Minimization of Xeon Phi Core Use with Negligible Execution Time Impact

Roberto Camacho
Barranco
University of Texas at El Paso
500 W. University Avenue
El Paso, TX 79968-0518
rcamachobarranco@utep.edu

Patricia J. Teller
University of Texas at El Paso
500 W. University Avenue
El Paso, TX 79968-0518
pteller@utep.edu

Michael Gerndt
Technische Universität
München
Boltzmannstr. 3, D-85748
Garching, Germany
gerndt@in.tum.de

ABSTRACT

For many years GPUs have been components of HPC clusters (Titan and Piz Daint), while only in recent years has the Intel® Xeon Phi™ been included (Tianhe-2 and Stampede). For example, GPUs are in 14% of systems in the November 2015 Top500 list, while the Xeon Phi™ is in 6%. Intel® came out with Xeon Phi™ to compete with NVIDIA GPUs by offering a unified environment that supports OpenMP and MPI, and by providing competitive and easier-to-utilize processing power with less energy consumption. Maximum Xeon Phi™ execution-time performance requires that programs have high data parallelism and good scalability, and use parallel algorithms. And, improved Phi™ power performance and throughput can be achieved by reducing the number of cores employed for application execution. Accordingly, the objectives of this paper are to: (1) Demonstrate that some applications can be executed with fewer cores than are available to users with a negligible impact on execution time: For 59.3% of the 27 application instances studied, doing this results in better performance and for 37% using less than half of the available cores results in performance degradation of not more than 10% in the worst case. (2) Develop a tool that provides the user with the optimal number of cores to employ: We designed an algorithm and developed a plugin for the Periscope Tuning Framework, an automatic performance tuner, that for a given application provide the user with an estimation of this number. (3) Understand if performance metrics can be used to identify applications that can be executed with fewer cores with a negligible impact on execution time: We identified, via statistical analysis, the following three metrics that are indicative of this, at least for the application instances studied: low L1 Compute to Data Access ratio, i.e., the average number of computations that are performed per byte of data loaded/stored in the L1 cache, high use of data bandwidth, and, to a lesser extent, low vectorization intensity.

CCS Concepts

•General and reference → Performance; Measurement; Metrics; •Computing methodologies → Parallel computing methodologies;

Keywords

Performance, parallel programming, Intel® Xeon Phi™, Periscope, autotuning

1. INTRODUCTION

High Performance Computing (HPC) is vital to solving the ever-increasing number of highly complex problems that span a wide variety of areas in science and engineering [9]. Large HPC clusters cost millions of dollars to run and consume incredible amounts of energy. Thus, a current trend in HPC is to combine high-frequency multi-core processors with different types of (many-core) accelerators to improve computational power and energy efficiency [11].

Accordingly, some of today's HPC systems include nodes with processors that have multi- and many-core architectures, such as GPUs and the Intel® Xeon Phi™, which is an accelerator based on the Intel® Many Integrated Core (MIC) architecture. The current Xeon Phi™ has up to 61 cores and is designed to be power efficient, provide high throughput, and perform best for highly parallel, computation-dense applications. Optimal performance on the Xeon Phi™ can be obtained by adhering to Intel's® five cornerstones of Phi™ performance: (1) high parallelism, (2) high vectorization, (3) low use of memory bandwidth, (4) good TLB usage, and (5) good data locality [24]. Unlike GPUs, the Phi™ can be used as a standalone platform, which is known as native mode, or, like GPUs, can be coupled as an accelerator with a high-frequency host processor.

The power efficiency of the Xeon Phi™ can be improved by executing an application on fewer cores than are made available to users. In fact, decreasing the number of Phi™ cores used in a computation is beneficial for several reasons. First, since multiple applications can run concurrently on a Phi™ (as long as there are a sufficient number of available cores and ample memory), using fewer cores than are available allows multiple applications/tasks to execute in parallel and, thus, can increase throughput. However, over-subscription of cores can cause resource contention and pipeline latency to rise and, thus, can increase execution time and possibly energy consumption. Second, high memory bandwidth applications executed with large numbers of

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

XSEDE16, July 17-21, 2016, Miami, FL, USA

© 2016 ACM. ISBN 978-1-4503-4755-6/16/07...\$15.00

DOI: <http://dx.doi.org/10.1145/2949550.2949581>

threads on too many cores can increase memory contention and possibly saturate the memory bandwidth, which also can increase execution times and energy consumption [18]. Decreasing the number of cores utilized by such applications can ameliorate this situation. And, finally, as shown in [27], when cores are left idle, the average power consumption (Joules/second) decreases significantly. However, not using all of the available cores to execute a low memory bandwidth application can increase its execution time, which can result in higher energy consumption (Joules).

With these points in mind, the first objective of our research is to demonstrate that when representative OpenMP-based parallel applications are executed on the Xeon PhiTM in native mode with fewer cores than are made available to users, either execution time decreases or there is a negligible negative impact on execution time. As discussed in Sections 3 and 4, this demonstration is provided by an experimental study that was driven by a matrix-multiply kernel (a common component of many applications) and ten representative Rodinia benchmarks [5] that solve problems from the HPC dwarves [3], such as N-body simulation, dense linear algebra, graph traversal, dynamic programming, and structured and unstructured grids. This study: (1) quantifies, for 27 application instances, how execution time changes with the number of employed PhiTM cores and threads per core (configuration) and (2) demonstrates that for 59.3% of the application instances executing with fewer cores than are available results in better performance and for 37% using less than half results in performance degradation of not more than 10% in the worst case. Note that for each application instance studied, 180 runs are required to quantify how execution time changes with the employed configuration and to identify the configuration (and optimal core count) that results in the shortest average execution time. This “exhaustive-search” process can take days or even weeks to complete for very large problem sizes.

Our Periscope [10] Minimum Core Identification on Phi (MCIP) plugin, which uses our performance-bounded binary-search algorithm, addresses our second objective, i.e., to design and implement a prototype tool that provides users with the optimal number of cores to employ when executing an application on an Intel[®] Xeon PhiTM. Although our performance-bounded binary-search algorithm works as expected, the MCIP plugin provides some erroneous results. The good news is that for the workloads studied: (1) the plugin provides output in 3.65% of the time required by the above-mentioned exhaustive-search process, and (2) for 67% of the application instances studied, its output, i.e., the number of PhiTM cores to employ, is within 10% of the optimal core count. However, the bad news is that for the other 33% of the application instances the recommended number of cores is between 60% (for Breadth-First Search) and 250% (for LU Decomposition) above the optimal core count. As discussed in Section 5, these errors are likely due to the differences in the methods used by the plugin and the experimental study to report application execution times.

Finally, our third objective is to identify, via statistical analysis, performance metrics that are indicative of an application’s propensity to execute on fewer cores with a negligible impact on execution time. As discussed in Section 6, we attempt to do this by iteratively employing unsupervised k-means clustering of hardware event counts that were collected for 27 application instances via Intel[®] VTuneTM

Amplifier. The results suggest that, for the application instances studied, these metrics are: a low L1 Compute to Data Access ratio, i.e., the average number of computations that are performed per byte of data loaded/stored in the L1 data cache, high use of data bandwidth, and, to a lesser degree, low vectorization intensity.

The next section presents an overview of related work. Then, Section 3 describes our experimental platform, focusing, in particular, on the Intel[®] Xeon PhiTM. As described above, Sections 4, 5, and 6 present the execution-time behavior of the 27 studied application instances when executed with different numbers of PhiTM cores and threads per core, describe the Periscope MCIP plugin and discuss its efficacy, and briefly explain our statistical analysis of hardware event counts, which identifies indicators of an application’s propensity to execute on fewer cores with acceptable execution-time performance. Finally, we present our conclusions and outline future work.

2. RELATED WORK

The recent introduction of the Intel[®] Xeon PhiTM, which is based on one of the latest accelerator technologies, has motivated several related publications that focus primarily on its use in HPC clusters, e.g., on application porting, optimization, performance evaluation, and energy consumption. For example, in terms of performance of the Xeon PhiTM, Misra, et al. [21] compare the performance of a standard Intel[®] Xeon CPU to that of a Xeon PhiTM using two applications from the Rodinia benchmark suite, LU and HotSpot. Their results show that the Xeon CPU significantly outperforms the PhiTM (with the best configuration), which took almost 8X more time to execute LU and 2X more to execute HotSpot. And in [8], Gallardo, et al. present a comparison of the performance of the LULESH 1.0 proxy application executed on a PhiTM, NVIDIA Fermi and Kepler GPUs, and an Intel dual Xeon E5-2680 (Sandy Bridge) multi-core processor. This comparison, which is in terms of achieved instructions per cycle, vectorization usage, memory behavior, and energy consumption, shows that although the distribution of execution time among the four main computational phases of LULESH 1.0 is similar across these computing platforms, the application runs 7X faster on the Kepler [8]. Also, several performance-related benchmarks have been ported to the PhiTM including the STREAM benchmark, which is used to measure memory bandwidth [7], and the EPCC micro-benchmarks, which are used to measure the overhead of OpenMP constructs [7, 26].

Two publications that focus on the energy consumption of the Xeon PhiTM, [18] and [27], present different approaches to its measurement. The energy model of Shao and Brooks, which is generated at the instruction level, is used in [27] to measure energy consumption while varying the number of cores, threads per core, and instruction types. In contrast, in [18], Li, et al. present application-level measurements and vary only the thread count, focusing more on the possible causes of energy and computation performance degradation on the PhiTM: oversubscription of cores, its small L1 and L2 caches, memory contention, and its lack of an L3 cache [18].

In contrast, the main focus of our work is the reduction of the number of cores employed by OpenMP-based applications executed on the Intel[®] Xeon PhiTM. Regarding thread-count minimization, there are no publications spe-

cific to the PhiTM, but Cochran, et al. [6] present a general online approach that optimizes energy/performance functions by changing the number of OpenMP threads and using DVFS to modify processor frequency. Other publications regarding thread-count optimization focus on heterogeneous systems [20, 23] or scheduling mechanisms [22].

In addition, since we use OpenMP-based applications to drive our experiments, we mention two other publications that are relevant to our research: (1) Mathur, et al. [19] present a summary of common factors that affect OpenMP performance, and (2) Ryoo, et al. [25] present an analysis of matrix multiplication (one of the OpenMP-based applications used in our study) and the optimal configuration for a GPU; metrics derived from static code are used to estimate the first-order factors of performance, which in turn define a specific search space for the execution time that lies on a Pareto-optimal curve [25].

3. EXPERIMENTAL SETUP

The applications selected to drive our experiments and the test bed used to conduct them are described next. The test bed is a standalone computer system located at the University of Texas at El Paso. It contains: (1) an Intel[®] Xeon E5-1650 host processor comprised of six 3.2 GHz cores with two hardware threads per core and an Intel[®] Xeon PhiTM 5110P accelerator comprised of 60 1.052631 GHz cores with four hardware threads per core. Below, in Section 3.2, we provide more details regarding the accelerator’s architecture and execution environment. The host processor and the accelerator are connected by a PCIe interface. The Intel[®] Many-core Platform Software Stack (MPSS), a suite of tools, is required for execution on the accelerator; our test bed employed MPSS version 2.1.6720-15.

3.1 Applications

The applications used in this study include an OpenMP implementation of the General Matrix-Matrix Multiplication with double-precision (DGEMM) and ten OpenMP benchmarks from the Rodinia Benchmark Suite: Back Propagation (Backprop), Breadth-First Search (BFS), CFD Solver, HotSpot, K-means, LavaMD, LU decomposition (LUD), Needleman-Wunsch, Pathfinder, and Streamcluster [1]. Table 1 presents the 27 workloads used for the 11 selected applications. Each of the 11 applications were instrumented to report the execution time of its parallel section of code.

The Rodinia benchmarks are available in different versions, which use various parallel programming languages or libraries, such as OpenMP, CUDA, and OpenCL. Although they were developed to evaluate heterogeneous multi-core systems, they are suitable for evaluating many-core homogeneous systems such as the Intel[®] Xeon PhiTM when used in native mode. Since the Rodinia benchmarks have been used to evaluate GPU-based platforms, there is a significant amount of data available to compare performance across platforms [21].

3.2 Intel[®] Xeon PhiTM

The PhiTM is comprised of a maximum of 61 dual-issue in-order cores that are connected through a high-performance on-die bidirectional interconnect. Each core supports the fetch-and-decode of instructions from four different hardware threads, which are accessed in a round-robin fashion. Applications that are highly tuned can reach maximum per-

Table 1: Application workloads

Application	Input	Input type
Backprop	10,000,000	Elements
	20,000,000	Elements
BFS	8M	File
	16M	File
Euler3D	Fvcorr	File
	Missile	File
-Euler3D double	Fvcorr	File
	Missile	File
-preEuler3D	Fvcorr	File
	Missile	File
-preEuler3D double	Fvcorr	File
	Missile	File
HotSpot	1,024	Grid size
K-means	kdd_cup	File
	819,200	File
LavaMD	10	Boxes per dimension
	20	Boxes per dimension
LUD	512	Dimensions
	2,048	Dimensions
Matrix-multiply	2,048	Dimensions
	4,096	Dimensions
	6,000	Dimensions
Needleman-Wunsch	4,096	Dimensions
	8,192	Dimensions
PathFinder	10,000	Width
Streamcluster	10-20	Centers Allowed
	30-40	Centers Allowed

formance using only two threads, but usually three or four threads per core are required [16]. Each core has local L1 instruction and data caches, each of which has a capacity of 32KB, and a local 512KB unified L2 cache. The PhiTM also includes a data Translation Lookaside Buffer (DTLB) and a 512-bit wide Vector Processor Unit (VPU) [15, 14]. Additionally, it has eight GDDR5 memory controllers with two channels each, offering a total of 16 channels with a transfer speed of 5.5 GT/s (gigatransfers per second), which together can deliver a theoretical bandwidth of 352 GB/s (gigabytes per second). Because of its large number of cores, it is only recommended for highly-parallel applications with a high computation-to-memory-access ratio. The PhiTM uses a PCI Express* 2.0 bus as the system interface to the host CPU. Power and thermal management are also available, including power capping support [15].

3.2.1 Intel[®] Xeon PhiTM Execution Environment

The Intel[®] Xeon PhiTM supports three different execution modes, which can be used to execute serial or parallel applications. The applications can be executed on the host, the accelerator, or a combination of both. The most common libraries used for parallelization are OpenMP, MPI, and a hybrid of these two.

Because it has an embedded Linux operating system, the PhiTM can be used as a standalone machine in native mode (which was used for all of the experiments described in this paper). However, it is worth doing an analysis of the characteristics of the code to determine if native mode is a suitable option [2]. In general, an application will execute well on the PhiTM if it: (1) has high data parallelism, (2) has good scal-

ability, and (3) employs parallel algorithms [17]. For native mode, it is particularly important that the memory size be sufficient for the application’s requirements. In addition, the serial portion of the code should be small and I/O instructions should be minimized. If the programmer can identify hotspots in the code that do not require a significant amount of data transfers, the offloading of these kernels should be considered.

In offload mode, specific computationally-intensive sections of a program are executed on the PhiTM. This requires moving data between the host processor and the PhiTM. Two different offload modes are available: automatic offload and compiler-assisted offload. The former is performed by special libraries such as the Intel[®] MKL (Math Kernel Library), while the latter requires the developer to add special pragma directives to the code, which indicate the sections to be offloaded and the data to be moved [28]. However, data transfers are expensive and, thus, should be avoided [17].

Because an Intel[®] Xeon PhiTM has many cores and four hardware threads per core, there are many possible ways to map virtual OpenMP threads to the available physical cores and threads. The MIC architecture allows the user to define: the number of cores and the number of threads per core to be employed, thread affinity, and thread placement. Changes to these options can have a noticeable effect on an application’s execution-time performance [13].

4. EXECUTION-TIME BEHAVIOR

As mentioned earlier, the first objective of our research is to demonstrate that representative OpenMP-based parallel applications executed on the PhiTM in native mode achieve optimal or close to optimal execution time when executed on fewer cores than are made available to users. This is demonstrated by the results of our first set of experiments, which employ the experimental platform and plan described in Sections 3 and 4.1, respectively. As described next, these results, which are presented in Section 4.2, also are used in Sections 5 and 6.

4.1 Experimental Plan

This first set of experiments was designed to: (1) identify the optimal PhiTM configuration for each workload, i.e., the one that results in the shortest execution time; (2) establish baselines for quantifying the accuracy of the Periscope MCIP plugin, which employs a new search algorithm that is based on the binary-search algorithm; and (3) determine for each workload, the PhiTM configurations to use when collecting performance data to be analyzed to identify performance metrics that indicate a workload’s propensity to execute on fewer cores with a negligible increase in execution time. These experiments measure the execution times of the studied workloads (described in Table 1) when executed on 180 different configurations of the Intel[®] Xeon PhiTM in our experimental test bed. The set of configurations studied varies the number of cores from 1 to 60, which is the maximum number of cores available for this model of the PhiTM, and the number of threads per core from 2 to 4. One thread per core was not used since a general recommendation is to use more than one in order to reduce the inherent data access latency [13].

To measure and report workload execution times, each application was manually instrumented by adding Intel’s[®] MKL function `dsecnd()` at the beginning and end of the

Table 2: Optimal number of PhiTM cores for the studied 27 workloads executed with four threads per core.

Cores (c)	Frequency	Percentage of Workloads
$1 \leq c \leq 30$	5	18.5%
$31 \leq c \leq 55$	5	18.5%
$56 \leq c \leq 58$	6	22.2%
$59 \leq c$	11	40.7%

code region of interest, i.e., the program’s computational phase [12]. The affinity setting for the experiments was set to **COMPACT**, with the granularity set to fine. These settings were used because if `KMP_PLACE_THREADS` is specified, the **COMPACT** and **BALANCED** affinity settings are equivalent; and the selection of fine granularity assures that the threads are only assigned to one context.

Each workload was executed on each configuration 10 times to obtain the minimum, maximum, and average execution times. From these results, the optimal configurations for each workload were identified, i.e., the ones with the shortest execution time, as well as the configurations that provide execution times within 10% and 15% of the shortest execution time. These percentages were chosen because the measured execution times of each application have high variation on the Xeon PhiTM. In [4], we report, for each workload/configuration pair, the mean percentage variation of the 10 measured execution times as well as the steps taken to address this problem. In particular, Matrix Multiply has an execution-time variance of 10%, while other applications that do not use random variables have similar execution-time variances.

4.2 Experimental Results

Our analysis of the collected execution times shows that for more than 80% of the 27 workloads studied the optimal performance is obtained with four threads per core. Thus, due to space limitations, in this paper we present and analyze only the results for the experiments run with four threads per core. Accordingly, Table 2 presents the optimal number of PhiTM cores to employ for the execution of the 27 workloads using four threads per core. In this table, frequency denotes the number of workloads for which the specified number of cores is optimal. The percentage of workloads that can reap execution-time performance benefits from using fewer cores than are available to users is 59.3%. This result indicates the importance of appropriate selection of the number of cores to employ to execute an application.

Tables 3 and 4 present the number of cores required by the 27 workloads to attain an execution time within 10% and 15%, respectively, of that of the optimal core count. It is interesting to note that for 37% of the workloads it is possible to use less than half of the available cores and still attain an execution time that is within 10% of that of the optimal core count. In terms of the results presented in the next section, keep in mind that the total time required to run all of the experiments discussed in this section is approximately 144 hours, which is equivalent to six days!

To verify the effect of using different affinity settings, we repeated this first set of experiments using the **SCATTER** and **BALANCED** affinity settings. While the execution-time perfor-

Table 3: Number of cores that attained an execution time within 10% of that of the optimal core count for the studied 27 workloads.

Cores (c)	Frequency	Percentage of Workloads
$1 \leq c \leq 30$	10	37.0%
$31 \leq c \leq 55$	14	51.9%
$56 \leq c \leq 58$	3	11.1%
$59 \leq c$	0	0.0%

Table 4: Number of cores that attained an execution time within 15% of that of the optimal core count for the studied 27 workloads.

Cores (c)	Frequency	Percentage of Workloads
$1 \leq c \leq 30$	10	37.0%
$31 \leq c \leq 55$	15	55.6%
$56 \leq c \leq 58$	2	7.4%
$59 \leq c$	0	0.0%

mance of 59.3% of the 27 workloads improved using fewer cores than are available to users, the results of these additional experiments show that this is true for 60.9% (using SCATTER affinity) and 52.2% (using BALANCED affinity) of 23 of the workloads, respectively. When setting the affinity to BALANCED (SCATTER), the average of the absolute values of the differences between the optimal core count for each workload and that identified with COMPACT affinity employed is 2.6 (3.2). The average of the raw differences is -0.2 (2.3).

5. PERISCOPE MCIP PLUGIN

The objective of the Periscope [10] Minimum Core Identification on Phi (MCIP) plugin is to automatically identify the best number of PhiTM cores (or threads) to employ for the execution of a user-specified region of an application (not the entire application). Although the MCIP plugin targets the Intel® Xeon PhiTM, with small modifications it can be used for other many-core architectures.

Each Periscope plugin is associated with a tuning objective, as well as tuning parameters and tuning actions. The tuning objective defines the execution parameter to be measured and optimized. The tuning parameters are the modifiable variables that directly affect the tuning objective, while the tuning actions are the modifications made by the plugin to the tuning parameters. The tuning objective of the MCIP plugin is to minimize the execution time property. The plugin has only one tuning parameter, i.e., the number of OpenMP threads employed, and its tuning action is defined as a modification of the value of the tuning parameter. The number of OpenMP threads was selected as the tuning parameter because this is the only parameter that can be modified during runtime. Thus, the MCIP plugin uses the affinity settings set before runtime.

The MCIP plugin can explore the search space using two different approaches: exhaustive search and performance-bounded binary search. Using the exhaustive-search methodology, the MCIP plugin measures the execution times of all possible tuning parameter values. Thus, the user must specify the number of threads per core and the upper and lower bounds of the tuning parameter, in this case, the number of threads. For example, assuming that 60 cores are available, the user must set: (1) threads per core to 2 and

run the workload with 1 to 120 threads; (2) then set threads per core to 3 and run the workload with 1 to 180 threads; and (3) then set threads per core to 4 and run the workload with 1 to 240 threads.

Our performance-bounded binary-search algorithm is similar to the traditional binary-search algorithm, but in this case it takes into account: (1) the possibility that using fewer cores may significantly increase a program’s execution time, and (2) the fact that we can accept a small decrease in the program’s execution-time performance in order to reduce the number of cores employed. The algorithm is described next:

- Define PERF_BOUND, the percentage by which the program’s execution time is allowed to increase from the best measured execution time. Any configuration with an execution time below this performance boundary is considered acceptable.
- Bound the search space by the minimum (MIN_THREADS) and maximum (MAX_THREADS) number of threads to be evaluated. Define the value at the middle of this range to be MID_THREADS.
- Measure and compare the program’s execution times using MAX_THREADS and MID_THREADS.
- If the execution time attained using MID_THREADS is greater than that attained using MAX_THREADS by a percentage higher than PERF_BOUND, create a new search space for which the new MIN_THREADS value is MID_THREADS; MAX_THREADS remains the same.
- Else, create a new search space that is between MIN_THREADS and MID_THREADS.
- Continue this process until the search space is reduced to one element, which is transformed from number of threads to number of cores and then is returned as the number of cores to use for this particular performance boundary.

5.1 Experimental Plan

Our second set of experiments is designed to: (1) evaluate the accuracy of the MCIP plugin when it uses the performance-bounded binary-search algorithm, which aims to reduce the number of configurations evaluated to identify the optimal configuration with which to execute a workload and (2) automate the process of instrumentation and execution of the experiments via the Periscope Tuning Framework.

Two different applications, Euler3D and LavaMD, are not included in the evaluation of the MCIP plugin because it was not possible to instrument them using Periscope. Thus, the plugin is evaluated using 18 of the 27 workloads, COMPACT affinity with fine granularity, and three different performance-boundaries, i.e., 0%, 10%, and 15%. In terms of the Periscope MCIP plugin, a 0% performance boundary means that only the number of OpenMP threads with the minimum execution time is returned.

Each of these experiments utilizes a configuration with four threads per core. This parameter was chosen because for 80% of the experiments discussed in Section 4 optimal execution-time performance is achieved using four threads per core. The accuracy of the MCIP plugin is evaluated by computing the percentage difference between the number of

cores recommended by the plugin and the optimal number of cores identified by our first set of “exhaustive” experiments.

5.2 Experimental Results

For the workloads studied, the MCIP plugin using the performance-bounded binary-search algorithm provides the recommended number of cores in 3.65% of the time required by the plugin when it uses the exhaustive-search methodology. The number of execution-time measurements per workload is reduced from 180, using the exhaustive-search methodology, to at most 9. For 67% of the 18 workloads studied the number of cores recommended by the MCIP plugin (using the performance-bounded binary-search algorithm) are within 10% of the optimal core count. However, for the other 33%, the error w.r.t. the optimal core count is not acceptable, i.e., it is from 60% (for BFS) to 250% (for LUD).

We believe that these errors may be due to the different ways that execution time is measured in our first set of experiments (described in Section 4), which use Intel® MKL’s `dsecnd()` function, and in the Periscope MCIP plugin experiments, which use OpenMP’s `omp_get_wtime()`. This claim is supported by our testing of the performance-bounded binary-search algorithm outside of the Periscope Tuning Framework. These tests, driven by measurements provided by the first set of experiments, provide significantly better results, i.e., errors ranging from 0% (for BFS) to 55% (for Backprop), with an insignificant increase in execution time of 2.82% in the worst case.

6. PERFORMANCE ANALYSIS

Our third objective is to identify performance metrics that indicate an application’s propensity to execute on fewer PhiTM cores with a negligible impact on execution time. This was accomplished by: (1) computing selected performance metrics for each of the 27 workloads described in Section 3.1 executed on a subset of the PhiTM configurations used in our first set of experiments, and (2) analyzing the behavior of each of these metrics across all of the executions of the selected workload/configuration pairs.

The selected performance metrics are a subset of those presented in [14], which are aimed at optimization and performance tuning of applications executed on the Intel Xeon PhiTM. (We actually used all of the metrics presented in [14] but determined that three of them, L1 TLB misses per L2 TLB miss, Estimated Latency Impact, and L2 Compute to Data Access Ratio were not relevant.) To compute these metrics, we conducted a third set of experiments to collect the required hardware event counts for each of the executions of the selected workload/configuration pairs using Intel® VTuneTM Amplifier XE.

Although the open-source PAPI library also can be used for this type of profiling, we selected VTuneTM because it can access all of the required PhiTM hardware event counts, while PAPI cannot. Also, as a vendor-specific solution, VTuneTM is expected to provide more accurate results. Note that the affinity setting used for these experiments is `COMPACT` with fine granularity and the number of threads per core is set to four. The thread distribution is defined via the `KMP_PLACE_THREADS` environment variable.

6.1 Experimental Plan

The performance metrics used in our analysis are: average cycles per instruction (CPI), L1 Compute to Data Access ra-

tio (average number of computations per byte loaded/stored in L1 cache), L1 Cache Hit Rate, L1 TLB Miss Rate, Vectorization Intensity, Read/Write Bandwidth (bytes/clock), and Bandwidth (GB/sec). Consequently, the hardware event counts collected are: `CPU_CLK_UNHALTED`, `INSTRUCTIONS_EXECUTED`, `VPU_ELEMENTS_ACTIVE`, `DATA_READ_MISS_OR_WRITE_MISS`, `L1_DATA_HIT_INFLIGHT_PF1`, `DATA_READ_OR_WRITE`, `L2_DATA_READ/WRITE_MISS_CACHE_FILL`, `L2_DATA_READ/WRITE_MISS_MEM_FILL`, `DATA_PAGE_WALK`, `L2_VICTIM_REQ_WITH_DATA`, `SNP_HITM_L2`, `LONG_DATA_PAGE_WALK`, `VPU_INSTRUCTIONS_EXECUTED` and `HWP_L2MISS`.

Since the profiling of a workload with VTuneTM can take a significant amount of time, i.e., much longer than the execution of the workload, itself (in this study up to five times longer) and can collect a very large amount of data (for some of the workloads studied, 2GB to 3GB per experiment), we carefully selected, for each of the 27 workloads, the PhiTM configurations to study. This down-selection was done by: (1) plotting the execution times attained via our first set of experiments (for 1 to 60 cores with 2 to 4 threads per core) against the number of cores employed, and then (2) selecting the configurations that best describe the execution-time behavior of each workload as the number of cores increases. This resulted in from 7 to 11 configurations per workload, and the collection of the aforementioned hardware event counts for each workload executed on its selected configurations.

Using these hardware event counts, the above-mentioned performance metrics were computed for each workload/configuration pair under study. Next, the metrics associated with each workload were averaged across all of the configurations (from 7 to 11) under study. Finally, to compare the values of the different performance metrics across the 27 workloads, the average value of each metric (across the 27 workloads) was computed and standardized using the mean and standard deviation.

Given these data, the next step is to identify the performance metrics that indicate an application’s propensity to execute on fewer cores with a negligible impact on execution time. For this we used the k-means clustering algorithm implemented in MATLAB using `kmeans++` initialization. The distance measures used for these experiments were: Manhattan distance (L_1 norm), squared Euclidean distance (L_2 norm), cosine distance, and correlation distance. To obtain the performance metrics that resulted in the best clustering, i.e., discern which workloads can execute on fewer cores with a negligible impact on execution time, we used a greedy algorithm: Initially all features, i.e., performance metrics, are included in the model and the Rand index and average Silhouette coefficient evaluation measures are computed. Afterwards, one feature is removed from the model at a time. If the removal of a feature causes the Rand index to decrease, this feature is reinserted into the model. This process is repeated iteratively until the removal of any feature decreases the Rand index. The remaining set of features are considered to be the most relevant performance metrics. Finally, we performed a more in-depth analysis of the obtained set of performance metrics. Next we present the results of this analysis, focusing on the main trends as well as the probable causes for the observed outliers.

6.2 Experimental Results

The best Rand index value obtained in our experimen-

tation, i.e., 0.8575, was computed using cosine distance; in this case, only two samples were clustered incorrectly. The performance metrics that provided this value were: L1 Compute to Data Access ratio, Read Bandwidth (bytes/clock), and Bandwidth (GB/sec). For this case, the average Silhouette coefficient obtained was 0.3199, which indicates an acceptable clustering.

Even so, it is important to mention that a classification algorithm should be used after clustering and removal of outliers. However, this was not possible in our case because of the small number of samples, i.e., 27 workloads, which was constrained by the time and storage space requirements of hardware event count collection via Intel[®] VTune[™]. For example, each experiment associated with one of our workloads ran for two days and consumed almost 2GB of memory; and, there are from 7 to 11 experiments per workload. Consequently, we did not employ a classification algorithm in this study but, nonetheless, analyzed the available data for the three performance metrics with the best Rand index. The results of this analysis indicate the following:

- For 52% of the workloads (14 of 27) the trend is very clear: a low L1 Compute to Data Access ratio (< 0.4 standardized) and high Read Bandwidth and Bandwidth values (> 0.4 standardized) may indicate that using fewer cores will realize improved execution-time performance. On the other hand, a high L1 Compute to Data Access ratio (> 0.0 standardized) and low Read Bandwidth and Bandwidth values (< 0.0 standardized) may indicate that using most of the available cores will decrease execution time.
- For 11% of the workloads (3 of 27), it appears that a very low L1 Compute to Data Access ratio (< -1.3 standardized) may indicate that using fewer cores will improve execution-time performance, even if the Read Bandwidth and Bandwidth are close to zero (between -0.5 and 0.4). Similarly, for 7% of the workloads (2 of 27), even when the L1 Compute to Data Access ratio is relatively high (1.2 standardized), high Read/Write Bandwidth and Bandwidth values (> 0.9 standardized) indicate that using fewer cores will result in optimal execution-time performance.
- An interesting case is Matrix Multiply using a 4096×4096 matrix, which is one of the samples that was clustered incorrectly. In this case, the behavior is very similar to the previous one, however, here the optimal number of cores to use to execute the workload is “all of the cores”. But, after further analysis, it seems that, unlike the previous case, this workload has one of the highest vectorization intensities, which could explain the benefit of using more cores.
- In 7% of the workloads (2 of 27), even though the L1 Compute to Data Access ratio is lower than the average (-0.34 standardized), the low values for Read Bandwidth and Bandwidth (< -0.95 standardized) indicate that using all of the available cores will improve execution-time performance.
- Similarly, for 11% of the workloads (3 of 27), high values of the L1 Compute to Data Access ratio (> 0.5 standardized) combined with average values of Read Bandwidth and Bandwidth (0.0 standardized) indicate that using more cores will result in better performance.
- Finally, LU Decomposition (LUD) appears to be an outlier in the sense that this is the only application that we studied for which a smaller problem size (512×512) performs best with fewer (17) cores than does a larger one ($2,048 \times 2,048$), which performs best using all available cores. Although this, in and of itself, is not indicative of a performance issue, after further analysis of the performance data, we found this version of LUD to be far from optimal, i.e., of all the applications studied, both related workloads have the lowest values of Vectorization Intensity (-1.24 and -1.77 standardized) and the highest values of Cycles per Instruction (1.8 and 4.3 standardized).

7. CONCLUSIONS AND FUTURE WORK

In this paper we demonstrated that executing representative OpenMP-based application workloads on the Intel[®] Xeon Phi[™] with fewer cores than are available to users either improves execution-time performance or has a negligible impact on execution time. As mentioned, this also can result in improved power efficiency or throughput. We also presented the Periscope MCIP plugin, which automatically provides, in a minimal amount of time, an estimation of the optimal number of cores to employ for the execution of a given application on the Phi[™]. Finally, we identified, via statistical analysis, that a low L1 Compute to Data Access ratio, high use of data bandwidth, and, to a lesser extent, low vectorization intensity indicate that using fewer cores may improve an application’s performance.

Future work could further enhance the Intel[®] Xeon Phi[™] Periscope MCIP plugin, e.g., affinity and thread distribution could be included as tuning parameters. In addition, a machine-learning search algorithm could be implemented using previous predictions and input vectors to predict the optimal number of Phi[™] cores to use for an application’s execution. Clearly, the variations in Phi[™] execution times should be addressed and a similar study should be conducted for the next release of the Phi[™], i.e., Knights Landing.

8. ACKNOWLEDGMENTS

This research was performed in collaboration with Technische Universität München, in particular with the help of Isafas Comprés Ureña and Umbreen Sabir Mian. This work was funded by the National Science Foundation Stampede grant through the University of Texas at Austin grant no. UTA 13-000072, the Army Research Laboratory’s Army High Performance Computing Research Center grant through the Stanford University sub award no. 60300261-107307-B, and the Consejo Nacional de Ciencia y Tecnología de México.

9. REFERENCES

- [1] ADEPT. Rodinia applications benchmark suite (HPC). Retrieved from: <http://www.adept-project.eu/case-studies/rodinia-applications-benchmark-suite-hpc.html>, 2014.
- [2] A. Argüeta, R. Camacho Barranco, E. Gallardo, P. J. Teller, L. Fialho, and J. Browne. *Quick Start Guide for Using Intel’s Xeon Phis on Stampede*. 2014. Retrieved from: <https://www.tacc.utexas.edu/documents/1084364/1157236/QSG-Main-Final-PJT.pdf/bede045-4a1b-45ce-8a99-2269ebc1836d>.

- [3] K. Asanovic, J. Wawrzynek, D. Wessel, K. Yelick, R. Bodik, J. Demmel, T. Keaveny, K. Keutzer, J. Kubiawicz, and N. Morgan. A view of the parallel computing landscape. *Communications of the ACM*, 52(10):56, 2009.
- [4] R. Camacho Barranco and P. J. Teller. *Analysis of the Execution Time Variation of OpenMP-based Applications on the Intel Xeon Phi*. Technical Report UTEP-CS-16-31, Dept. of Computer Science, University of Texas at El Paso, El Paso, TX, June 2016.
- [5] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, S.-H. Lee, and K. Skadron. Rodinia: A benchmark suite for heterogeneous computing. In *Proc. of the Int. Symp. on Workload Characterization*, pages 44–54. IEEE, 2009.
- [6] R. Cochran, C. Hankendi, A. Coskun, and S. Reda. Identifying the optimal energy-efficient operating points of parallel workloads. In *Proc. of the Int. Conf. on Computer-Aided Design*, pages 608–615. IEEE/ACM, Nov 2011.
- [7] T. Cramer, D. Schmidl, M. Klemm, and D. an Mey. OpenMP programming on Intel Xeon Phi coprocessors: An early performance comparison. In *Proc. of the Many Core Appl. Res. Community Symp.*, pages 38–44, 2012.
- [8] E. Gallardo. *A Case Study of Accelerator Performance*. M.S. thesis, University of Texas at El Paso, El Paso, TX, 2015. Retrieved from: <http://digitalcommons.utep.edu/dissertations/AAI10000797>.
- [9] Georgia Institute of Technology. High Performance Computing, 2016. Retrieved from: <http://www.cse.gatech.edu/content/high-performance-computing>.
- [10] M. Gerndt and M. Ott. Automatic performance analysis with Periscope. *Concurrency and Computation: Practice and Experience*, 22(6):736–748, April 2009.
- [11] 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 Xeon Phi coprocessor. In *Proc. of the 27th Int. Symp. on Parallel and Distributed Processing*, pages 126–137. IEEE, May 2013.
- [12] Intel. Measuring performance with Intel MKL support functions. Retrieved from: <https://software.intel.com/en-us/node/529745>.
- [13] Intel. OpenMP thread affinity control, 2012. Retrieved from: <https://software.intel.com/en-us/articles/openmp-thread-affinity-control>.
- [14] Intel. Optimization and performance tuning for Intel Xeon Phi coprocessors, Part 2: Understanding and using hardware events, 2012. Retrieved from: <https://software.intel.com/en-us/articles/optimization-and-performance-tuning-for-intel-xeon-phi-coprocessors-part-2-understanding>.
- [15] Intel. Intel Xeon Phi coprocessor: Software developers guide, 2013. Retrieved from: <http://www.intel.com/content/www/us/en/processors/xeon/xeon-phi-coprocessor-system-software-developers-guide.html>.
- [16] Intel. Measuring performance in HPC, 2013. Retrieved from: <http://software.intel.com/en-us/articles/measuring-performance-in-hpc>.
- [17] Intel. Building a native application for Intel Xeon Phi coprocessors, 2014. Retrieved from: <https://software.intel.com/en-us/articles/building-a-native-application-for-intel-xeon-phi-coprocessors>.
- [18] B. Li, H.-C. Chang, S. Song, C.-Y. Su, T. Meyer, J. Mooring, and K. W. Cameron. The power-performance tradeoffs of the Intel Xeon Phi on HPC applications. In *Proc. of the Int. Parallel Distributed Processing Symp. Workshops*, pages 1448–1456. IEEE, May 2014.
- [19] J. Li, J. Shu, Y. Chen, D. Wang, and W. Zheng. Analysis of factors affecting execution performance of OpenMP programs. *Tsinghua Science and Technology*, 10(3):304–308, June 2005.
- [20] Y. Luo, V. Packirisamy, W.-C. Hsu, and A. Zhai. Energy efficient speculative threads: dynamic thread allocation in Same-Isa heterogeneous multicore systems. In *Proc. of the 19th Int. Conf. on Parallel Architectures and Compilation Techniques*, pages 453–464. ACM, 2010.
- [21] G. Misra, N. Kurkure, A. Das, M. Valmiki, S. Das, and A. Gupta. Evaluation of Rodinia codes on Intel Xeon Phi. In *Proc. of the 4th Int. Conf. on Intelligent Systems, Modelling and Simulation*, pages 415–419. IEEE, Jan 2013.
- [22] G. Nelissen, V. Berten, J. Goossens, and D. Milojevic. Techniques optimizing the number of processors to schedule multi-threaded tasks. In *Proc. of the 24th Euromicro Conf. on Real-Time Systems*, pages 321–330, July 2012.
- [23] V. Petrucci, O. Loques, D. Mosse, R. Melhem, N. A. Gazala, and S. Gobriel. Thread assignment optimization with real-time performance and memory bandwidth guarantees for energy-efficient heterogeneous multi-core systems. In *Proc. of the 18th Real Time and Embedded Technology and Applications Symp.*, pages 263–272. IEEE, April 2012.
- [24] R. Reed. Performance tuning for Intel Xeon Phi coprocessors, 2013. Retrieved from: <https://software.intel.com/sites/default/files/Slides-Performance-analysis-and-events-with-Intel-Vtune-Amplifier-XE.pdf>.
- [25] S. Ryoo, C. I. Rodrigues, S. S. Stone, S. S. Bagsorkhi, S.-Z. Ueng, J. A. Stratton, and W.-m. W. Hwu. Program optimization space pruning for a multithreaded GPU. In *Proc. of the 6th Annu. Int. Symp. on Code Generation and Optimization*, pages 195–204. ACM/IEEE, 2008.
- [26] D. Schmidl, T. Cramer, S. Wienke, C. Terboven, and M. S. Müller. Assessing the performance of OpenMP programs on the Intel Xeon Phi. In *Proc. of the Euro-Par 2013 Parallel Processing*, pages 547–558, Springer Berlin Heidelberg, 2013.
- [27] Y. S. Shao and D. Brooks. Energy characterization and instruction-level energy model of Intel’s Xeon Phi processor. In *Proc. of the Int. Symp. on Low Power Electronics and Design*, pages 389–394. ACM, 2013.
- [28] Texas Advanced Computing Center. TACC Stampede user guide, 2014. Retrieved from: <https://www.tacc.utexas.edu/user-services/user-guides/stampede-user-guide>.