5-2017

An Analysis of Variation Between Cores For Intel Xeon Phi Knights Corner And Xeon Phi Knights Landing

Jamar Robinson
Clemson University, robinsr@clemson.edu

An Analysis of Variation Between Cores For Intel Xeon Phi Knights Corner And Xeon Phi Knights Landing

A Thesis
Presented to
the Graduate School of
Clemson University

In Partial Fulfillment
of the Requirements for the Degree
Master of Science
Computer Engineering

by
Jamar Robinson
May 2017

Abstract

As we move towards exascale computing, the efficiency of application performance and energy utilization, must be optimized by redefining architectural features and application performance analysis. This research analyzes the performance per core of 8 applications on Intel Xeon Phi Knights Corner (KNC) and Knights Landing (KNL) to determine if performance variation within cores can lead to performance and energy improvements. Our results showed that KNC architecture's core vary in performance, leading to faster inner core performance as a result of memory characteristics and core utilization. It also shows that cores 17, 34, and 51 on the KNL architectures performs consistently slower than other cores, with core 0 performing either faster, slower or within the average performance time all the cores. A power performance study was then done utilizing different core configurations on the KNC. The results show that by targeting inner cores for applications that exhibit better inner core performance, a maximum energy reduction of 16.4% compared to a configuration using all cores was possible with its optimal thread configuration. Energy reduction was achieved with along with a 2% reduction in the fastest execution time of the same application. Our results also show how application characteristics lead to different core variation performances on KNC and KNL Xeon Phi architectures.

Dedication

I dedicate this thesis to my family, my academic advisor and department chair whose belief in and support in me has lead to this milestone achievement for me and my family. I also dedicate this to my research group members of the Future Computing Technology Laboratory whose friendships and insights are integrated into this body of work.

Acknowledgments

This Body of work was made possible due to the help and support of the following persons.

I would first like to thank my academic advisor Dr. Melissa Smith, whom not only introduced my to the area of High Performance Computing, but also guided me in accomplishing this masters thesis. Both her an Dr. Noneaker's support has helped me navigate through my academic requirements successfully.

I would next like to thank the faculty and staff of Clemson University's Electrical and Computer Engineering Department. Special thanks goes out to Dr. Adam Hoover and Dr. Richard Brooks whom have helped developed my research acumen and necessary skills to document and present my research findings.

I would also like to thank Dr. Barry Rountree of Lawrence Livermore National Laboratory whom this core variation investigation started with and whom has helped me along the way in completing this body of work.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

As we move toward exascale computing, the efficiency in performance and power consumption of accelerators is becoming increasingly important, due to reduction in performance leaps with silicon advancement. This new computing challenge has resulted in accelerators being developed by redefining the architectural features and structure to increase computing performance, while minimizing energy consumption. The software model used for application development is also influenced by these hardware changes to further optimize computing in performance and energy consumption. The two major accelerator architectures currently leading the charge to exascale computing are the Many Integrated Core Architecture (MIC) by Intel and the General Purpose Graphics Processing Unit (GPGPU) by Nvidia.

This research will explore the Many Integrated Core platform by Intel,in particular how it can be used to possibly increase performance and power efficiency. The Intel MIC architecture is in essence the packaging of a large amount of cores on a chip to form a energy efficient, highly parallel processor architecture. This architecture, which now also features a high bandwidth on package memory and wide vector registers, achieves extremely high performance due to its high degree of parallelism and ease of programming. The Intel product name for this architecture is the Xeon Phi. There are currently two generations: the Knights Corner (first generation introduced in 2013) and Knights Landing (second generation introduced in 2016).

Few studies over the past few years on Intel CPU cores has however shown that the manufacturing process has lead to variation in power and temperature of Intel chips such as Haswell and Sandybridge. These differences also manifest itself into variation with processor frequency especially when these processors are under turbo boost dynamic overclocking. Research looking at these observations such as chun [12] shows how this can impact performance significantly on supercomputers using multiple nodes of CPU's with cores under performing compared to others. This is due to the fact that high performance computing workloads, such as homogeneous synchronous applications which are running on multiple cores or processors are limited by the slowest ranks [12]. Their research has shown that per core performance can be affected by about 18% with dynamic overclocking with Xeon processors running jobs across multiple nodes. There has however been no research at this time focusing on analyzing performance variations on Intel Xeon Phi cores and its effects. This is an important research area as based on the level of variation between cores; their may be significant performance or efficiency gains obtained by MIC architecture as it contains upwards of 50 cores. As a result if cores are under performing there may be performance impacts across homogeneous applications across multiple nodes.

This research will focus on analyzing the performance variations of cores using different applications on Intel Xeon Phi KNC and KNL. Analysis of the per core variation on these chips will lead to classification of algorithmic properties that influence performance and energy consumption on the Xeon Phi platform. This research will also identify faster performing cores which can be utilized to improve performance and energy consumption for particular applications.

The analysis of per core performance of applications on the Xeon Phi platform for different applications will also provide more information on how these applications work on a per core level. This body of research utilizes different benchmark applications either with large amounts of memory accesses (memory bound) and/or significant computations, i.e. high degree of floating-point operations compared to non floating-point operations. The existence of core variations due to differences in algorithmic features of these applications can be very useful to developers. If results show that particular types of applications lead to more or less core variation which impacts on performance, developers may decide to use a different architecture which may have improve application performance and energy consumption due to consistent core performance.

In analyzing application to architecture mapping for these cores, the intent is to have a finer granularity of results on a hardware and algorithm level. This is needed as If we only utilize elements showing whether or not a memory bound or compute bound application match well to the cores, this may lead to inconclusive application analysis. In analyzing the mapping of application to per core performance it should also be noted that the performance of one core will not necessarily indicate how an application will perform when all cores are being utilized. There are other architectural elements and system features that may also influence this collective performance.

The contributions made by this research are:

1. Prove that core variation with respect to performance exist in Intel Xeon Phi KNC and KNL

2. Identification of application characteristics that influence core variation performance in Intel Xeon Phi KNC and KNL

3. Identification of application characteristics that can be utilize to save energy on Intel Xeon Phi KNC

Chapter 2

Related Work

2.1 Core Variation

There are a few research studies in the current literature that analyze the variability of cores within processors. These papers are mostly focused on Intel multicore processors and their turbo-boost features as this is were the phenomena was widely discovered. The most comprehensive investigation is done in the paper by Acun, et al. [12]. Here the authors ran compute intensive kernels on four supercomputers: Edison [3], Cab [2], Stampede [8] and Blue Waters[1] to observe variation among processors with applications Naive-DGEMM, MKL-DGEMM, LEANMD, JACOBI2D. The results show that with turbo boost enabled, core performance varied up to 16% on the Edison cluster. The lowest core variation performance was on the Blue Water Cluster, which uses AMD chips. These authors also propose a dynamic load balancing solution that achieved performance improvement of up to 16% with each individual core operating on approximately the same frequency in a multi-node execution.

Rountree, et. al from Lawrence Livermore National Laboratory also reports a variation among processor cores with a power bound in an experiment with Sandy

Bridge processors [20]. His experiment showed that with different power caps on the processors there is a performance degradation as the power level decreases. Acun, et al. [11] investigates the correlation between using turbo boost and these power differences. They also analyze possible impacts of core temperature on the behavior of turbo boost. Results show how the frequency of a processor is affected by temperature and the resulting impacts on multi-node applications. From a design perspective the authors Hebert et. al [17] shows that the design process of processors using frequency island and globally clocked techniques leads to variability. The results show how the tolerance used in the design of processors using these techniques impact on core performance. The authors also showed the impact of larger and smaller core design on performance variability within processors. .

2.2 Xeon Phi

The previously mentioned research papers focus on Intel Xeon processors with Sandy Bridge, Ivy Bridge and Haswell architectures. Most of the research involving the Xeon Phi focuses on energy and performance utilizing all cores with different thread configurations [23]. Research has also been conducted in analyzing performance trade offs for this architecture with different types of applications in single [16] and multi-node instances [22]. Currently, there is ongoing research with the new Knights Landing architecture that has similar focus to that done for KNC and aims to differentiate how the new hardware architecture features changes performance.

There has been less research however in relation to the performance and power consumption per core of these Intel MIC chips. This is due to lack of studies of per core variation as they are believed to be equally manufactured and hence, even if there were differences it would have minimal impact on the performance. The Xeon phi hardware also has no reliable method of obtaining per core power or temperature data for a reliable power study.

At the time of writing, no published study existed in the literature on core performance variation of Xeon Phi Knights Corner or Knights Landing. The majority of published literture on the Xeon phi is based on performance modelling of all cores with a few articles on modelling energy on the Xeon phis [18]. Other research focuses on analyzing how applications perform with and without being optimization for Xeon Phis using parallel programming paradigms such as OpenMP and MPI. A study by Schmidl et al.[23] highlights the discrepancy between the theoretical memory bandwidth of the Xeon Phi (352 GB/s) and that the measured in performance testing with micro-benchmarks(150GB/s). Another performance paper [25] shows how optimization's for the Parsec micro-benchmarks improves performance over Sandy Bridge Xeon chips on the effects of the VPU and pipeline stages of the KNC. These works provided great insights into the architecture of the Xeon Phi and the possible causes of performance variations.

The Knights Landing Xeon Phis were released last year in fall 2016 and as a result a limited amount of work has been published analyzing the performance of this platform. Research papers thus far are focused on the new elements added to the KNL (compared to the KNC) and not necessarily on the per core performance. These research provides valuable information regarding execution of diiferent applications and how the new hardware additions will possibly impact on application performance. The authors in [19] have detailed how the new memory architecture helps increase memory bandwidth and reduce latency.The authors also provide insights for taking advantage of the new memory architecture, with a sample case study using the Black-Scholes benchmark. In [21], Rucci et al. analyze the optimization of the Smith Waterman bio-informatics algorithm on the KNL architecture and the performance effects with different thread configurations and memory bandwidth. This Information lead to choosing the cluster and memory mode used for the KNL architecture in this research.

A comprehensive energy performance studied by the authors in [24] also provided vital information relating to the KNL architecture. The research conducted by these authors involved the optimization of a modelling system for nested air quality. The optimized model implemented on the KNL architecture showed improvements of 26% reduction in energy compared to an Intel Xeon e5-2697v4 processors. Their results also showed a total efficiency (performance + energy improvements) of 47% over the previously mentioned processors. This body of work was instrumental in understanding application performance and energy performance of the KNL architecture compared to KNC, and how optimization's of KNL applications influence performance over KNC Xeon phi.

2.3 Summary

This chapter details the research studies in the area of core variation and performance studies on Xeon phi KNC and KNL. In the area of core variation there has only been a few research studies but they show insightful discoveries on Intel processors. This research will be the first to explore this phenomena of core variation on Xeon Phi KNC and KNL. This research will also be the first utilizing the Rodinia benchmark to explore this phenomena. The results of the application performance on both these architectures, will then be used to perform an application to architecture analysis for performance and power consumption on the Xeon Phis. This methodology of doing a performance based study utilizing per core characteristics will be a first in this area of computer engineering.

Chapter 3

Intel Xeon Phi Platform

In this chapter we introduce the super computing cluster Stampede and the two architectures hardware features used in this research: Knights Corner and Knights Landing. We talk about the hardware features and their effects and how the two architectures are different. The different types of usage modes are also explained for the architectures and attention is drawn to the mode(s) used in this research.

3.1 Stampede

The Stampede Cluster is located at the Texas Advanced Computing Center and is ranked 17th in the top 500 supercomputing list with a total of 462,462 cores and peak teraflops of 8520.10 as of Spring 2017. The Stampede Cluster can be viewed as having two different hardware clusters: the original Sandy Bridge cluster and the Stampede KNL cluster. The system uses slurm to schedule jobs on the cluster and uses the Lustre shared file system across all nodes with three files systems: Home, Work and Scratch.

The Sandy Bridge Cluster contains 6400 Sandy Bridge nodes with either one

Table 3.1: Stampede Hardware Configuration for Sandy Bridge Cluster [8]

Component	Technology
Sockets per Node/Cores per Socket Coprocessors/Cores	2/8 Xeon E5-2680 2.7GHz (turbo, 3.5) 1/61 Xeon Phi SE10P 1.1GHz
Motherboard	Dell C8220, Intel PQI, C610 Chipset
Memory Per Host Memory per Coprocessor	32GB 8x4G 4 channels DDR3-1600MHz 8GB GDDR5
Interconnect Processor-Processor Processor-Coprocessor	QPI 8.0 GT/s PCI-e
PCI Express Processor PCI Express Coprocessor	x40 lanes, Gen 3 x16 lanes, Gen 2 (extended)
250GB Disk	7.5K RPM SATA

or two Xeon Phi KNC processors connected to each node. The Sandy Bridge nodes contains two Xeon E5-2680. This Sandy bridge Xeon has a clock frequency of 2.7GHz, turbo boosted up to 3.5 GHz with 8 cores. The Xeon Phi devices are the first generation SE10P model. There are memory capacity configurations on the nodes but our experiments were done with the 8GB GDDR5 KNC memory configuration. The network configuration used in this section of the cluster is the Mellanox FDR infiniband with fat-tree topology. An overview of the cluster is provided in Table 3.1.

The KNL Stampede Cluster is constructed differently since the KNL processors are not traditional coprocessor cards like the first generation KNC; the KNL processors can form stand-alone nodes. There are a total of 508 nodes with each node being a self hosted KNL card with 68 cores, 4 threads per core. Each node runs Centos 7 and consists of 96GB of DDR4 Random Access memory, and 16GB high bandwidth Multi Channel Dynamic Random Access Memory. The network configuration used on this part of the cluster is the Omni-Path network fat tree topology with 100GB/sec. An overview of the cluster is provided in Table 3.2.

Table 3.2: Stampede Hardware Configuration for KNL Cluster[8]

Component	Technology
Sockets per Node/Cores per Socket	1/68 Intel Xeon Phi 7250 KNL 1.4GHz
Memory Per Host	96GB channels DDR4-1600MHz 16GB MCDRAM
InterconnectProcessor-processor	100Gbsec Omnipath
112GB Disk	SSD

3.2 Xeon Phi Knights Corner (KNC)

The Intel xeon Phi KNC coprocessor contains 61 scalar unit processor cores. Each core can execute 2 instructions per cycle and contains a L1 cache and D cache. The L2 cache is located on the core ring interface, with each core having two connections with the core ring interface. The core ring interconnect is a high performance bidrectional bus. An overview of the architecture is shown in Figure 3.1. The Xeon Phi has data and instruction L1 caches of 8-way set associative 32KB with a cache line size of 64 byte. The L2 cache is also 8-way associative with each core contributing 512KB to the L2 cache for a total of 31MB. The cache latency for L1 is approximately 3 cycles and 14-15 cycles for the L2 cache. There is also a TLB for L1 Data and a L2 TLB that behaves like a second level TLB.

Instructions can be pipelined at a throughput rate of one vector instruction per cycle. The Xeon Phi core is able to fetch and decode instructions from four hardware threads, with each instruction set able to utilize a dedicated 512-bit wide vector floating-point unit (VPU). The VPU is able to perform 16 single-precision floating-point, 16 32-bit integer operations or 8 double precision-floating-point operations per cycle. The vector register file contains 32 512-bit wide registers per thread context with the ability to hold 16 singles or 8 doubles[10].

The cores are able to execute 2 instructions per cycle because it has a 2-wide processor with an operation executing on the U-pipe and another on the V-pipe. The

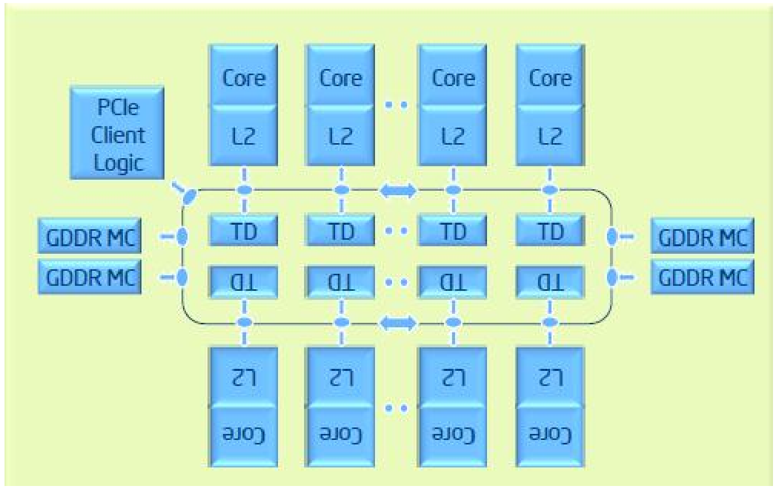

Figure 3.1: Xeon Phi KNC Architecture Overview [4]

vector unit is able to communicate with the core and executes allocated instructions on the U or V pipe. The multi-threaded nature of the cores with 4 each, reduces the effect of vector pipeline latency and memory access latency through concurrent execution of up to 4 threads per process. The Xeon phi has 8 memory controllers supporting up to 16 GDDR5 channels, which facilitates a transfer speed of 5.5 GT/s and a theoretical aggregated bandwidth of 352 GB/s.

There are three ways to execute applications on KNC: Native, Heterogeneous Offload and Cilk Offload. The main models used in high performance computing due to ease of use is the Native and Heterogeneous Offload model. Native Execution allows your application to run entirely on the Xeon Phi coprocessor and is the easiest and most used mode as it only requires programmers to add a compilation flag of *-mmic* and requires very little to no code changes from the original implementation in MPI or OpenMP. Applications parallelized in MPI and OpenMP paradigms will achieve good performance benefits if there are few serial segments, small number of I/O operations and low memory utilization. The application used in this research are compiled an executed in this mode.

In the Heterogeneous Offload model, the main processor connected to the KNC executes the application and the programmer designates parts of the code to be offloaded and executed on the KNC. This mode is most representative of the heterogeneous computing model in high- performance computing. Portions of the application that are embarrassingly parallel are identified for execution on the KNC using pragmas and compiler directives. More changes are needed to get applications operating in this mode. Applications with few instances of embarrassingly parallel regions will spend significant time transferring data back and forth between the main processor and KNC coprocessor, which may negatively impact performance times. The native execution model is selected for our experiments in this research to focus on the full performance of cores with no additional external factors that may add variations to the results.

3.3 Xeon Phi Knights Landing (KNL)

The Intel Xeon Phi KNL processor contains 72 Silvermount-based cores running at 1.3 - 1.4 GHz connected in an out-of-order configuration. 36 compute tiles are connected in a 2d mesh with each tile composing 2 cores as can be seen in Figure 3.2 with each tile configuration shown in Figure 3.3. The core is based on the Intel Atom Silvermount micro architecture core with each containing two AVX-512 vector processing units and 4 SMT threads. The core is able to execute up to six operations per cycle. The cores also support all legacy x86 instructions making it backward compatible with previous Intel processor targeted binaries. Figure 3.4 shows the binaries supported from the Sandy Bridge, Haswell and KNL instruction set. The cores are divided into five units: the front-end unit (FEU), the allocation unit, the integer execution unit (IEU), the memory execution unit (MEU), and the VPU.

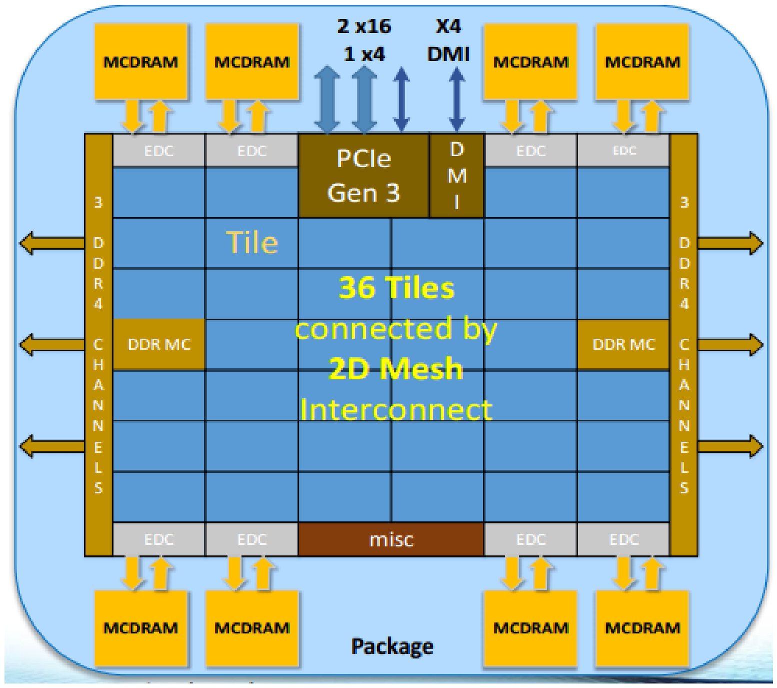

Figure 3.2: Xeon Phi KNL Mesh Architecture Overview [9]

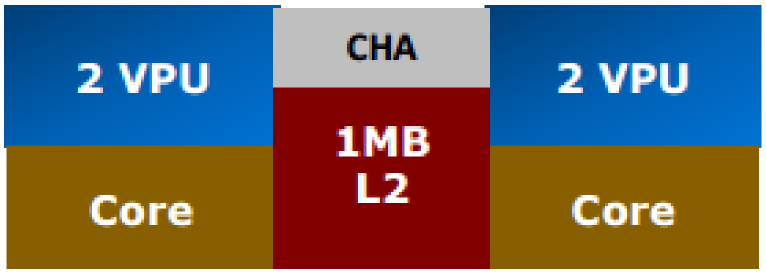

Figure 3.3: Xeon Phi KNL Tile Configuration[9]

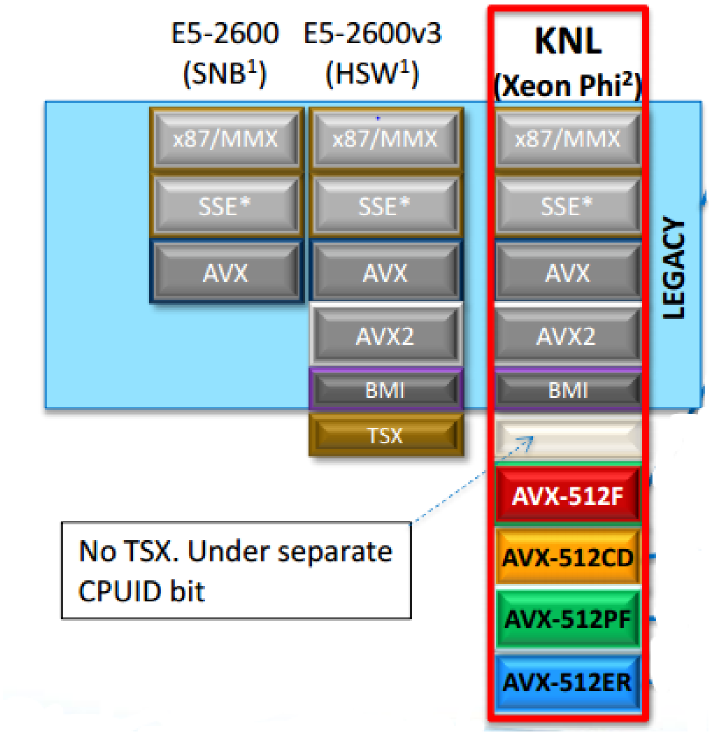

Figure 3.4: Xeon Phi KNL Instruction Set Support[9]

Multiple core architecture improvements have been implemented to improve performance over the KNC generation. The core now supports out of order execution with increased buffer support that helps with instruction level parallelism. Each core has 2-wide decode/rename/retire stage, 72 entry ROB and rename buffers, 6-wide execution and 72 in-flight core out of order buffers. There are two main versions of the KNL processors: PCIe-card based and Self Boot Socket with the latter eliminating the bottleneck that occurs with use of a PCIe connection to a host/support processor.

The interconnect of the KNL is a 2D cache coherent mesh architecture connecting I/O controllers, memory controllers, and other elements. It uses the MESIF (modified, exclusive, shared, invalid, forward) cache coherent protocol. It consiss of a 16GB high-speed stacked memory accessible by high speed memory controllers, as well as a maximum capacity of 384GB of 2,400 MHz DDR4. The Xeon Phi has a data and instruction L1 cache of 32KB and cache line size of 64 bytes similar to the Xeon Phi KNC first generation. It now also contains L1/L2 prefetchers and fast unaligned, cache-line split support for fast gather/scatter operations.

3.3.1 Memory Modes

The new generation Xeon Phi also contains three different memory modes: Cache, Flat and Hybrid mode as shown in Figure 3.5, which are determined at boot time. The KNL now contains a multi-channel dynamic random access memory (MC-DRAM), which is a form of on-package memory and is a high-bandwidth memory on the CPU chip next to the cores. This type of memory is capable of performing 5 times as fast as traditional DRAM memory with theoretical bandwidth up to 400+ GB/s, resulting in improve performance for applications that are limited by memory bandwidth. Applications such as vector dot-product and matrix vector multiplications

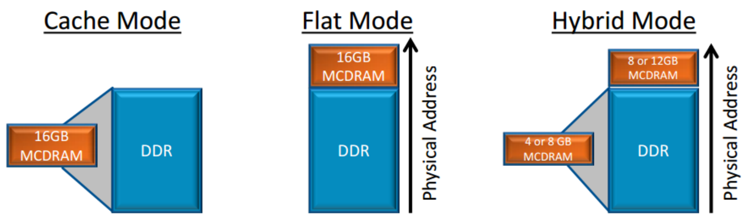

Figure 3.5: Xeon Phi KNL Memory Modes

with low arithmetic density will experience better performance with MCDRAM.

The maximum amount of MCDRAM per KNL chips is 16GB and it can be used in three different modes. The MCDRAM can be used as addressable memory in flat mod, where the MCDRAM is viewed as another portion of DRAM but may may require modifications to the application code for maximum performance. From a system perspective, the MCDRAM is allocated as another NUMA (Non uniform memory access) node without cores.

MCDRAM can also be used in cache mode where it is allocated as system cache. When the KNL is configured in this mode, the MCDRAM is a last level cache in between the L2 and the DDR4 memory. No work is required on part of the programmer as the operating system allocates the MCDRAM as cache. This mode may however lead to an increased latency due to cache misses in both L1, L2 and MCDRAM. When a miss occurs in the MCDRAM this leads to increased latency in retrieving data from the DRAM.

There is also a hybrid mode in which the MCDRAM is used as both addressable DRAM and cache and will gain benefits of each scenario. This mode will not be explored in this research as the Stampede Cluster used in this research does not support this mode.

3.3.2 KNL Cluster Modes

The KNL cluster has several modes that enable cache coherency. Coherence on the KNL tile architecture requires both tile-to-tile and tile-to-memory communication with cores that manage and utilize the data. There are three modes on the KNL cluster:

- All-to-all - This mode is the easiest and most flexible mode but does have higher latencies compared to other modes because the processor does not attempt to optimize coherency-related communication paths.
- Quadrant (hemisphere) - In this mode communication is localized by grouping tiles into four logical quadrants. It then requires each quadrant to perform local memory management for addresses on the MCDRAM and DDR. As a result the average number of hops for tile-to-memory request is reduced compared to that for all-to-all.
- Sub-NUMA 4 - This mode divides the chip into four NUMA nodes like a four socket processor to optimize coherency-related on chip communication. To get performance benefits from this mode it requires manual memory management and more programming effort compared to the other modes.

3.4 Summary

This chapter presents an in depth look at the Stampede cluster which contains the KNC and KNL accelerators used in this research. The different architectural components across the two generations are discussed and components that may influence application behaviors presented. Both architectures can be utilized in a variety of ways in running applications on different clusters. The framework work of the Intel MIC framework is the foundation of these two architectures but the hardware components and the utilization of the different modes make the two Xeon Phis are vastly different. This chapter also discusses the modes of both generations of Xeon Phis that will be used in this performance study and mentions the reason these modes are chosen.

Chapter 4

OpenMP Benchmark Applications

In this chapter the programming API OpenMP, which is used in this research, is introduced along with the various applications used for core performance analysis. The application characteristics are discussed and the configuration of inputs used for each application analysis is presented.

4.1 OpenMP

OpenMP (Open Multi Processing) implements different compiler directives and libraries that enables C, C++ and FORTRAN code to exploit/implement parallelization on CPU's via shared memory multiprocessing. Parallelization is focused on sequential loops by specifying pragmas around these instructions. The execution of these codes will continue sequentially until it reaches the pragma directives; where it will initialize parallelism execution and when finished, it will continue with sequential execution. The OpenMP methodology allows for straighforward parallelism of code with minimal restructuring.

The overall implementation of parallelism in OpenMP is based on the multithreading model, whereby a master thread forks other threads called slave threads and the system assigns different tasks to them for execution. This model is especially suitable for this investigation since it allows the Xeon Phi cores to be targeted individually and enables the best possible performance for these algorithms with their parallel implementation. In this research, seven OpenMP applications from the Rodinia Benchmark were used in addition to a sparse matrix multiplication application.

4.2 Rodinia

The Rodinia benchmark was developed/organized by the university of Virginia to create a diverse collection of applications/benchmark for supercomputing accelerators [15]. The suite includes implementations in OpenMP, CUDA and OpenCL parallel programming paradigms. This research utilizes the OpenMP implementation of the following applications: Needleman-Wunsch, Breadth-First Search, Speckle Reducing Anisotropic Diffusion (SRAD), Streamcluster, Particle Filter, LavaMD, and Myocyte. The benchmark version used was 2.2 and the application's OpenMP versions were optimized for multicore CPU processors. No additional optimizations were done to the benchmark codes other than compiler optimizations used to enable them to work on the KNC and KNL Xeon Phis. Table 4.1 lists the Rodinia applications and their respective domains.

4.2.1 Breadth-First Search (BFS)

Breadth-First Search is an algorithm for searching or traversing a tree or graph data structure. The data is stored in an adjacent matrix that is represented by a 2-dimensional array in the Rodinia BFS implementation. The algorithm works by first searching neighbouring nodes on each level, and then moving on to the next lower level

Table 4.1: Rodnia Applications Analyzed and Their Domains

Applications	Dwarves	Domains
Needleman-Wunsch	Dynamic Programming	Bioinformatics
Breadth-First Search1	Graph Traversal	Graph Algorithms
SRAD	Structured Grid	Image Processing
Streamcluster1	Dense Linear Algebra	Data Mining
Particle Filter	Structured Grid	Medical Imaging
LavaMD2	N-Body	Molecular Dynamics
Myocyte	Structured Grid	Biological Simulation

and repeating this procedure. Each level of neighbouring nodes that is discovered is added to a queue at each level. The Rodinia implementation reads in a file with the data to be traversed and implements the BFS algorithm by parallelizing the for-loop using the OpenMP pragma. Analysis in this research uses four threads and an input file with eight million data points.

4.2.2 Needleman-Wunsch (NW)

Needleman-Wunsch is a dynamic programming algorithm used in the domain of DNA sequencing. It is a nonlinear global optimization method with potential pairs of sequences organized in a 2D matrix. It is one of the first applications of dynamic programming to compare biological sequences. The algorithm works by first filling the matrix from top left to bottom right, step by step. A score is assigned based on the value of the maximum weighted path ending at that particular cell. The optimum alignment is then found to be the pathway through the matrix array with the maximum score. Hence the value of each data element depends on values of its northwest-, north- and west-adjacent elements [14]. The maximum path is then traced backward to deduce the optimal alignment. The OpenMP implementation in this research uses OpenMP pragmas to parallelize the for-loops used to compute the alignment of the top left and bottom right matrix. The application requires three inputs: the x- and y-dimensions which was set to 16384, a penalty value that was set to 10 and the number of threads which was set to 4 for the experiments in this research.

4.2.3 Streamcluster

Streamcluster is a dense linear algebra algorithm that takes a stream of input points, and finds a predetermined number of medians so that each point is assigned to its nearest center. The quality of the clustering is measured by the sum of the squared distance (SSQ) metric [13]. This algorithm is used in the field of data mining. Because the algorithm is based on the stream benchmark which is used to test memory bandwidth, it serves as a good benchmark in the performance analysis. This application requires 8 inputs: Minimum number of centers allowed (set to 10), Maximum number of centers allowed (set to 15), Dimension of each data point (set to 256), Number of data points (set to 65536), Number of data points to handle per step (set to 65536), Maximum number of intermediate centers (set to 1000), Input file = none, Output file, and Number of threads (set to 4).

4.2.4 Particle Filter

The Particle Filter is a structured grid type application used to track objects in noisy environments with a statistical estimator. The Particle Filter works by making a series of guesses or estimates on the current position based on data from the previous position. The model then determines the confidence of each estimation using a likelihood model. Estimates are then normalized and accumulated to estimate the position. The Particle Filter is used for numerous tracking solutions from object detection in vehicle tracking to tracking blood cells in the body. The Particle Filter implemented from the Rodinia benchmark takes in a video of a moving object that is circular in shape with known background and foreground intensity. This image is corrupted with a zero mean Gaussian noise implemented in the application. The implementation in Rodinia uses an optimized OpenMP of the particle filter to estimate the location of the objects compared to the center. OpenMP pragmas are defined to parallelize the for-loops for weight initialization, weight sharing, model motion, likelihood computation for the particle filter, weight update and normalization, and location estimation of calculated values. This application requires 8 inputs which are configured as follows in this research analysis: x-dimension = 256, y-dimension = 256, number of frames = 100, and number of particles = 20000.

4.2.5 Speckle Reducing Anisotropic Diffusion

Speckle Reducing Anisotropic Diffusion (SRAD) is a structured grid algorithm used to remove locally correlated noise in images without affecting the image features. This application performs image extraction, continuous iterations over the image (preparation, reduction, statistics, and computations) and image compression. The stages must be performed sequentially and hence requires synchronization between each stage. The image used in the Rodinia OpenMP suite is generated by expanding the original image called image.pgm. This application uses OpenMP pragmas for all of the for-loops that handle image manipulations and calculations such as directional derivatives, instantaneous coefficient of variation (ICOV), diffusion coefficient, divergence and updating image. This application requires 5 inputs which are configured as follows: number of iterations = 1000, saturation of coefficient = 0.5, number of rows = 2048, number of columns = 2048, and number of threads = 4.

4.2.6 LavaMD

LavaMD is a N-body algorithm that calculates particle potential and relocation due to mutual forces between particles within a large 3D space. The space is divided into cubes which are further subdivided into cubes. In each box there are particles that interact with each other in the inner box and then other particles in the outer box. The calculations on these particles is a single stage calculation in a loop and these were parallelized to enable calculation with adjacent memory locations. LavaMD is a memory bound application and comparison of the single core CPU versus GPU implementaiton shows that the GPU had a speed up that saturates at 16x. The calculation for estimating the neighbouring distance, charge and force between particles, was optimized in parallel using OpenMP pragmas on the for-loops. This application requires 2 inputs configured as follows: cores = 1, boxes1d = 15.

4.2.7 Myocyte

The Myocyte application simulates cardiac myocyte (heart muscle cell) by modeling 91 ordinary differential equations. These equations are related to biochemical reactions, ion transport and electrical activity, determined by more than 200 experimentally validated parameters. The solution of these ordinary differential equations is mostly a sequential staged solution. At each time step the result is checked with a particular tolerance and if it is not met, another calculation is done. The original application code was implemented in MATLAB ode45 but that implementation did not leave enough room for parallel optimizations so a simpler math solver was used from mathematics source library. There was still not massive gains with this however as parallelism for the application is on a fined-grained level and thread creation/launches incur significant overhead compared to the achievable performance. To get the most performance utilizing OpenMP, the algorithm was implemented with concurrent simulations turning it into an embarrassingly parallel algorithm. Within each simulation the ordinary differential equations solution was calculated with OpenMP pragmas parallelizing the for-loops. The application requires 4 inputs configured as follows for this research:Simulation Time interval = 0, Number of Instances = 30000, Method of Parallelization = 1, Number of threads = 4.

4.2.8 Matrix Multiplication

The Matrix Multiplication algorithm is a standard sparse implementation that multiplies two square matrices. OpenMP pragmas are used to parallelize the for-loops in the implementation. This application only requires one input which represents the size of the matrices to be multiplied. The application also implements a function to ensure that the output is correct and does not produced undefined outputs. The analysis configuration used in this research was a matrix size of 3300.

4.3 Summary

This chapter explores the Programming API OpenMP and applications used in deriving the core variation of the different architectures. A high level description of the applications and their algorithmic properties are explained in this chapter. The different configurations used in executing these applications are also mentioned in this chapter.

Chapter 5

Experimental Setup And Analytical Tools

In this chapter we first highlight how each application is configured for the Xeon Phi platform on the Stampede Cluster. Next, the application parameters are discussed along with details regarding results collection and analysis including the tools used.

5.1 Initial Core Variation Study on Xeon Phi Platforms

The first steps in this research were conducted at the Lawrence Livermore National Laboratory (LLNL) on 2 KNC Xeon Phi processors set up in two different server-based clusters. Initial experiments that inspired this research were conducted on Xeon e5 at this laboratory where core variations were observed in these Intel devices with OpenMP.

To see if this variation was also evident in the Xeon Phi processor family (KNC generation), a matrix multiplication OpenMP application was written and executed 12 times on each individual core with 4 threads and a constant input size. The resulting execution times were then plotted on a graph of core versus time, which revealed that cores in the "middle" of the ring performed faster than cores at the end.

To further verify these results, the Streamcluster OpenMP program was executed on Xeon Phi KNC processors at the TACC Stampede cluster to obtain a larger sample size and confirm these results. The KNC processors at TACC verified the previous findings and lead to a full investigation into the core performance variability in the Xeon Phi family of coprocessors.

5.2 OpenMP Configuration

In researching the core variability we chose to use the OpenMP programming model based on its ability to specifically target cores and maximise/utilize all of the core threads in parallel execution. A collection of benchmarks were investigated to determine which set provided the best domain coverage and variability to test the core performance across application characteristics and functionality. The Rodinia benchmark suite was selected as it incorporated a multitude of real-world application implementations with varying characteristics.

In configuring the thread and core configuration for KNC and KNL processors, there were slight differences in environment commands. The export command was first used to explicitly show that the KNC processor would be active and not the Sandy Bridge CPUs on Stampede. This command was followed by export MIC_KMP_AFFINITY=compact and export OMP_NUM_THREADS=4 to configure the KNC Xeon Phi for compact thread placement of 4 threads. The command export MIC_KMP_PLACE_THREADS was then used to specify the core where the application will execute. For the KNL Xeon Phi processors, only the export OMP_NUM_THREADS=4 and the export OMP_PLACES = core number are needed to select and run the application on the desired core. It should be noted that the number placed in OMP_PLACES is based on the virtual core which is then mapped to the physical core.

5.3 Application Benchmark Configuration

The Rodinia benchmark was installed on the Stampede Cluster and executed on both the Xeon Phi KNC and KNL platforms to determine the best suited applications from the benchmark suite for this research. "Best suited" means applications whose execution time are long enough so as to represent the majority execution time and not noise such as start up time, or executing faster than the sensitivity of our timing program. In choosing these applications, multiple input sizes were used for multiple runs for the applications on both the KNC and KNL platforms. After these tests, the applications Myocyte, Breadth-First Search, Particle Filter, Streamcluster, SRAD, Needleman-Wunsch, LavaMD, and Matrix Multiplication were identified.

A variety of trials were conducted to ensure accurate values and determine the appropriate input sizes for each application. During the initial runs, constant execution times were observed in interactive mode with the job script. However, when multiple jobs each selecting a full node were submitted, it was noticed that after a job was finished executing, another job may be scheduled on the same core. This behavior affected the initial core's performance of the next job, which may be due to forced full utilization of the Xeon Phi core. Further, when the job completes and the next application is immediately scheduled, the heat resulting from the previous job may lead to suboptimal performance. Hence jobs are submitted with 5 trials per core for all cores and after that job is finished the next job is inserted into the queue.

After collecting these initial results, the input size is varied incrementally for each application on the different architectures to see how input size affects performance and how performance variations between cores scale. These experiments were scheduled such that jobs were inserted in the queue after the previous one had finished. Results for the KNL platform was gathered from the normal, flat all-to-all, and quadrant all-to-all as observations confirmed that these queue configurations performed relatively the same.

5.4 Configuration of Intel Vtune Profiler

Hardware-level details was collected using the Intel VTune compiler, while the applications were executed on the KNC and KNL processors. VTune Version 14.1 was installed on the Sandy Bridge KNC cluster and 17.0 on the KNL cluster. VTune is a application profiler developed by Intel for their CPU products. It is able to give you information on a variety of parameters based on how your code performs on the specific hardware with extremely low overhead so as not to interfere with the overall performance. To collect information, the user specifies the type of collection needed and then uses the report function to view the collected data. The Stampede implementation of VTune allows the collection of bandwidth, general-exploration, and data hotspots. For the KNL VTune tool we are able to use advanced hotspots, general exploration and hotspots, however with certain data sizes the information bus could not accurately collect information. So after doing a multitude of testing with both Xeon Phis and the VTune software, the best option to obtain information on performance was general explorations and KNC-bandwidth on the KNC architecture.

Table 5.1: Hardware Components measured on Xeon Phis Using Vtune

Xeon Phi Version	Hardware Component Measured
KNL	Instructions Retired
	Frequency Ratio
	CPI
KNC	Instructions Retired
	L1 hit ratio
	Clockticks
	CPI Rate
	Estimated latency Impact
	Bandwidth

The metrics collected are shown in Table 5.1, with a summary of the data collected after the table.

- Frequency - Shows the frequency at which the core is operating during the execution of the application [5].

- Estimated Latency Impact - Represents the amount of clock cycles that are used for a L1 cache miss; a representation and an indication of L2 performance. The threshold for this value is 145 and any value above this means that the application is performing badly in relation to L2 memory misses [5].

- L1 Hit Ratio - Represents the percentage of memory accesses satisfied by the L1 cache [5].

- CPU Frequency Ratio - The ratio between actual and the nominal CPU frequencies. Values above 1 indicate that the CPU is operating in turbo boost mode [5].

- Instructions Retired - When an application is being executed, the CPU executes more instructions than what is in the flow of the program. The instructions retired variable only counts the instructions that are used by the program and not the other unneeded or unexecuted instructions [5].

- Bandwidth - Represents the bandwidth for the memory writes and reads for the application. It can reach a maximum bandwidth of around 140 GB/sec and memory usage should be investigated if its below 80 GB/s. It should however be noted that for the VTune software, when streaming stores are being used, the bandwidth will be underestimated [5].

- Clockticks - There are moments when the CPU is active and when the CPU is asleep, but the CPU will have a time measurement for both of these events. The clockticks indicator collected by the VTune profiler will only measure the unhalted thread events of the CPU clock, which gives a more accurate measurement of how the application is performing on the hardware and not actions of the system events [5].

- CPI - Clockticks per Instructions Retired (CPI) event ratio, also known as Cycles per Instructions, is one of the basic performance metrics for the hardware event-based sampling collection. This ratio is calculated by dividing the number of unhalted processor cycles (Clockticks) by the number of instructions retired. If The CPI is a high number, it indicates that this code section is taking a high number of processor clock times to execute. A high value for this ratio is 5 and a low value is 1 [5].

An interactive node was used to analyze the applications on KNC with the general exploration option. For the KNL platform, the option with the most promising results was the advanced hotspot selection. The other options produced several errors in compiling the final metrics. As a result, parameters were collected for Instructions

Retired, CPI Rate, Frequency Ratio, and Frequency. KNL results were collected for only two applications as the VTune software running on Stampede began to experience some errors and results could not be validated. As a result, the KNL analysis will be solely based on application performance characteristics, complete VTune analysis will be left for future work.

5.5 System Management and Configuration Utility (MICSMC)

The System Management and Configuration (SMC) Utility MICSMC is a utility provided by Intel that is able to monitor characteristics of the KNC such as temperature, core utilization, memory usage, power consumption, etc [7]. It can used graphically or from the command line. This research utilizes it in a command line script that collects the power consumption of the core every second when the application is being executed. The core utilization information provided by the MICSMC was also vital in analyzing the application performance, as this factor impacts how a particular type of application performance varies between cores.

5.6 Summary

This chapter details the application, programming paradigm (OpenMP) and hardware analysis configuration used in obtaining the data collected for this research. The reasons for the different configurations are discussed as well as problems faced while collecting the data on the different platforms. The following chapter will present the results of all these experiments and different reason for these findings.

Chapter 6

Results

In this Chapter we will analyze how applications perform per core on the Xeon Phi KNC and KNL platforms. We also present hardware information to denote the possible trends that correlates the application performance on the KNC architecture with the per core performance and how both are influenced as input sizes are scaled.

6.1 Streamcluster

6.1.1 Streamcluster Performance On KNC and KNL

Figure 6.1 shows the execution time performance of the Streamcluster application with the analysis configuration mentioned in chapter 4. The average execution time was found to be 55.54 seconds. We noticed execution time faster than 55.5 seconds for 44.67% of the total runs with 97% of the cores with execution time being between cores 10 and 40. The performance of cores 0 - 9 and 41 to 60 had majority of its execution times below 56.4% with only 5% of trials on these core having an execution time above 56.5 seconds.The standard deviation found for 5 trials was 0.43.

The execution times for the five trials on the KNL showed an average perfor-

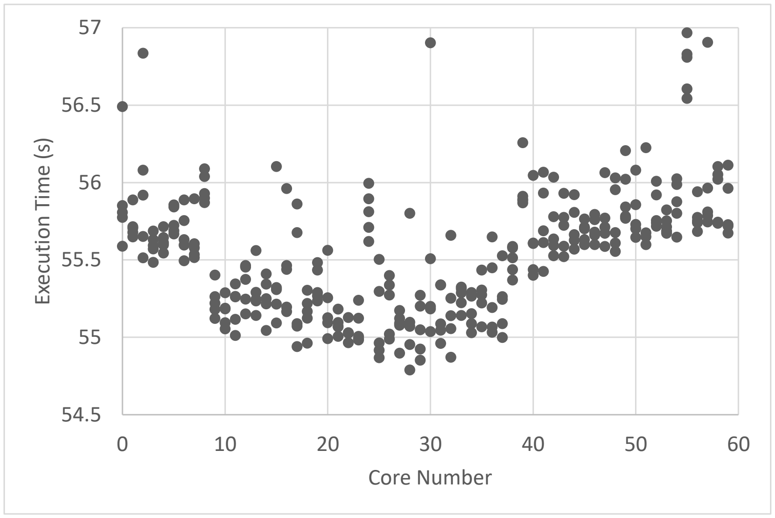

Figure 6.1: Streamcluster Application Performance Per Core on KNC

mance of 26.5 seconds for the streamcluster application. The standard deviation of the trials was found to be 0.52. We however noticed that the performance on cores 17, 34 and 51 consistently performed slower compared to the other cores as can be seen in Figure 6.2 with times of 28.21 seconds, 28.27 seconds and 28.33 seconds respectively.

6.1.2 Streamcluster Core Variation Scale

Figure 6.3 shows the results of the streamcluster application as we scale the input sizes from a dimension of 32768 to 131072 on the KNC. The graph shows that the execution times for cores performing in the middle is still lower as we scale the input sizes. The Percentage difference between the fastest executing core and the other cores increases from the middle core outward up to 4%. Figure 6.4 shows the results of scaling input sizes to a dimension of 114688 on the KNL. Most of the cores have a constant difference between the fastest execution time except for cores 17, 34 and 51. These cores were consistently slower with the same percentage difference. At the highest input sizes of 114688 we however notice that the performance on the

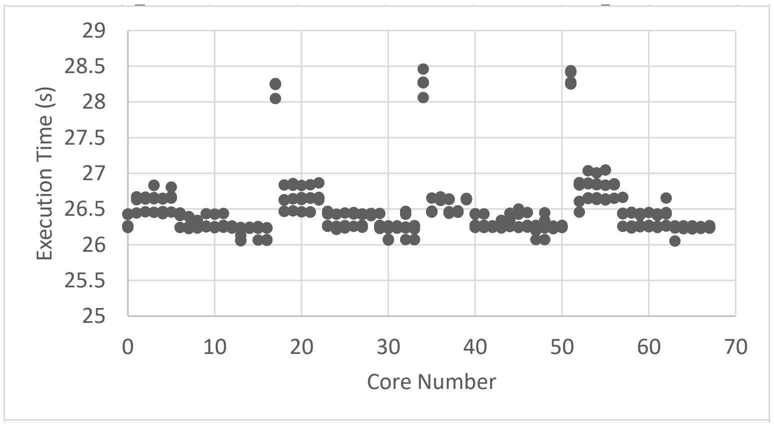

Figure 6.2: Streamcluster Application Performance Per Core on KNL

cores were sporadic. This shows the the performance of this applications scales up to a certain execution point, unlike on the KNC in which the performance scales consistently.

6.1.3 Streamcluster KNC Hardware performance

The Streamcluster is a real world application used in the analysis of memory performance in high performance computing. Its performance is directly influenced by the memory bandwidth of an architecture as it streams in inputs during its execution. Figure 6.5 shows the bandwidth performance of this application and we can notice that the cores in the middle have a higher bandwidth than those at the edge. This gives us some insights on why this particular application performs better in the middle compared to the cores numbered at the edge. From a hardware viewpoint, the bandwidths gradual decrease from inner to outer cores is caused by the ring bus interconnect in the KNC. The cache coherence is set up on the ring bus with cache to cache transfer using Distributed Tag Directories. As a result based on what core the data is located, the latency will increase; decreasing bandwidth.

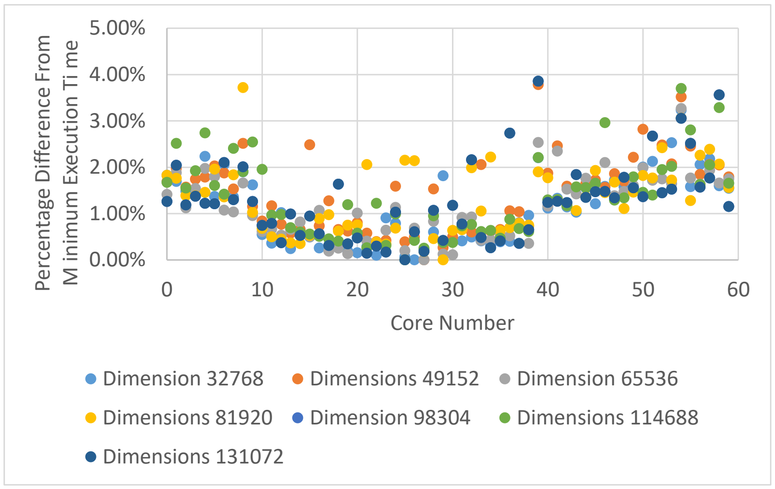

Figure 6.3: Streamcluster KNC Performance With Multiple Inputs

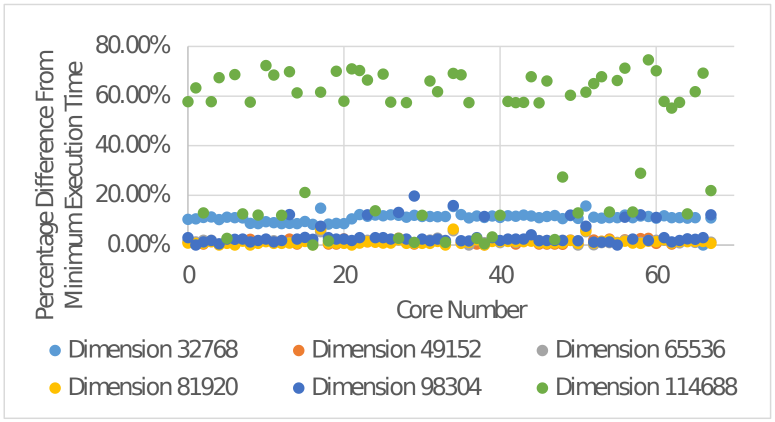

Figure 6.4: Streamcluster KNL Performance With Multiple Inputs

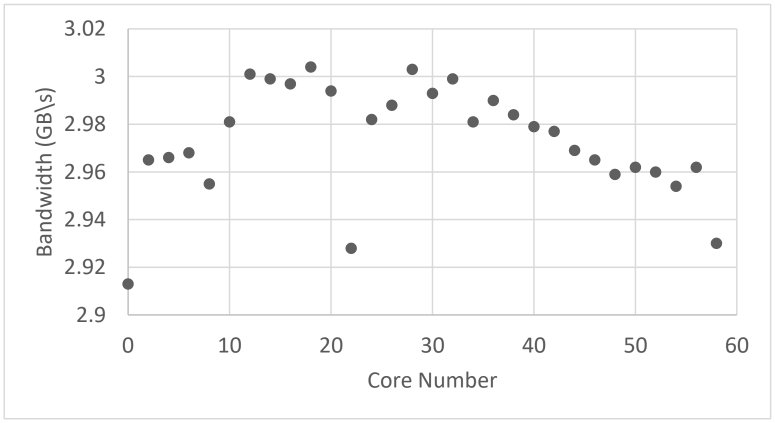

Figure 6.5: Streamcluster Application Bandwidth Per Core on KNC For a Trial

6.2 Needleman-Wunsch

6.2.1 Needleman-Wunsch Performance On KNC and KNL

Figure 6.6 shows the execution of the Needle Wunsch performance with five trials on KNC with the main analysis configuration mentioned in chapter 4. The graph shows a similar shape to the results of the streamcluster graph with the cores in the middle performing faster compared the cores at the edges. The average execution time was found to be 224.94 seconds and the standard deviation for the trials 1.61. We have 30% of the data having execution times under 223 seconds between cores 9 - 38, and 60% of the execution times above 223 seconds of which 73% of these runs came from cores 0 to 8 and 40 to 58.

The execution times for the five trials on the KNL showed and average performance of 44.54 seconds. The standard deviation for the trials was found to be 0.78. We however noticed that cores 17, 34 and 51 consistently performed slower compared to the other cores as can be seen in Figure 6.7 with times of 45.84 seconds, 47.87 seconds and 45.83 seconds respectively (average 2.96%). However the performance

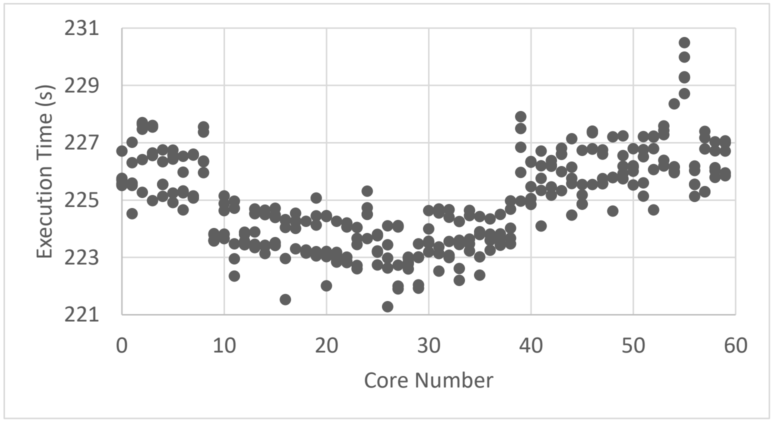

Figure 6.6: NW Application Performance Per Core on KNC

on Core 0 is faster compared to all the other cores with an execution time average of 42.67 seconds or 4.2% faster.

6.2.2 Needleman-Wunsch Core Variation Scale

Figure 6.8 and 6.9 shows the results of the NW application on KNC and KNL respectively, as we scale the input sizes from a dimension of 12288 to 28672. KNC graph shows that the execution times for cores performing in the middle is still lower as we scale the input sizes. The peercentage difference between the fastest executing core and the other cores increases from the middle core outward up to 5%. The graph for KNL shows that as we scale input sizes the core variation observed with our smaller input size also scales with cores 17, 34, 51 performing about 10% slower than other cores on the different input sizes.

6.2.3 Needleman-Wunsch KNC Hardware performance

NW is a dynamic programming algorithm which organize values in a 2d matrix and searches through it to find the optimal alignment. The execution time perfor-

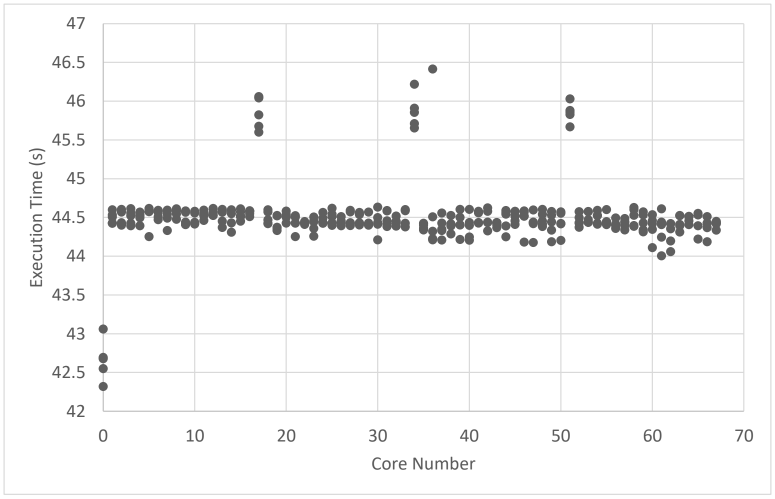

Figure 6.7: NW Application Performance Per Core on KNL

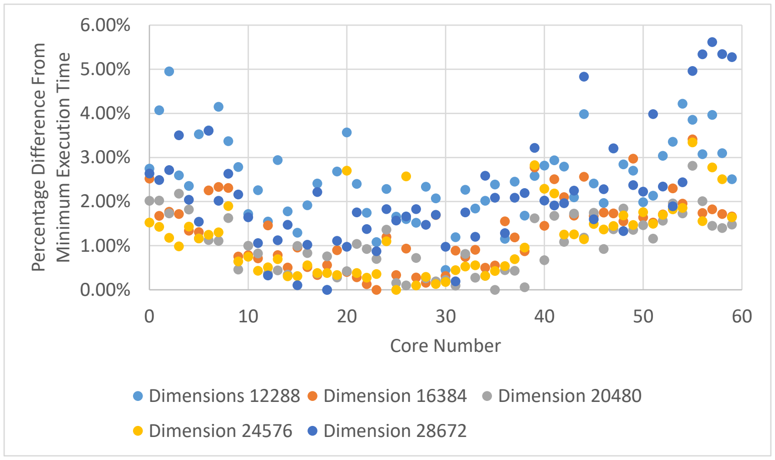

Figure 6.8: NW KNC Performance With Multiple Inputs

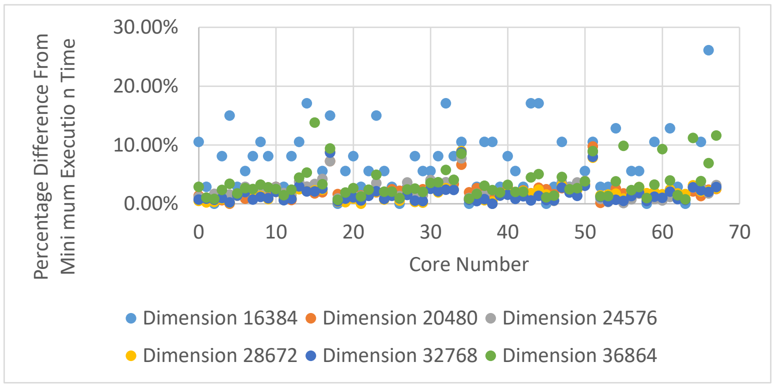

Figure 6.9: NW KNL Performance With Multiple Inputs

mance showed that the application works better in the middle cores and not good at the end. Figure 6.11 shows the CPI performance of the application on all cores of the KNC on a sample trial. This is a high CPI value which shows that there is high latency when this application is being executed. Figure 6.10 shows the memory bandwidth of the application on each core. This graph is shaped identical to the observed performance graph for the five trials. This increased bandwidth in the middle cores directly contribute to the performance of inner cores being better as this dynamic programming algorithm have large recursive memory accesses. Hence the cache coherence architecture of the KNC leads to the same effect of inner cores performing faster, as was shown with the streamcluster application.

6.3 Myocyte

6.3.1 Myocyte Performance On KNC and KNL

Figure 6.12 shows the execution time performance of the Myocyte application on the KNC's. The average execution time was found to be 41.2 seconds and the standard deviation found to be 1.54. there were no notable observations from this

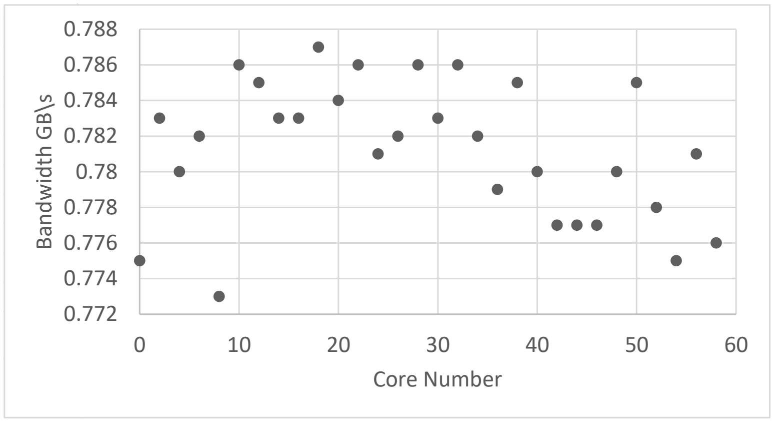

Figure 6.10: NW bandwidth Performance Per Core on a sample trial run on KNC

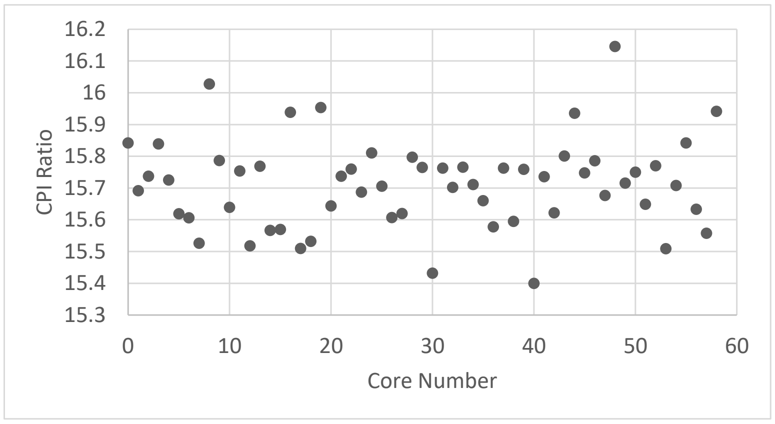

Figure 6.11: NW CPI Performance Per Core on KNC

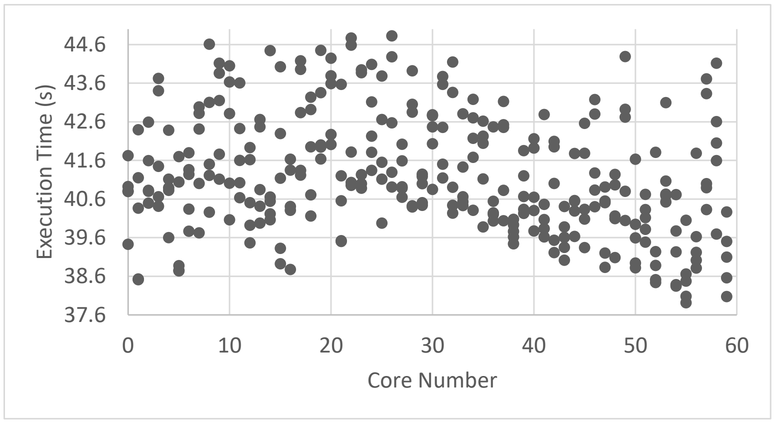

Figure 6.12: Myocyte Application Performance Per Core on KNC

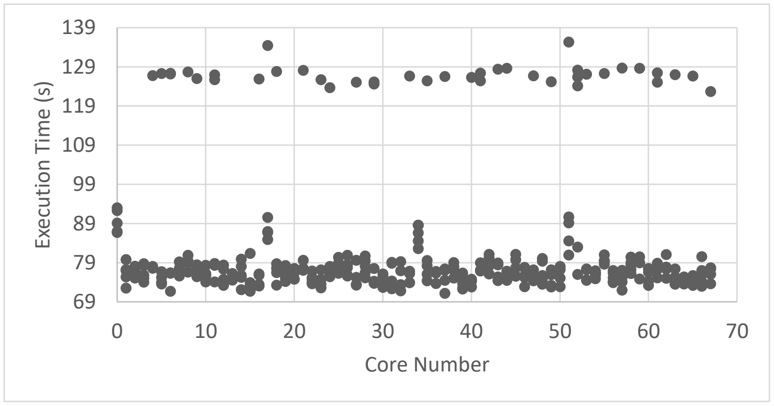

Figure 6.13: Myocyte Application Performance Per Core on KNL

Figure. Performance per trial was randomly distributed between 37 and 45 seconds.

In this particular application, the average execution times for KNL was 50% slower at 82.79 seconds than the execution time on KNC. The standard deviation was found to be 16.49. Figure 6.13 represents the execution time performance of the application which shows cores 0, 17, 34, 51 performing slower compared to the others with an average performance difference of 5%. A constant performance outlier with an average execution time of 126 seconds can also be seen for 8% of our total results.

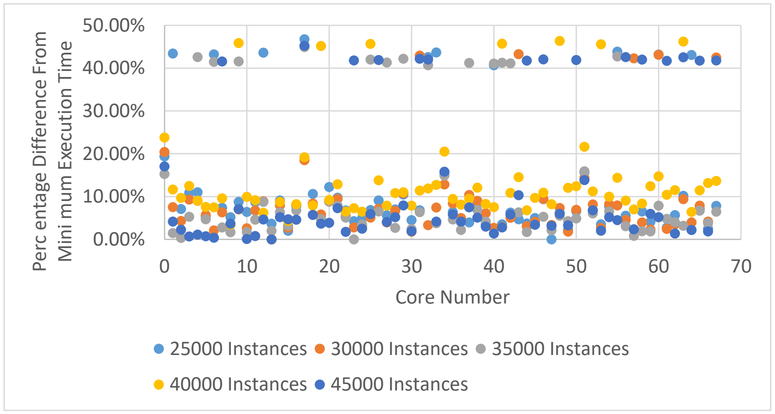

Figure 6.14: Percentage Execution time Difference From Minimum Execution Time for Myocyte on KNL

6.3.2 Myocyte Core Variation Scale

Figure 6.14 shows how scaling the input sizes on KNL affect the variation between the cores. We scale the input sizes from 25000 to 45000 instances and the variation was constant over all the sizes. We noticed cores 0, 17, 34 and 51 performed around 20% slower than the minimum execution time. We also noticed the same random occurrence of cores performing drastically slower than average execution time; which we saw to be around 40% slower than the fastest execution time.

6.3.3 Myocyte KNC Hardware performance

This application is used to model numerous differential equations in a sequential staged solution. Most of the execution of this program is spent in reading in inputs and the bandwidth observe was fairly constant around 0.3 GB/s. Other hardware factors such as number of instructions and clockticks taken to execute these instructions were directly proportional. Figure 6.15 shows a sample trial execution with Figure 6.16 showing its CPI for each core. This shows that ratio of instructions being retired and amount of clockticks is performing fairly constant. As a result of

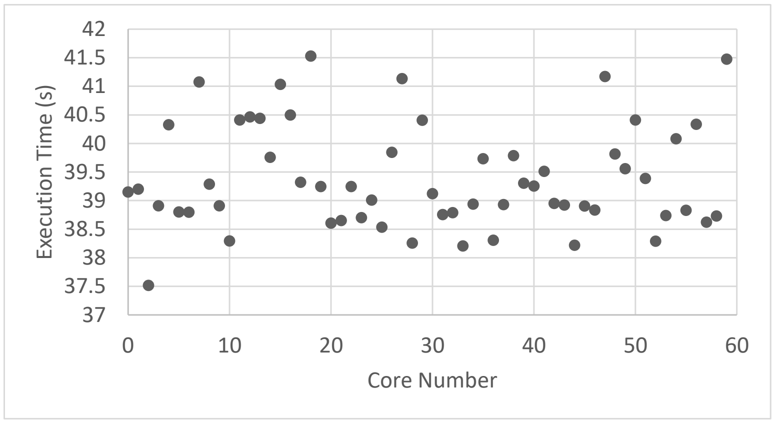

Figure 6.15: Myocyte Execution Performance Per Core on KNC

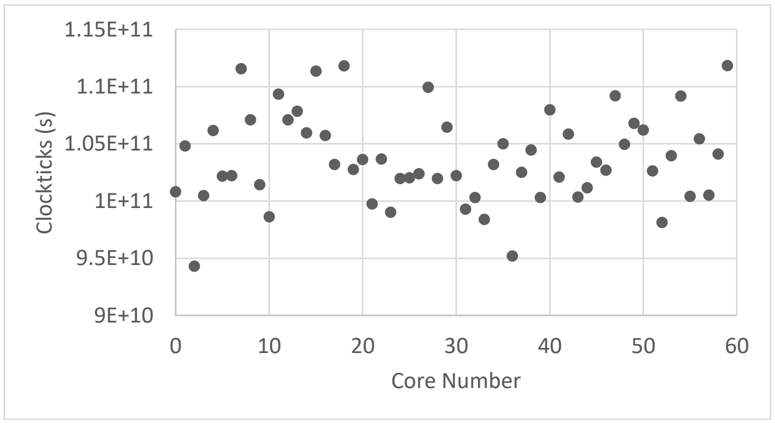

Figure 6.16: Myocyte Clocktick Performance Per Core on KNC

no other hardware factors having a substantial variation; this leads to a somewhat consistent performance.

6.4 Matrix Multiplication

6.4.1 Matrix Multiplication Performance On KNC and KNL

Figure 6.17 shows the performance of Matrix multiplication on KNC. We can see a constant threshold of performance for the cores between 50 and 63 seconds.

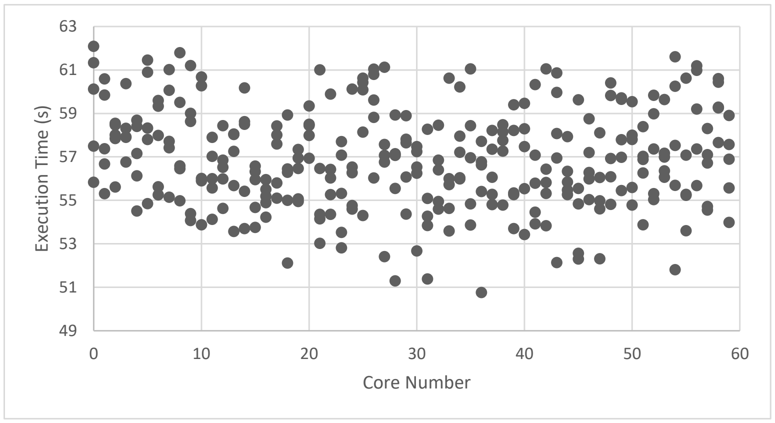

Figure 6.17: Matrix Multiplication Application Performance Per Core on KNC

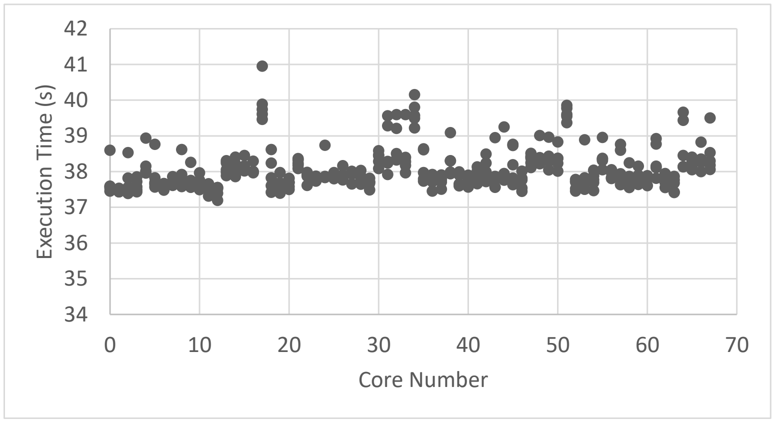

Figure 6.18: Matrix Multiplication KNL Application Performance Per Core on KNL

The average execution time was found to be 56.9 seconds with standard deviation of 2.276.

Figure 6.18 shows the performance of the application on the cores of the KNL. The average execution time on the KNL was 33.5% faster than KNC with an average execution time of 38.07 seconds. We observed than on this application only cores 17, 34 and 51 had a notable performance difference with their average execution times being 4.67% 3.98% and 3.94% slower than the average performance time of 38.07 seconds. The standard deviation found for the results on KNL was 0.57.

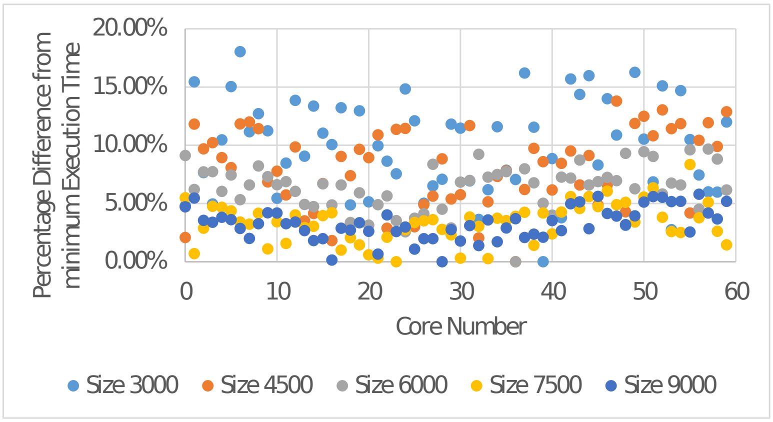

Figure 6.19: Matrix Multiplication KNC Performance With Multiple Inputs

6.4.2 Matrix Multiplication Core Variation Scale

Figure 6.19 shows the variability among KNC cores as we increased the input size for the matrix multiplication. For each input size the performance difference from the fastest executing core was within a constant threshold for all cores. It was however notice that as we increase the input size the performance difference decreased. The average performance difference was 9.80%, 8.11%, 6.40%, 3.35% and 3.28% for input sizes 3000, 4500, 6000 and 9000 respectively.

The results for the sparse matrix to matrix multiplication on KNL scaled with input sizes between 3000 and 9000 as can be seen in Figure 6.20. Performance difference on cores 17, 34 and 51 were observed with all input sizes with the execution time being around 6% slower. The percentage difference was constant at an average of 1.21% unlike the decreasing trend observed in the KNC.

6.4.3 Matrix Multiplication KNC Hardware performance

Figure 6.21 and Figure 6.22 shows the CPI and execution time of a trial of the matrix multiplication application. The implementation will have some latency

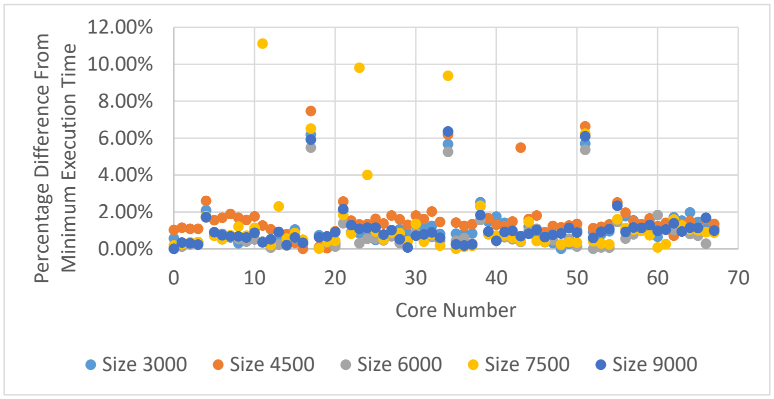

Figure 6.20: Matrix Multiplication KNL Performance With Multiple Inputs

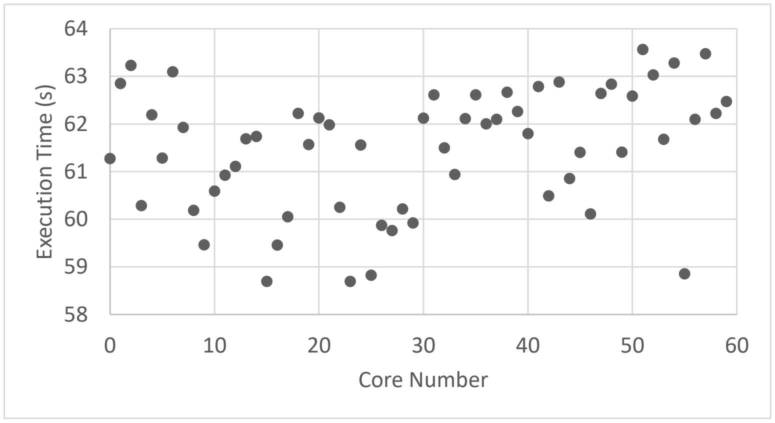

Figure 6.21: Matrix Multiplication Execution Performance Per Core of a Trial Run on KNC

factors as is suggested by the high CPI Rate between 6.5 and 7. We however notice that the CPI rate trend on the cores correlates with the trend of the execution time performance. This highlights that when number of instructions is directly related to frequency at which the core is performing; the core performance is more consistent. This leads to other architectural elements such as the bus interconnect; having a reduced impact on performance.

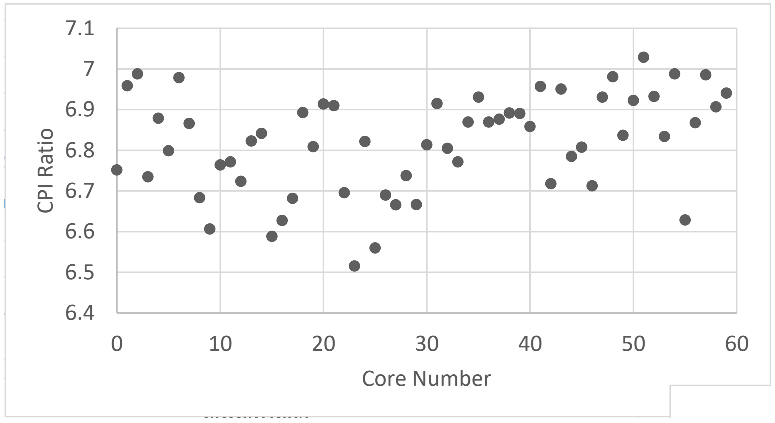

Figure 6.22: Matrix Multiplication CPI Performance Per Core of a Trial run on KNC

6.5 Speckle Reducing Anisotropic Diffusion

6.5.1 Speckle Reducing Anisotropic Diffusion Performance On KNC and KNL

Figure 6.23 shows that the execution time performance of the SRAD on the Xeon phi was relatively constant between 120 seconds and 128 seconds. The standard deviation of the execution times was found to be 1.64. The average execution time for the performance on this application was found to be 123 seconds.

Similar results can also be see in Figure 6.24 showing the execution times of SRAD on the KNL. The standard deviation of SRAD performance was found to be 0.54 and has an average execution time of 70.88 seconds. Unlike the other applications with variation on cores 0, 17, 34 and 51; this application only exhibits performance variation on core 0. Core 0 performed 3.2% faster than the average performance of the other cores.

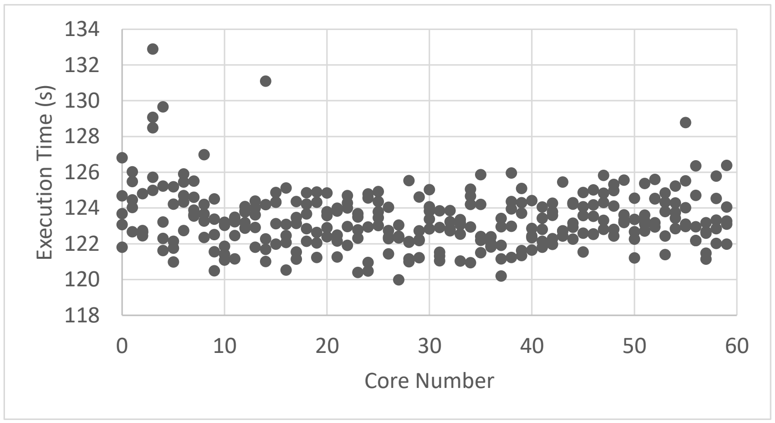

Figure 6.23: SRAD Application Performance Per Core on KNC

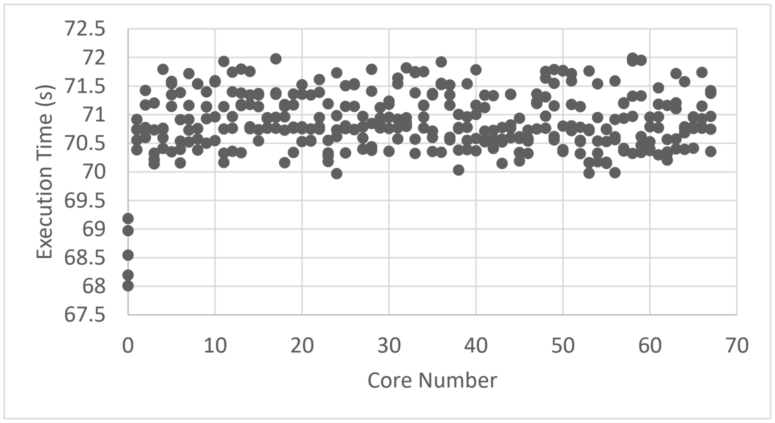

Figure 6.24: SRAD Application Performance Per Core on KNL

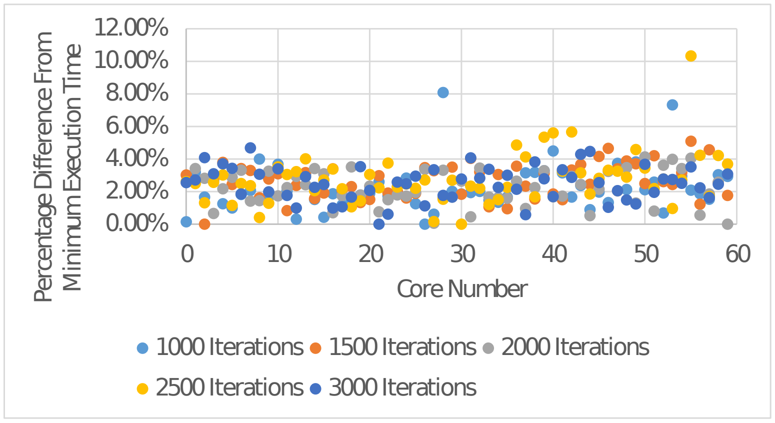

Figure 6.25: SRAD KNC Performance With Multiple Input

6.5.2 Speckle Reducing Anisotropic Diffusion Core Variation Scale

As we scaled the input sizes for the SRAD on the KNC, the performance difference remained under 4% slower than the fastest trial. This can be seen in Figure 6.25 an it shows that the constant performance observed in 5 trials holds for any input sizes.

Figure 6.26 shows that the performance of the SRAD application is constant among all cores except 0 as we scale the input sizes. We scaled the input from 1000 iterations to 3000 and core 0 performed between 2% and 4% faster than the other cores in executing the application.

6.5.3 Speckle Reducing Anisotropic Diffusion KNC Hardware performance

SRAD is structured grid algorithm that a wider array of image manipulations in sequential stages. Based on the nature of image processing, memory affects the execution of the SRAD algorithm out of all the application. This application had

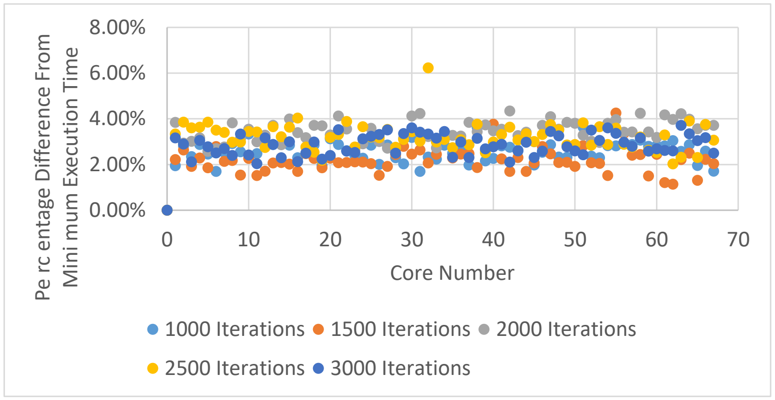

Figure 6.26: Percentage Execution time Difference From Minimum Execution Time For SRAD

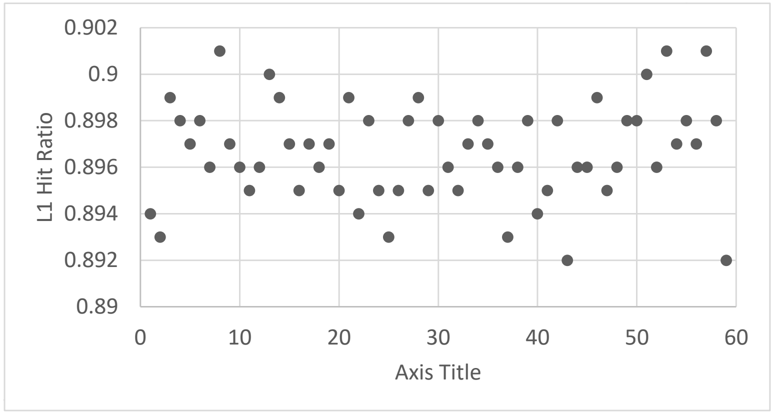

Figure 6.27: SRAD L1 Cache Performance Per Core on KNC

the most variation and lowest L1 hit rates among all the applications as can be seen in Figure 6.27. It also had the lowest latency impact with an average of 190 as can be seen in Figure 6.28. There where no observable trends regarding the Bandwidth. The combination of memory access to cache and synchronization lead to a constant bottleneck on performance leading to the cores performing relatively the same.

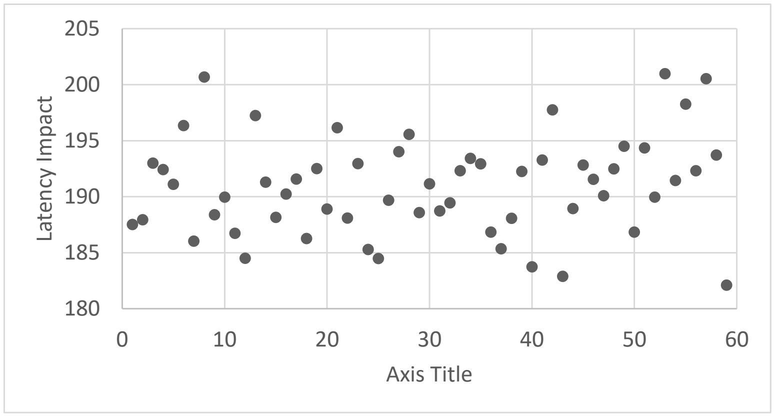

Figure 6.28: SRAD L2 Cache Performance Per Core on KNC

6.6 LavaMD

6.6.1 LavaMD Performance On KNC and KNL

Figure 6.29 shows the execution time of the LavaMD application on the different cores on the KNC. We found the average execution time to be 56.95 seconds for the cores with a standard deviation of 0.783. The Figure also shows than there are random outliers throughout the trials within the same performance range of 58.5 seconds.

Figure 6.30 shows the execution time performance of the lavamd application on the KNL. This application has an average performance of 44.5 seconds however core 0 performs 4% faster than this average. Cores 17, 34, 51 are also distinctively with them performing 2.91%, 2.97% and 2.94% slower than the other cores. The standard deviation for the execution of this application on the KNL's was found to be 0.783.

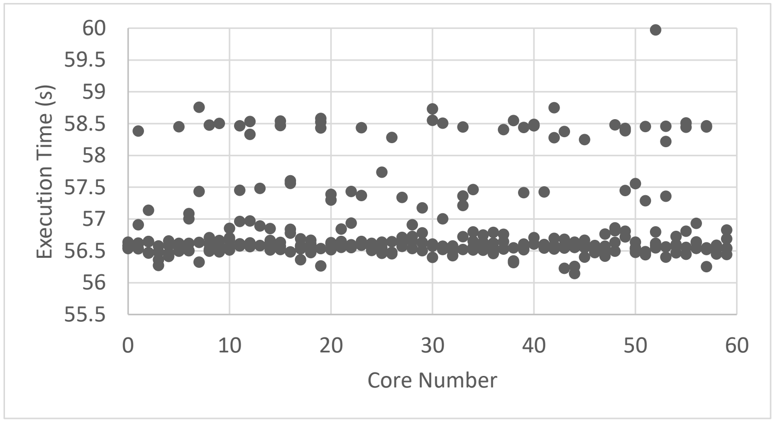

Figure 6.29: Lavamd Application Performance Per Core on KNC

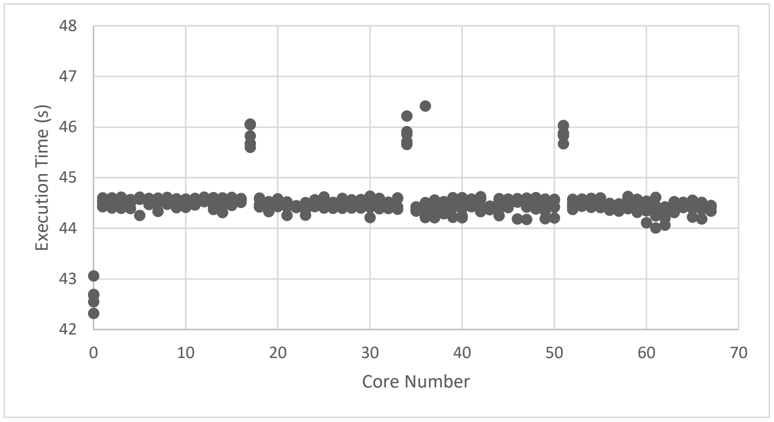

Figure 6.30: Lavamd Application Performance Per Core on KNL

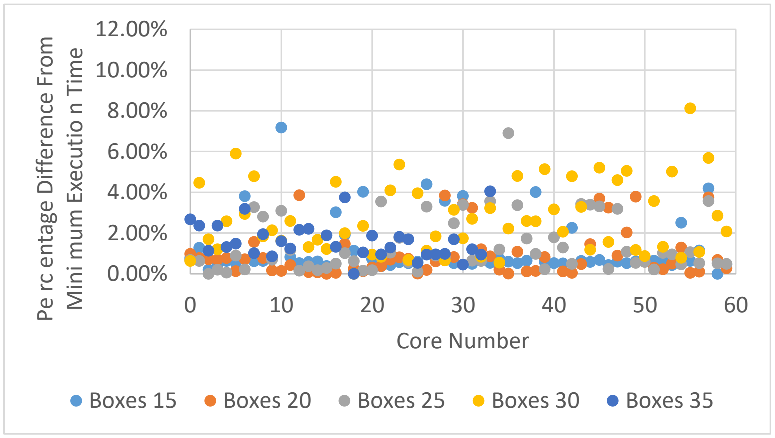

Figure 6.31: Percentage Execution time Difference From Minimum Execution Time on KNC

6.6.2 LavaMD Core Variation Scale

Figure 6.31 and Figure 6.32 shows the performance of the Lavamd application as we scaled the input size from 15 boxes to 30 on the KNC and KNL respectively. The performance difference between cores scaled proportional on both KNC and KNL. The kNC showed performance variation below 8% with majority of the cores performing close to the fastest execution time. The KNL showed performance variation below 11%. Majority of the cores had a performance time difference of 4% from best execution times which were on core 0. Cores 17, 34 and 51 were approximately 6% slower than the fastest execution time.

6.6.3 LavaMD KNC Hardware performance

Lavamd is a nbody application which is a high flops computational algorithm. Figure **??** shows the execution time and Figure 6.34 shows the CPI rate of a sample execution of the application on each core. This correlation between CPI and execution leads to more consistent performance as the instructions executed directly relates to

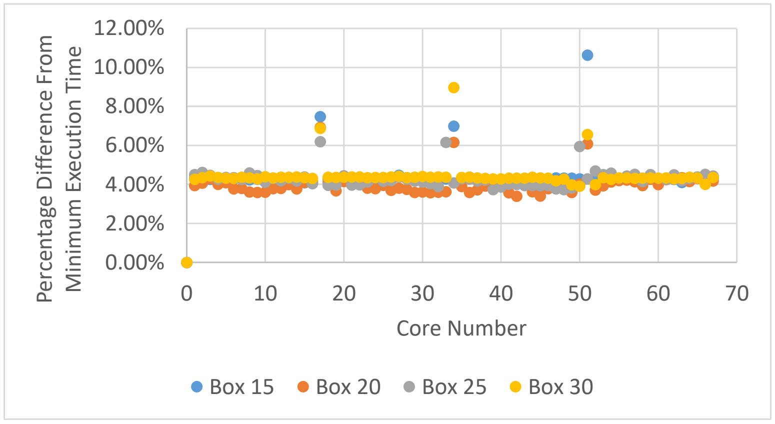

Figure 6.32: Percentage Execution time Difference From Minimum Execution Time For Lavamd on KNL

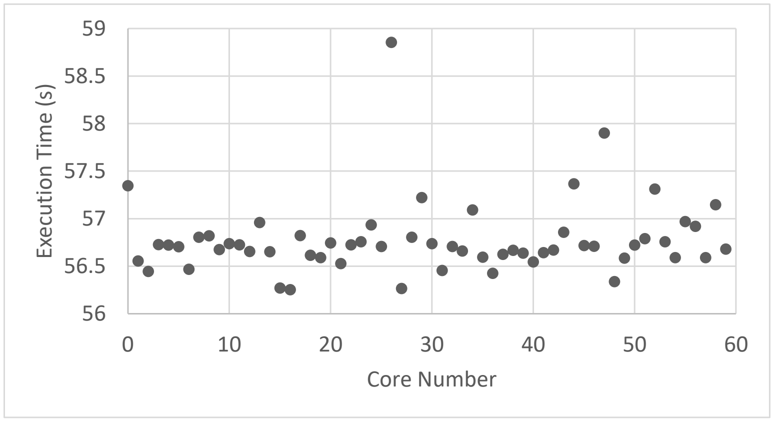

Figure 6.33: Lavamd Execution Performance Per of a trial Core on KNC

operating frequency of the core. Other architectural features such as bandwith, cache access were relatively constant.

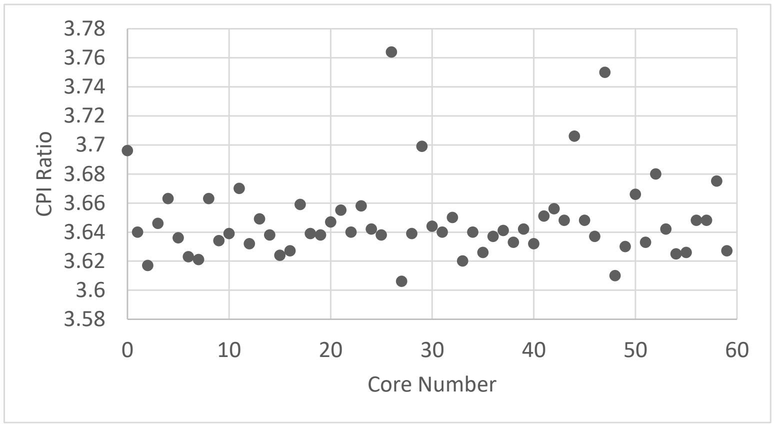

Figure 6.34: Lavamd CPI Performance Per Core of a trial on KNC

6.7 Particle Filter

6.7.1 Particle Filter Performance On KNC and KNL

Figure 6.35 shows the execution time performance of the particle filter on the KNC. This application performed very constant over all the cores with a standard deviation of 0.268. The average execution time for this application was 58.63 seconds which was 20.45% slower than the average execution time performance on the KNL's.

The performance per core on the KNL's had a similar trend as the Lavamd and NW application as cane be seen in Figure 6.36. The first core performs 1.8% faster than all the cores. Cores 17, 34 and 51 performed 3.4%, 1.78%, 1.82% slower than all the cores. The standard deviation of the execution time performance on the KNL was 1.22 which was higher than KNC due to the previously noted cores performance variability.

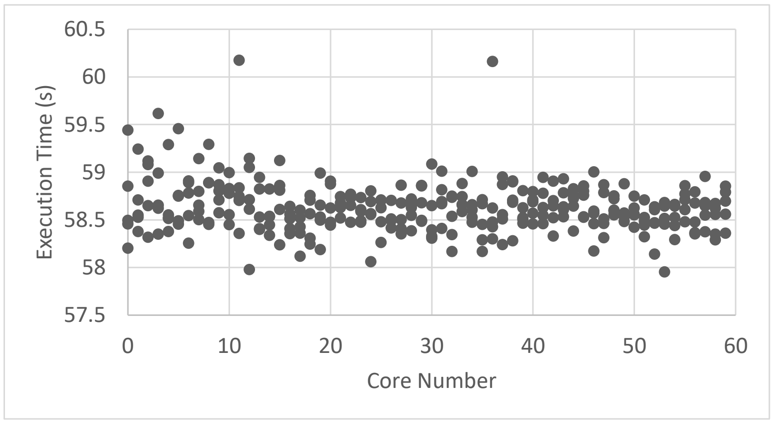

Figure 6.35: Particle Filter Application Performance Per Core on KNC

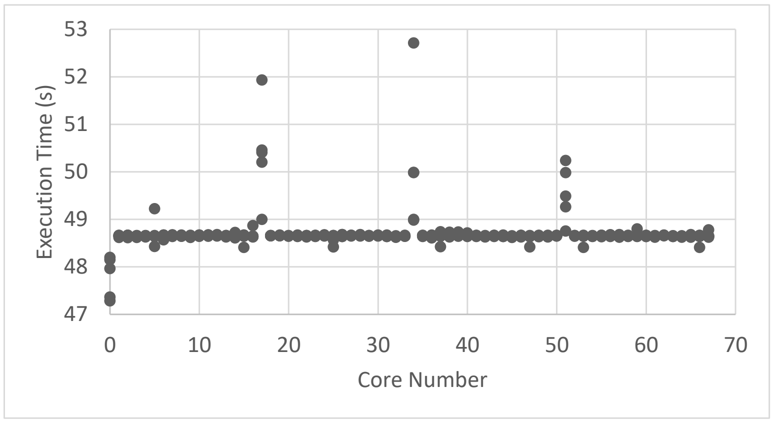

Figure 6.36: Particle Filter Application Performance Per Core on KNL

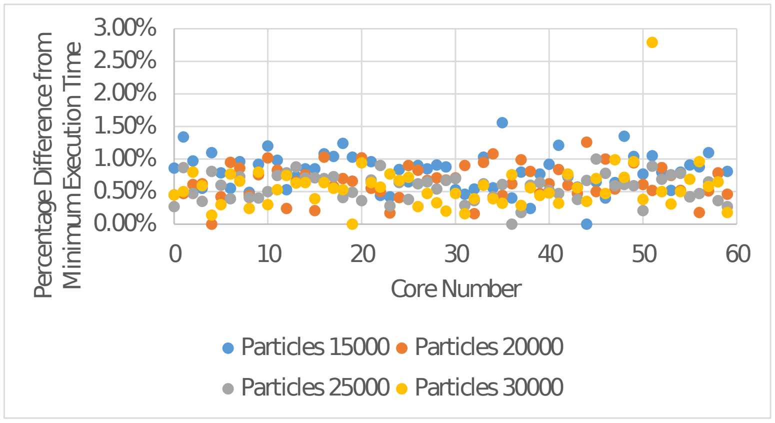

Figure 6.37: Particle Filter KNC Performance With Multiple Inputs

6.7.2 Particle Filter Core Variation Scale

Figure 6.37 shows the performance of the particler filter application as we scale the input sizes. In our 5 trials that we used we noticed that the execution time performance was fairly constant with no distinctive performance variations. This is also seen per input size as the percentage difference from the fastest time is within a particular threshold. We however notice that as the input sizes increase the percentage difference from the fastest executing core is gradually decreasing. As the input sizes increases the percentage difference were 0.81%, 0.65%, 0.57% and 0.57%.

We also notice a similar trend with decreasing percentage difference as we scale our input on the KNL. Figure 6.38 shows the results when we scale the input on KNL's. We can observe the same trend with core) performing fastest and cores 17, 34, 51 consistently slow at all the input sizes. From taking the averages of the performance difference from minimum execution time we saw the reduction in this percentage as we scaled up. The average performance difference were 4.41%, 2.85%, 1.93% and 1.77% for input sizes of 15000, 2000, 25000 and 30000 respectively.

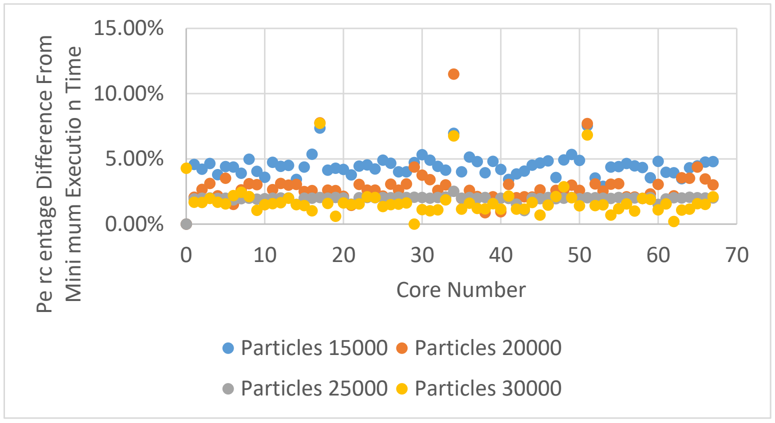

Figure 6.38: Particle Filter KNL Performance With Multiple Inputs

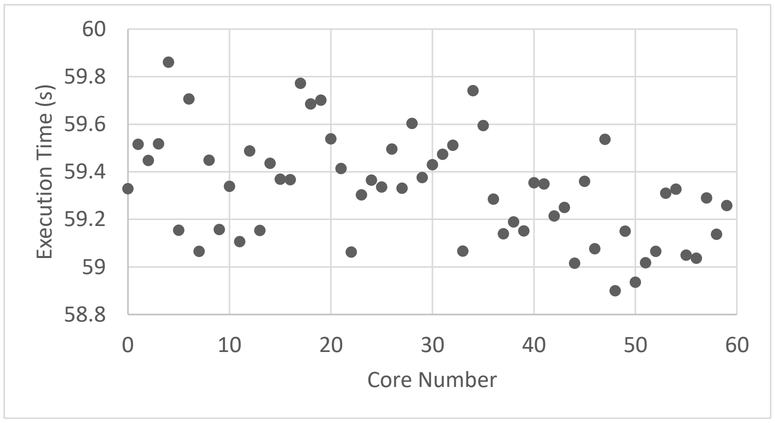

Figure 6.39: Particle Filter Execution Time Performance Per Core on KNC

6.7.3 Particle Filter KNC Hardware performance

Particle filter is a dynamic programming algorithm which does a lot of computation in comparison to the NW application. Figure 6.39 and Figure 6.40 shows the execution time and CPI performance respectively of a sample trial. The cores performance is randomly distributed within a time range and the CPI correlates with these performance. This means that the operating frequency is proportional to the rate at which instructions are being retired.

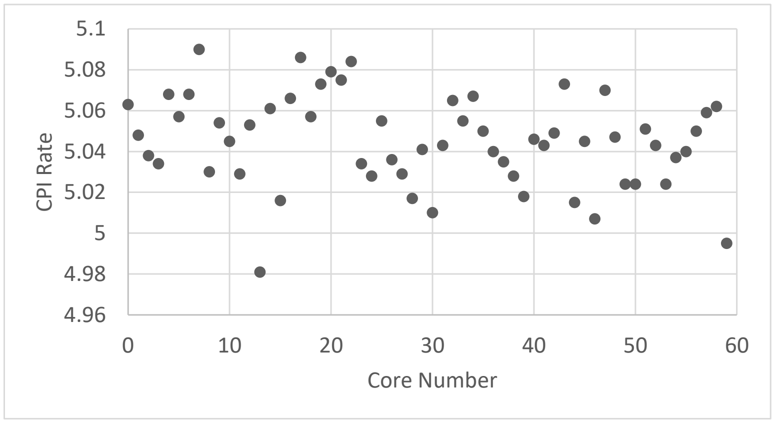

Figure 6.40: Particle Filter CPI Performance Per Core on KNC

6.8 Breadth First Search

6.8.1 Breadth First Search Performance On KNC and KNL

Figure 6.41 shows the performance execution time of Breadth First Search application on KNC. The execution time performance is constant over all the cores in the KNC with an average execution time of 186.71 seconds and a standard deviation of 3.30.

The performance was similar for the BFS application on the KNL as can be seen in Figure 6.42. It had an execution average time of 55.94 seconds with an average standard deviation of 4.85.

6.8.2 Breadth First Search KNC Hardware performance

Figure 6.43 and Figure 6.44 shows the execution time and CPI performance of a trial run on the KNC. This application has a direct correlation between operating frequency and the number of instructions to be retired. This leads to a constant execution time range and other hardware components such as L1 and L2 hit ratio,

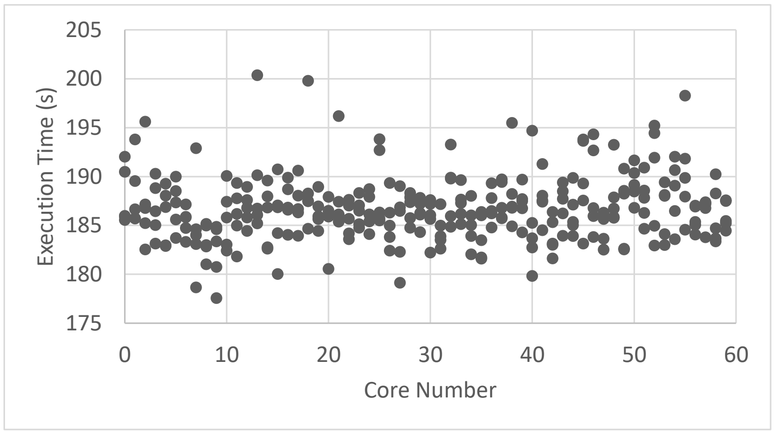

Figure 6.41: Breadth First Search Application Performance Per Core on KNC

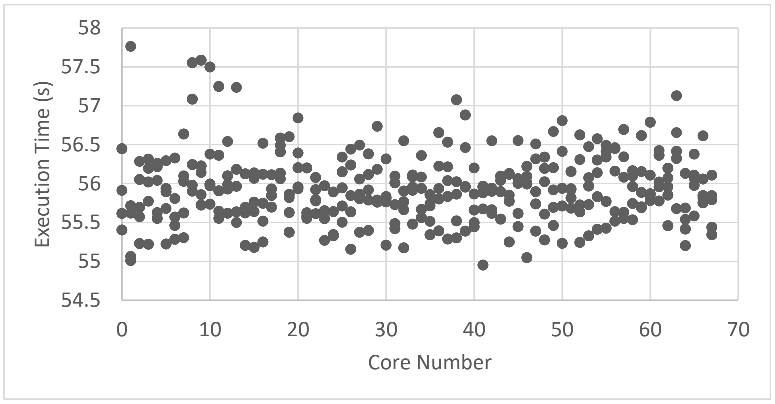

Figure 6.42: Breadth First Search Application Performance Per Core on KNL

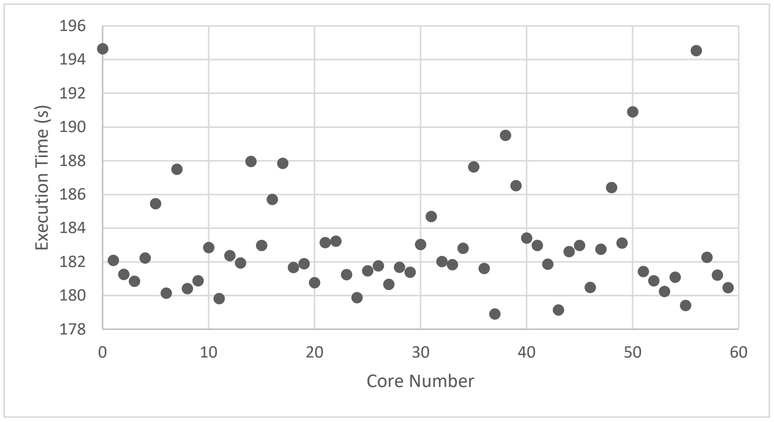

Figure 6.43: BFS Execution Performance Per Core on KNC

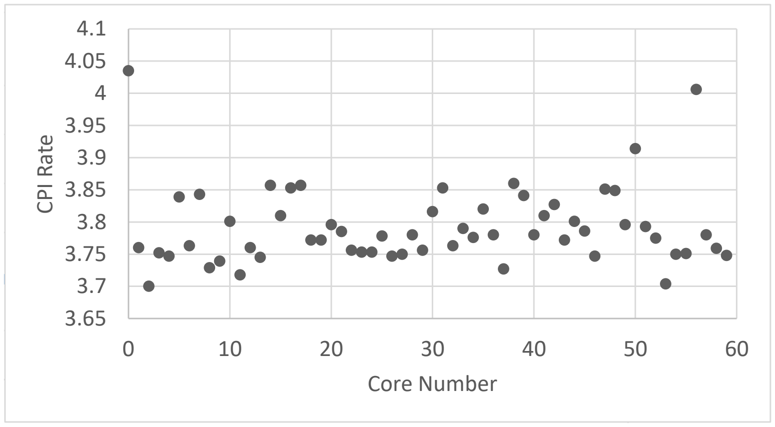

Figure 6.44: BFS CPI Performance Per Core on KNC

bandwidth remaining constant.

6.9 KNC Multi Core Xeon Phi Power and Performance

After analyzing all the application performance on the Xeon phi KNC; we noticed that majority of the application had a somewhat constant performance with only two application distinctively exhibiting performance degradation going out ward from cores 10 to 40. Utilizing this knowledge we executed a series of power and performance trials on the applications utilizing various core combinations. We choose core combinations of 1 - 60 (base configuration), 10 - 40 (middle cores) and 1 - 9 + 41 - 60(outer cores). We compared our latter two trials execution time and power consumed to our core combination of 1 - 60. In each core configuration we choose the optimal thread configuration to give us the best execution time and the results are shown in table 6.1. The applications BFS, NMyocyte and Needle were not included in the results because the core utilization was under 5 percent for these applications. This low core utilization lead to the power consumed being constant with the system power consumption due to lack of core usage.

Our results showed we had a 16.4% and 15.6% energy reduction while using middle cores, 14.9% and 11.6% energy reduction while using outer cores for the application streamcluster and particle filter respectively. The energy performance savings was expected for the streamcluster application based on the KNC trial results results. For the Particle filter application this was however not expected as the results showed constant performance when we did our trials. This result may due to the fact that the particle filter is a dynamic programming algorithm like the NW application. The NW application did show similar trial results to streamcluster. The difference between the NW and particle filter application is that there is a higher amount of computation and core utilization with particle filter than with NW. As a result of this the per core trial did not show the performance variation due to the high computation. However When the application is however ran on many cores it exhibits the energy saving characteristics but not the performance improvement due computation intensity.

The applications Matrix multiplication, LavaMD and SRAD performed within a 2% energy consumption difference for the outer and middle core configurations. The application performance for Streamcluster, Matrix Multiplication, and SRAD were

Table 6.1: Performance and Power consumption for Applications on KNC

Application	Cores	Thread	Power (Watts)	Execution Time (s)	Percent Energy Efficiency	Percentage Performance Difference
LavaMD	1 - 60	244	55485	160.937607	0%	0%
	10 - 40	120	55516	284.913	0%	44%
	1 - 9, 41 - 60	120	56220	287.424	1%	44%
Matrix Multiplication	1 - 60	244	139612	271.630308	0%	0%
	10 - 40	120	138586	285.145608	-1%	5%
	1 - 9, 41 - 60	120	140313	286.716225	0%	5%
Particle Filter	1 - 60	120	21781	111.21714	0%	0%
	10 - 40	120	18836	212.6887	-16%	48%
	1 - 9, 41 - 60	120	18958	217.253967	-15%	49%
StreamCluster	1 - 60	240	56069	224.76294	0%	0%
	10 - 40	120	48179	220.173186	-16%	-2%
	1 - 9, 41 - 60	120	50255	224.15	-12%	0%
SRAD	1 - 60	240	52111	194.96469	0%	0%
	10 - 40	120	51999	195.03065	0%	0%
	1 - 9, 41 - 60	120	53081	196.141647	2%	1%

within 5% of the optimal execution time of our base configuration. The applications Lavamd and particle filter had bad performance results with execution times 44% and 48.5% slower respectively, than our base configuration.

6.10 Summary

This chapter details the results collected after running all the application benchmarks on the Xeon Phi architectures. The architecture features which influenced the performance of the applications in single core and multicore executions are discussed. The information obtained from the KNC results were used to improve energy performance on this architecture and information provided on the influence of applications performance.

Chapter 7

Conclusions and Future Work

7.1 Conclusions

In this research we explore the per core performance of the Xeon Phi Knights Corner (first generation MIC) and the Xeon Phi Knights Landing (second generation MIC). Eight applications from the Rodinia Benchmark were implemented and tested to derive insights into how the MIC cores individually perform on these architectures. The results indicate that application characteristics impact or affect how they perform on a given core; some applications perform fairly consistently across all cores while others exhibit varying results. Lastly, there were some unique characteristics consistently exhibited by certain cores.

On the KNC architecture, the applications Needleman-Wunsch and Stream-Cluster performed better on the middle cores (10 - 40) with the performance gradually decreasing outwards to core 0 and core 60 on either side of the ring. This phenomon was seen throughout our trials and across various input sizes. The performance difference between cores extend to about 5% from the fastest execution time. The cause of this performance was due to the ring architecture of the KNC using Distributed Tag

Directories for cache-to-cache transfer, which enharently leads to reduced bandwidth at the outermost cores.

The applications BFS, Myocyte, LavaMD and Matrix Multiplication all performed fairly consistantly within a given time range. This performance was due to a direct correlation between the retired instructions and the operating frequency at which the core is retiring them. As a result other architectural features of the hardware has a reduced impact, which is observed as the latency impact: l1 hit ratio and bandwidth are fairly constant during the execution of the application.

The Particle Filter and SRAD applications also had a constant execution time within constant time period. Their performance was correlated to the programming implementation on the hardware. The L1 and L2 cache variation performance with and high core utilization flops calculation lead to constant performance. The Particle Filter is also a dynamic programming algorithm as is NW, but the high core utilization and flops calculation lead to the memory not being the determining factor for performance. These results show that there are variations within the KNC cores that are dependent on how the application utilizes the architecture of the KNC.

Utilizing this knowledge we implemented a performance and power execution analysis on the KNC processors using configurations of the middle cores (cores 10 - 40), outer cores (cores 0 - 9, 41 - 59) and base cores (0 - 59). We used all applications except the BFS, Myocyte and NW applications as those had low core utilization leading to very little affects on the power consumed during their execution on many cores. Our results showed about 16% reduction in the energy consumed while having a small impact on the execution time for the Streamcluster application. There was also a similar reduction in the energy consumed for the Particle Filter application but a major performance degrade of about 48%.

On the KNL architecture, three trends in application performance were noted.

The applications BFS and SRAD had core execution times that were constant over the application time; similar to the results from the KNC generation. The only exception being core 0 on the SRAD application performing faster than the other cores. For applications LavaMD, Particle Filter, and NW, core 0 also performed faster than the other cores; however cores 17, 34 and 51 were all consistently slower than the other cores. The applications Matrix Multiplication and Streamcluster had core 0 performing relatively consistant with the other cores while cores 17, 34 and 51 performed significantly slower.

Myocyte however had core 0 performing just as slow as 17, 34 and 51 compared to the other cores. With the above applications on the KNL architecture, the behavior or trends for Myocyte, BFS and NW are interesting based on their algorithmic properties and features observed from the KNC architecture execution. This bahavior may be no coincidence since these applications had extremely low utilization on the KNC architecture.

These results show that cores 0, 17, 34 and 51 on the KNL Xeon Phi has some characteristics or features that lead to interesting variation in performance. Investigation is beyond this research, but may be a result of some additional instructions or features added to these particular nodes to make the KNL architecture capable of the different memory modes in the ring mesh interconnect.

7.2 Future Work

This research uncovered many interesting characteristics and features regarding how the Xeon Phi KNC and KNL processors perform on a per core level. However, much work remains to further understand these architectures in general, and the per core performance power relation and its affect on full core application runs.

During this research, hardware information was collected using VTune but information for the KNL generation was limited due to the inconsistancy of VTune on Stampede. In the future, more hardware results for these architectures are needed to understand why cores 0, 17, 34 and 51 have such performance variations. Other analysis should include details of the applications, to quantify how flops, non-flops and other application characteristics influence the Xeon Phi performance. These parameters can be collected using the PAPI performance API on Stampede and utilized to predict best suited applications using the Tesseract modeling framework [6].

Currently there are only 8 applications evaluated in this research. To better understand the performance across domains, the number of applications should be increased to obtain more variety and more accurate information of how aspects of different applications map to these architectures.

An analysis of power and performance behaviours utilizing all cores except 0, 17, 34 and 51 on the KNL generation compared to utilizing all cores on the KNL would also shed light on the different behaviour of these cores, providing insight onto whether the results from the KNC power performance study was mainly due to architecture or with application affects on hardware. More detailed analysis should also reveal how core variations impact performance on the different memory modes and cluster modes of the KNL architecture.

Bibliography

[1] Blue waters user portal — #welcome. https://bluewaters.ncsa.illinois.edu/. (Accessed on 05/02/2017).

[2] Cab — high performance computing. https://hpc.llnl.gov/hardware/platforms/cab. (Accessed on 05/02/2017).

[3] Edison. http://www.nersc.gov/users/computational-systems/edison/. (Accessed on 05/02/2017).

[4] Intel xeon phi x100 family coprocessor - the architecture — intel software. https://software.intel.com/en-us/articles/intel-xeon-phi-coprocessor-codename-knights-corner. (Accessed on 05/02/2017).

[5] Optimization and performance tuning for intel xeon phi coprocessors, part 2: Understanding and using hardware events — intel https://software.intel.com/en-us/articles/ software. optimization-and-performance-tuning-for-intel-xeon-phi-coprocessors-part-2-understanding.(Accessed on 05/03/2017).

[6] sc15.supercomputing.org/sites/all/themes/sc15images/doctoral showcase/doc files/drs117s2-file6.pdf. http://sc15.supercomputing.org/sites/all/themes/SC15images/doctoral_showcase/doc_files/drs117s2-file6.pdf. (Accessed on 05/04/2017).

[7] system-administration-for-the-intel-xeon-phi-coprocessor.pdf. https://software.intel.com/sites/default/files/article/373934/system-administration-for-the-intel-xeon-phi-coprocessor.pdf. (Accessed on 05/03/2017).

[8] Tacc stampede user guide - tacc user portal. https://portal.tacc.utexas.edu/user-guides/stampede. (Accessed on 05/02/2017).

[9] title. http://www.hotchips.org/wp-content/uploads/hc_archives/hc27/HC27.25-Tuesday-Epub/HC27.25.70-Processors-Epub/HC27.25.710-Knights-Landing-Sodani-Intel.pdf. (Accessed on 05/03/2017).

[10] xeon-phi-coprocessor-system-software-developers-guide.pdf. `http://www.intel.la/content/dam/www/public/us/en/documents/product-briefs/xeon-phi-coprocessor-system-software-developers-guide.pdf`. (Accessed on 05/03/2017).

[11] B. Acun and L. V. Kale. Mitigating processor variation through dynamic load balancing. In *2016 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pages 1073–1076, May 2016.

[12] Bilge Acun, Phil Miller, and Laxmikant V. Kale. Variation among processors under turbo boost in hpc systems. In *Proceedings of the 2016 International Conference on Supercomputing*, ICS '16, pages 6:1–6:12, New York, NY, USA, 2016. ACM.

[13] Shuai Che, Michael Boyer, Jiayuan Meng, David Tarjan, Jeremy W. Sheaffer, Sang-Ha Lee, and Kevin Skadron. Rodinia: A benchmark suite for heterogeneous computing. In *Proceedings of the 2009 IEEE International Symposium on Workload Characterization (IISWC)*, IISWC '09, pages 44–54, Washington, DC, USA, 2009. IEEE Computer Society.

[14] Shuai Che, Michael Boyer, Jiayuan Meng, David Tarjan, Jeremy W. Sheaffer, and Kevin Skadron. A performance study of general-purpose applications on graphics processors using {CUDA}. *Journal of Parallel and Distributed Computing*, 68(10):1370 – 1380, 2008. General-Purpose Processing using Graphics Processing Units.

[15] Shuai Che, Jeremy W. Sheaffer, Michael Boyer, Lukasz G. Szafaryn, Liang Wang, and Kevin Skadron. A characterization of the rodinia benchmark suite with comparison to contemporary cmp workloads. In *Proceedings of the IEEE International Symposium on Workload Characterization (IISWC'10)*, IISWC '10, pages 1–11, Washington, DC, USA, 2010. IEEE Computer Society.

[16] A. Heinecke, K. Vaidyanathan, M. Smelyanskiy, A. Kobotov, R. Dubtsov, G. Henry, A. G. Shet, G. Chrysos, and P. Dubey. Design and implementation of the linpack benchmark for single and multi-node systems based on intel x00ae; xeon phi coprocessor. In *2013 IEEE 27th International Symposium on Parallel and Distributed Processing*, pages 126–137, May 2013.

[17] Sebastian Herbert and Diana Marculescu. Characterizing chip-multiprocessor variability-tolerance. In *Proceedings of the 45th Annual Design Automation Conference*, DAC '08, pages 313–318, New York, NY, USA, 2008. ACM.

[18] Gary Lawson, Vaibhav Sundriyal, Masha Sosonkina, and Yuzhong Shen. Modeling performance and energy for applications offloaded to intel xeon phi. In *Proceedings of the 2Nd International Workshop on Hardware-Software Co-Design*

for High Performance Computing, Co-HPC '15, pages 7:1–7:8, New York, NY, USA, 2015. ACM.

[19] S. Li, K. Raman, and R. Sasanka. Enhancing application performance using heterogeneous memory architectures on a many-core platform. In *2016 International Conference on High Performance Computing Simulation (HPCS)*, pages 1035–1042, July 2016.

[20] Barry Rountree, Dong H. Ahn, Bronis R. de Supinski, David K. Lowenthal, and Martin Schulz. Beyond DVFS: A first look at performance under a hardware-enforced power bound. In *26th IEEE International Parallel and Distributed Processing Symposium Workshops & PhD Forum, IPDPS 2012, Shanghai, China, May 21-25, 2012*, pages 947–953, 2012.

[21] Enzo Rucci, Carlos García, Guillermo Botella, Armando De Giusti, Marcelo R. Naiouf, and Manuel Prieto-Matías. First experiences optimizing smith-waterman on intel's knights landing processor. *CoRR*, abs/1702.07195, 2017.

[22] Subhash Saini, Haoqiang Jin, Dennis Jespersen, Samson Cheung, Jahed Djomehri, Johnny Chang, and Robert Hood. Early multi-node performance evaluation of a knights corner (knc) based nasa supercomputer. In *Proceedings of the 2015 IEEE International Parallel and Distributed Processing Symposium Workshop*, IPDPSW '15, pages 57–67, Washington, DC, USA, 2015. IEEE Computer Society.

[23] Dirk Schmidl, Tim Cramer, Sandra Wienke, Christian Terboven, and Matthias S. Müller. *Assessing the Performance of OpenMP Programs on the Intel Xeon Phi*, pages 547–558. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.

[24] H. Wang, H. Chen, Q. Wu, J. Lin, X. Chen, X. Xie, R. Wang, X. Tang, and Z. Wang. Accelerating the global nested air quality prediction modeling system (gnaqpms) model on intel xeon phi processors. *Geoscientific Model Development Discussions*, 2017:1–18, 2017.

[25] Cheng Zhang, Li Liu, Ruizhe Li, and Guangwen Yang. *Performance Characterization and Optimization for Intel Xeon Phi Coprocessor*, pages 16–33. Springer International Publishing, Cham, 2015.