed from the compute times with and without use of the OpenMP directives (Fig. 4). They range from 6.43 to 16.67, which are almost as the case 1 above.

By using only OpenMP on host gives better result for image size 256 *256, 512 * 512 and 768*768 but as we increases image size more than 768x768, the OpenMP with Offloading on host and device both gives better results than just using OpenMP individually on host. In offloading, the data transfer overhead should be much smaller than computation time. This can result if data volume is smaller and the code is highly compute intensive. For large volume data and less computations, the offloading is not economic. The data transfer process first initializes and allocates the memory on device and then map the variable to device from host. Therefore to get benefits of offloading, it is required to ensure that the data transfer overhead should be smaller than the communication overhead.

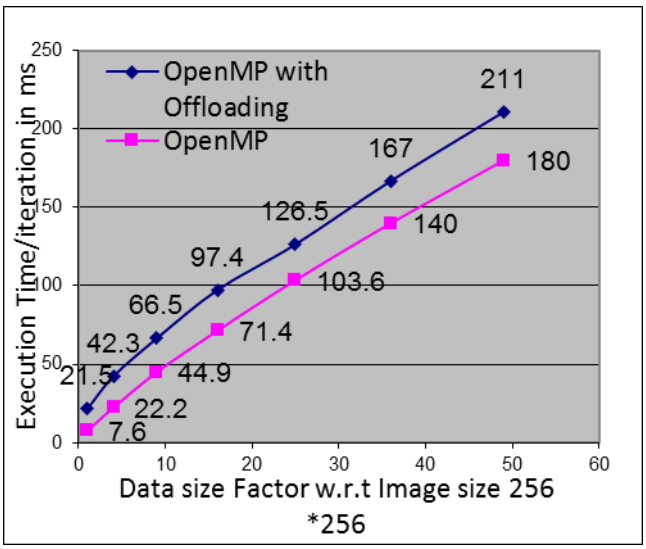

Fig. 6 Execution Time of offloading followed by OpenMP on Intel Xeon Phi coprocessor (device) and Intel Xeon core processor (host).

The Fig. 5 shows the Execution Times (required per iteration in K-Means clustering algorithm) of offloading followed by OpenMP optimization on host and device along with the Execution Time of OpenMP on host. Here, it is observed that OpenMP on host gives better result than that offloading computation on device. This is because of the slower 1.24 GHz processor speeds of Intel Xeon Phi coprocessor 7120P compute elements (device) which is less than 2.50 GHz processor speed of Intel Xeon 12-core processor E5-2680. Thus, OpenMP on host gives better result because the reading and writing file is faster and no communication overhead is negligible.

In Fig. 6 shows execution time per iteration by using offloading execution mode with OpenMP directives for data size raging up to 49. It increases from 21.5 ms for 256x256 size image data to 211 ms for 1792x1792 size images. In absence of optimization, the time should increase by a factor of 49 to 1053.5 ms (= 21.3*49) for later size image data. The computed per iteration execution time with respect to 256 * 256 Image size time (21.5 ms) is higher than the actual per iteration execution time. The Fig. 7 shows the speedup factors increasing with image size. It is seen that the speedup factor follows the data computation load provided by the data size. With initial faster increase in data size also is also accompanied by faster increase in speedup factor.

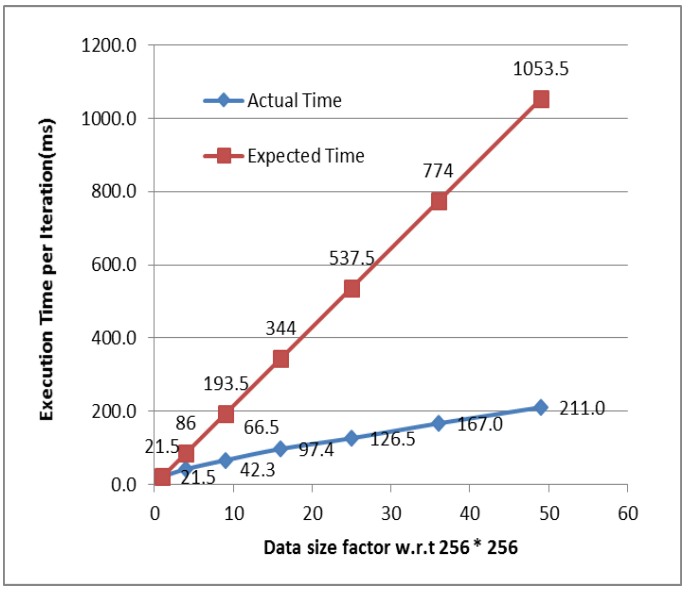

Fig. 7 Comparison of Calculated Execution Time w.r.t to 256*256 Image with Actual Execution Time by utilizing host and device both.

VI. CONCLUSION

As many parallel computing techniques and high performance computers are evolved so far, the present study analyses the Intel Xeon phi coprocessor 7120P(device) HPC accelerator performance with processor base frequency of 1.24 GHz along with OpenMP Parallel computing model. By utilizing both the host processor (Intel Xeon 12-core processor E5-2680 v3@2.50) and Intel Xeon Phi coprocessor, the performance can be accelerated when computation offloading execution mode is used. In computation offloading with OpenMP directives, the performance increases but not much because of some essential reason which are required to be considered while offloading. Firstly, offloading will not give best result with small data size. To get the full benefits of offloading on Intel Xeon phi coprocessor, it is required that Communication overhead should be lower than computation time. Secondly, the data size should be large enough. Thirdly, as hosts have usually higher processor speeds than the processors in a device, only OpenMP mode gives better result on host than that on device. Hence, Offloading with OpenMP utilizing both processor and coprocessor gains accelerations and increases

the performance if communication overhead is less than the computation times which is highly application dependent.

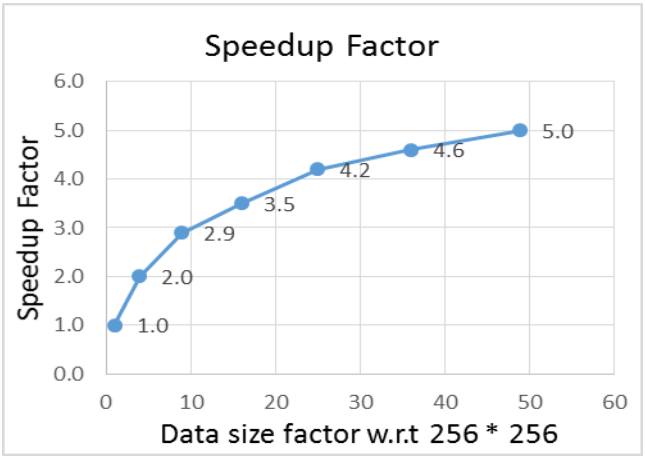

Fig. 8 Speedup factor of Calculated Execution Time w.r.t to 256 * 256 Image with Actual Execution Time by utilizing host and device both.

VII. ACKNOWLEDGMENT

We the authors thank Director, BISAG, for providing infrastructure and encouragement, and DDU, Nadiad for permitting to carry out this project at BISAG.

VIII. REFERENCES

[1] Chauhan, Kajal, C. K. Bhensdadia, and M. B. Potdar, (2017), Parallel Computing Models and Analysis of OpenMP Optimization on Intel i7 and Xeon Processors, International Journal of Computer Science and Software Engineering (IJCSSE), Volume 6, Issue 12, p. 315-322.

[2] Intel® Xeon Phi™ Coprocessor Architecture for Software Developers,https://software.intel.com/en-us/articles/intel- Xeon-phi-coprocessor-architecture-for-software-developers.

[3] Wilt, Nicholas. The CUDA handbook: A comprehensive guide to GPU programming. Pearson Education, 2013.

[4] Intel® Many Integrated Core Architecture – Advanced, https://www.intel.in/content/www/in/en/architecture-and-technology/many-integrated-core/intel-many-integrated-core-architecture.html.

[5] 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.

[6] A comparison of heterogeneous and Many Core Programming Model, https://www.hpcwire.com/2015/03/02/a-comparison-of-heterogeneous-and-manycore-programming-models/

[7] Intel Xeon phi Programming Environment, https://software.intel.com/en-us/articles/intel-xeon-phi-programming-environment

[8] Kowalik, Janusz, Piotr Arłukowicz, and Erika Parsons. "Speeding Up Computers." arXiv preprint arXiv:1603.05487 (2016).

[9] Culler, David, et al., "LogP: Towards a realistic model of parallel computation." ACM Sigplan Notices. Vol. 28. No. 7. ACM, 1993.

[10] James Jeffers, James Reinders, "Introduction" in Intel Xeon Phi Coprocessor High Performance Programming, 2013.

[11] CUDA vs. Phi: Phi Programming for CUDA Developers, http://www.drdobbs.com/parallel/cuda-vs-phi-phi-programming-for-cuda-dev/240144545.

[12] Capotondi, Alessandro, and Andrea Marongiu, 2016, "On the effectiveness of OpenMP teams for cluster-based many-core accelerators", in High Performance Computing & Simulation (HPCS), International Conference on. IEEE, 2016.

[13] Intel Pentium Processor, https://www.intel.com/content/www/us/en/products/processors/pentium.html.

[14] Hybrid Computing – Coprocessors/Accelerators Power-Aware Computing – Performance of Applications Kernels,https://www.cdac.in/index.aspx?id=pdf_xeon-phi-prog-overview-hypack

[15] Intel Many Core Platform Software Stack (Intel MPSS), https://software.intel.com/en-us/articles/intel-manycore-platform-software-stack-mpss

[16] Intel® Xeon Phi™ Coprocessor Developer's Quick Start Guide,https://software.intel.com/en-us/articles/intel-xeon-phi-coprocessor-developers-quick-start-guide

[17] Intel® Parallel Studio XE, https: //software.intel.com/en-us/parallel-studio-xe