Cross-Accelerator Performance Profiling

Esthela Gallardo
University of Texas at El Paso
500 W. University Avenue
El Paso, TX 79968-0518
egallardo5@miners.utep.edu

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

Arturo Argueta*
University of Notre Dame
Notre Dame, IN 46556
aargueta@nd.edu

Jaime Jaloma
University of Texas at El Paso
500 W. University Avenue
El Paso, TX 79968-0518
jajaloma@miners.utep.edu

ABSTRACT

The computing requirements of scientific applications have influenced processor design, and have motivated the introduction and use of many-core processors, i.e., accelerators, for high performance computing (HPC). Consequently, it is now common for the compute nodes of HPC clusters to be comprised of multiple computing devices, including accelerators. Although execution time can be used to compare the performance of different computing devices, there exists no standard way to analyze application performance across devices with very different architectural designs and, thus, understand why one outperforms another. Without this knowledge, a developer is handicapped when attempting to effectively tune application performance, as is a hardware designer when trying to understand how best to improve the design of computing devices. In this paper, we use the LULESH 1.0 proxy application to compare and analyze the performance of three different accelerators: the Intel® Xeon Phi™ and the NVIDIA Fermi and Kepler GPUs. Our study shows that LULESH 1.0 exhibits similar execution-time behavior across the three accelerators, but runs up to 7X faster on the Kepler. Despite the significant architectural differences between the Xeon Phi™ and the GPUs, and the differences in the metrics used to characterize their performance, we were able to quantify why the Kepler outperforms both the Fermi and the Xeon Phi™. To do this, we compared their achieved instructions per cycle and vectorization usage, as well as their memory behavior and power and energy consumption.

*Contributed to this research while at the University of Texas at El Paso.

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

DOI: http://dx.doi.org/10.1145/2949550.2949567

CCS Concepts

•**Computer systems organization** → **Heterogeneous (hybrid) systems;** *Single instruction, multiple data; Multicore architectures;* •**General and reference** → **Metrics; Performance;** •**Hardware** → **Emerging tools and methodologies;**

Keywords

Accelerators; Profiling; HPC

1. INTRODUCTION

The computing requirements of scientific applications have influenced processor design, and have motivated the introduction and use of many-core processors for high performance computing (HPC). Consequently, today's HPC clusters include multiple compute nodes that are comprised of general-purpose multi- and many-core processors. In comparison to conventional multi-core processors, many-core processors, i.e., accelerators, provide applications with a much larger number of (simpler) cores that can be used to exploit the parallelism inherent in computation. The architectures of multi-core processors and accelerators are diverse, and those of different accelerators can be very dissimilar as well. For example, while graphics processing units (GPUs) are designed to work with a multi-core processor to perform single operations on large blocks of data (i.e., exploit SIMD (single-instruction, multiple-data) parallelism), the Intel® Xeon Phi™ is designed to work with or independently of a multi-core processor to exploit both SIMD and MIMD (i.e., multiple-instruction, multiple-data) parallelism.

Although GPUs can significantly increase application performance and efforts have been made to improve their programmability, effective GPU programming does not come without a cost. For example, familiarity with the GPU architecture and the CUDA programming model and language is required to make good use of the parallel processing power available in a GPU. In addition, there is a significant overhead associated with moving data between a host processor and a GPU. Nonetheless, because of the execution-time performance benefits that GPUs can provide, it is very common to find them in HPC clusters. Given these facts, Intel® introduced the Many Integrated Core (MIC) architecture with the announcement of the Xeon Phi™. Although the MIC

architecture is similar to that of a stand-alone processor, it has a large number of simpler cores and, unlike GPUs, it: (1) supports programming models that are commonly used in parallel computing, (2) can be programmed using C or FORTRAN, and (3) is able to launch programs independently (while a GPU requires a host processor).

With the emergence of these new many-core processors, HPC systems are becoming increasingly complex. Often their nodes are comprised of multi-core processors along with many-core processors of different types, the performance of which vary for the application being executed. For an application developer, this can make it difficult to determine, without extensive experimentation, whether a program should be launched solely on multi-core processors or should employ accelerators; and, if the latter, to determine which accelerators should be employed and how they should be utilized. For a hardware designer this also can make it difficult to understand how best to enhance application performance. This is because, although several tools exist to study the performance of a computation on different computing devices, most are designed to work with only one class of devices, e.g., GPUs or multi-core processors. And, due to the large variation in the design of device architectures, many performance metrics are unique to a device or class of devices, e.g., GPUs or Intel® devices, making it difficult to understand why one device outperforms another.

To address this and other cross-architecture and cross-programming model concerns, Lawrence Livermore National Laboratory (LLNL) developed the Livermore Unstructured Lagrangian Explicit Shock Hydrodynamics (LULESH) proxy application. LULESH has been ported to several different programming models in order to identify optimization techniques that are portable across the models [5]. In addition, LULESH contains algorithms that are commonly used in hydrodynamics as well as in a variety of computer simulations of science and engineering phenomena. As a result, lessons learned from LULESH regarding its performance can be applied to larger applications that make use of these algorithms. Accordingly, LULESH is an ideal candidate to study the performance of different architectures.

Nonetheless, it is still the case that, given the available performance tools, it is difficult to analyze application performance across computing devices with very different architectures and understand why one outperforms another. In an effort to explore these difficulties and illustrate how available tools can be used, this paper, which is a short summary of the work presented in [3], uses LULESH 1.0 to study the performance of three accelerators: the Intel® Xeon Phi™ and the NVIDIA Fermi and Kepler GPUs. These computing devices are compared in terms of execution time, power consumption, memory behavior, vectorization usage, and instructions per cycle (IPC). In addition, the performance of LULESH 1.0 executed on the Xeon Phi™ and a dual Intel® Sandy Bridge multi-core processor is compared in terms of execution time, parallel overhead, and scalability.

As reported in the paper, LULESH 1.0 exhibits similar runtime behavior across the three accelerators, but executes up to 7X faster on the Kepler. Despite the significant architectural differences between the Xeon Phi™ and the GPUs, we demonstrate how we quantify the performance of LULESH executed on the three accelerators and show why the Kepler outperforms both the Fermi and Xeon Phi™. In addition, the paper demonstrates that comparing the performance of a class of devices (e.g., NVIDIA GPUs) and understanding why one outperforms another can be relatively easy, while understanding the performance differences between devices with diverse architectures remains a challenge.

The rest of the paper is organized as follows: Section 2 describes the application and systems employed in this study and how the performance metrics are computed for each computing device. Next, the collected performance data are presented and discussed in Section 3. Finally, before summarizing our findings and conclusions in Section 5, Section 4 provides an overview of related research.

2. METHODOLOGY

LULESH 1.0, the proxy application employed in this comparative performance study, is discussed in Section 2.1, while the studied computing devices are described in Sections 2.2. Sections 2.3, 2.4, and 2.5 present the methods that were used to collect device-specific performance data and compute and analyze the selected performance metrics. As noted below, these methods are dissimilar due to the differences in the device architectures. LULESH 1.0 was executed 10 times to collect the performance data needed to compute each performance metric; the minimum, maximum, and average values, along with the standard deviation, were recorded. Since the resultant standard deviations are acceptable, average values are used in our comparative analysis.

2.1 LULESH 1.0

There are multiple versions of LULESH, including ones appropriate for the studied architectures. Thus, an OpenMP code with data layout optimizations tuned for the Sandy Bridge (DL code) was executed on the Intel® devices, while two different CUDA codes tuned for the Fermi and Kepler were executed on the NVIDIA GPUs. Table 1 lists the device/code pairs used in this study and the notations with which we refer to them in the remainder of the paper.

Table 1: Device/code pairs.

Device	Code	Notation
dual Sandy Bridge	DL	SB/DL
Xeon Phi™	DL	Phi/DL
Fermi	CUDA Fermi	FGPU/F
Kepler	CUDA Kepler	FGPU/K

As shown in Figure 1, which illustrates the flow of execution of LULESH, each code version is comprised of four main phases of execution. The grayed-out portions of the figure, i.e., the initialization and termination phases, are not studied because they do not map well to real applications. The problem domain of LULESH is described by a mesh of elements, which is used to scale the problem size. Mesh sizes representative of the workload typically executed on a single node, i.e., 50^3, 70^3, and 90^3, are used in this study.

2.2 Experimental Platforms

The Texas Advanced Computing Center (TACC) Stampede cluster was used to collect performance data associated with SB/DL, Phi/DL, and KGPU/K, while LLNL's now-retired Edge cluster was used to collect the data associated with FGPU/F. In addition, stand-alone servers at the University of Tennessee-Knoxville (UTK) were employed to

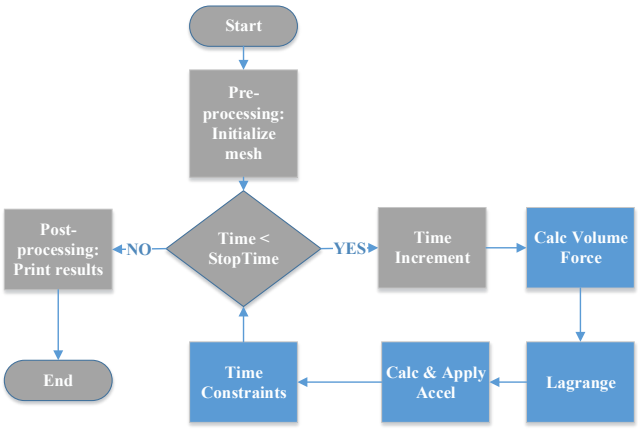

Figure 1: LULESH 1.0 flowchart.

collect the power and energy consumption data. The architectures studied are described in Table 2; note that the architectures of the accelerators in the UTK systems are identical to those in the Stampede and Edge clusters.

Table 2: Accelerator/host processor architectures.

Device	Architecture	Speed	Cores	RAM
dual Xeon E5-2680	Sandy Bridge	2.70 GHz	8	750 GB
Xeon Phi™ SE10P	MIC	1.10 GHz	61	8 GB
Tesla M2050	Fermi	1.15 GHz	(SM) 14	3 GB
Tesla K20	Kepler	705.00 MHz	(SMX) 13	5 GB

2.3 Execution Time and Parallel Overhead

To measure the solve time of LULESH 1.0, i.e., the time it takes to execute its four main phases of execution, and the execution time of each phase, each binary was launched in the best runtime environment of each studied device. On the Xeon Phi™, which supports multiple modes of execution, the application was executed in native mode. Experiments were conducted for each of the three problem sizes.

All versions of LULESH include the timing constructs required to measure execution time. For the Intel devices, the C function `gettimeofday()`, which collects the current time expressed in seconds and microseconds, is employed. In contrast, the GPUs require the use of CUDA events to obtain timestamps from the devices: calls to `cudaEventRecord()` collect timestamps and subsequent calls to `cudaEventElapsedTime()` report the elapsed time between the recorded timestamps of these two CUDA events.

In addition, for SB/DL and Phi/DL, the overhead introduced by parallel constructs is measured using the methodology employed by the EPCC OpenMP microbenchmarks [8]:

$$O_p = T_p - T_s/p, \tag{1}$$

where O_p is the parallel overhead, p is the number of processes that executed the program, and T_p and T_s are the execution times of the OpenMP program and the parallel program compiled without the `-openmp` flag, which causes the OpenMP pragmas to be ignored by the compiler and, thus, serializes program execution. To measure the growth of the overhead as p increases, executions with 2, 4, 8, and 16 Sandy Bridge threads and with 60, 120, and 240 Xeon Phi™ threads are employed. On the Xeon Phi™, `balanced` affinity is used to ensure that all cores are utilized and the number of threads per core is the same.

2.4 Power and Energy Consumption

Different tools are required to measure the power and energy consumption of the three accelerators. The power draw of the GPUs was measured using the NVIDIA System Management Interface program (NVIDIA-SMI) [12], while that of the Phi™ was measured using the Intel® System Management Controller (SMC) [4]. Using these utilities, we wrote scripts, which run concurrently with LULESH 1.0, to read power consumption data every .1 second. Given the solve time (t) and the average power consumption (P), the energy consumption (E) of LULESH 1.0 was calculated by:

$$E = P * t. \tag{2}$$

2.5 Vectorization, Memory Behavior, and IPC

The NVIDIA Visual Profiler [13] is designed specifically to provide data regarding an application's use of NVIDIA GPU resources. Thus, it was used to obtain execution profiles of FGPU/F and KGPU/K, from which we extracted the vectorization usage, memory behavior, and IPC. The vectorization usage was measured using the occupancy of the GPU, which ranges from zero to one and can be translated to a percentage of utilization of the available parallel resources. In contrast, PAPI [10], which requires instrumentation of the code, was used for the Xeon Phi™to collect event counts related to memory performance and IPC. However, due to the small number of available hardware event counters, only events for a single thread could be counted and vectorization usage could not be measured. Consequently, compiler vectorization reports were used to determine the number of loops that were effectively vectorized for the Phi™.

Direct comparison of of the accelerators' memory behavior and IPC is not straightforward. The memory hierarchy of the GPUs is much more complex than that of the Xeon Phi™. While the Phi™ has two levels of cache, an L1 data TLB, and an L2 unified TLB, the GPUs have a texture memory, constant memory, an L1 data cache, and a unified L2 cache. Furthermore, the GPUs include an abstraction of local (to each thread) and global memory, which reside in the same physical location, and the Kepler has a read-only data cache. As a result of these differences and because the number of L1 and L2 cache accesses could not be counted on the Phi™, only the memory behavior of the GPUs could be compared and only in terms of L1 and L2 cache misses.

There are other reasons, besides architectural differences, why direct comparison of the accelerators' IPC is not possible: The IPC of only a single thread (IPC_{Thread}) of the Phi™ could be counted, while the IPC of the GPUs was measured per streaming multiprocessor ($IPC_{SM/SMX}$). Since we know that the main computational unit of a Xeon Phi™ is a core, while that of a Fermi or Kepler is an SM or SMX, the IPC per Phi™ core (IPC_{Core}) is compared to the IPC per SM of the Fermi (IPC_{SM}) and the IPC per SMX of the Kepler (IPC_{SMX}). IPC_{Core} is calculated by multiplying IPC_{Thread} by the size of the Xeon Phi™ pipeline, and since each core consists of a dual-issue pipeline:

$$IPC_{Core} = IPC_{Thread} * 2. \tag{3}$$

Although one might argue that a better approach would be to multiply IPC_{Thread} by the number of threads used per core, it requires unrealistically assuming that each employed

Phi™ thread achieves the same IPC. Furthermore, directly comparing a Phi™ thread to a GPU SM/SMX is inappropriate since it does not distribute its work across many simpler parallel processing elements as does the GPU SM/SMX. Consequently, given the size of the Xeon Phi™ pipeline, it is more reasonable to assume that at least half of the threads executed on a core are able to attain IPC_{Thread}.

3. RESULTS

This section compares the execution-time and power and energy performance of the three accelerator/code pairs. To explain the differences in execution times and speedups, collected performance data are used to compare IPC, memory behavior, and vectorization usage. In addition, the performance of LULESH 1.0 executed on a dual Intel® Sandy Bridge multi-core processor and a Xeon Phi™ are compared in terms of execution time, parallel overhead, and scalability.

3.1 Execution Time

As shown in Figure 2, of the three accelerator/code pairs, KGPU/K provides the best execution-time performance for the three problem sizes. In comparison, FGPU/F and Phi/DL are about seven times slower.

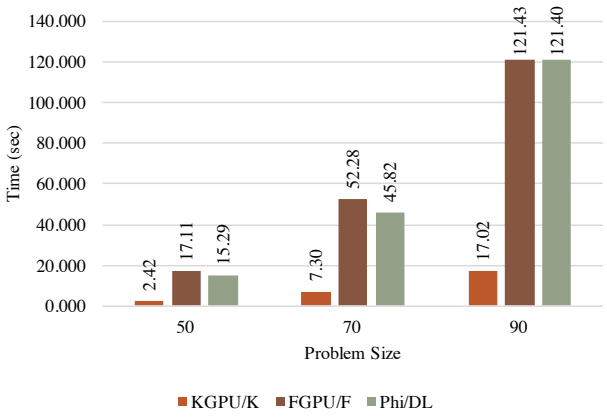

Figure 2: LULESH 1.0 execution times.

Although the accelerators have very different architectures, the distribution of LULESH's execution time across its four phases of execution is similar for all three problem sizes; Figure 3 depicts this behavior for the 90^3 problem size. When the percentage of execution time consumed by each phase is rounded to the nearest whole percentage, for KGPU/K and FGPU/F it changes by at most 2% from one problem size to the next, while for Phi/DL it changes by at most 5%.

For each problem size Calc Volume Force is clearly the most time-consuming phase, with the next being Lagrange. In contrast, Calc & Apply Accel and Time Constraints consume much less of the total execution time, from 5.07% to 14.61% depending on the accelerator/code pair and problem size. Of interest is the fact that although KGPU/K spends approximately the same amount of time in Time Constraints as does FGPU/F and Phi/DL, it spends about twice as much time in Calc & Apply Accel - this may represent an opportunity for further code optimization.

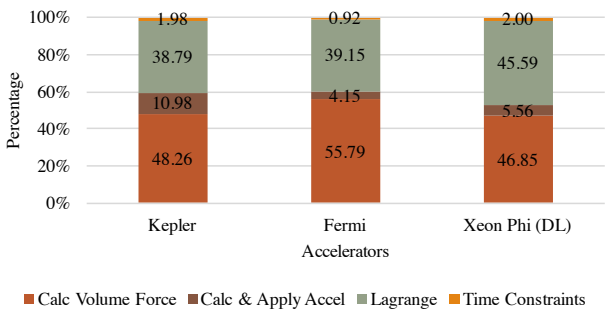

Figure 3: Distribution of execution time across execution phases of LULESH for 90^3 problem size.

3.2 Power and Energy Consumption

As shown in Figure 4, for the three accelerator/code pairs, the average power draw increases with the problem size.

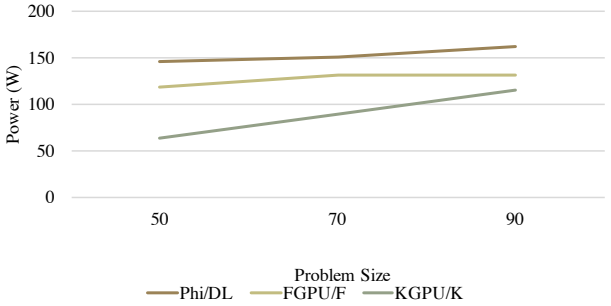

Figure 4: Average power draw.

Although Phi/DL is comparable to FGPU/F in terms of execution time, its average power draw is 21.8%, 14.5%, and 22.7% higher for the 50^3, 70^3, and 90^3 problem sizes, respectively. In fact, Phi/DL has the highest power draw of the three accelerator/code pairs. Although KGPU/K performs best in terms of execution time and power consumption, the rate at which its power consumption grows with the problem size is greater than that of FGPU/F or Phi/DL. The percentage change for KGPU/K maps to the percentage change in the problem size. Specifically, the 70^3 problem size is 40% larger than the 50^3, and the 90^3 is 28.5% larger than the 70^3. Similarly, KGPU/K uses 39.06% more power to execute the 70^3 problem size as compared to that used to execute the 50^3; and 29.21% more power for the 90^3 than for the 70^3. In contrast, the power consumption of the other accelerator/code pairs grows less than 12% from one problem size to the next.

For each of the accelerator/code pairs, average energy consumption tracks execution-time performance. As shown in Figure 5, the Kepler, which executed its version of LULESH 1.0 in the least amount of time, consumes the least amount of energy for all problem sizes. The energy consumption of the Xeon Phi™ and the Fermi GPU, executing their versions, are similar. For the 50^3 and 70^3 problem sizes, the difference between them is less than 10% but for the 90^3 problem size there is a 20% difference, with the energy consumption of the Phi™ being higher.

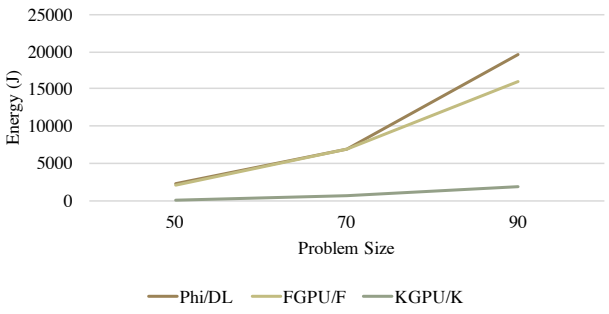

Figure 5: Average energy consumption.

3.3 Memory Behavior

Regardless of the computing device utilized, if an application exhibits poor memory behavior, its execution-time performance will be negatively affected. Accordingly, for each accelerator/code pair we attempted to quantify the number of misses at each level of the accelerator's memory hierarchy.

The Fermi's L1 cache stores data that is resident in either local or global memory, while the Kepler's stores only data that is resident in local memory. Thus, during the execution of LULESH 1.0, the L1-cache misses generated on the Fermi are due to accesses to both local and global memory, while those generated on the Kepler are due to accesses to local memory (note that Kepler global memory transactions are not cached at this level). Also, only on the Kepler does LULESH 1.0 use the available texture memory. Thus, the L2-cache misses generated by KGPU/K are the result of both L1-cache misses and accesses to texture memory, while for FGPU/F they are the result of only L1-cache misses.

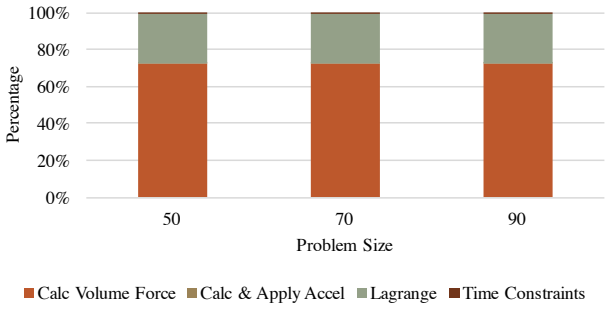

Figure 6: Phi/DL L1-cache misses.

For all three problem sizes, the distribution of L1-cache misses generated by the execution phases of LULESH is similar on the Xeon Phi™ and Fermi. As shown in Figure 6, almost all of the L1 data cache misses generated by LULESH 1.0 on the Xeon Phi™ are attributable to Lagrange and Calc Volume Force. This is true on the Fermi as well. Essentially, for the three problem sizes: (1) Calc Volume Force generates over 70% of the Phi™ L1 data cache misses and 60% (due to local and global memory transactions) of the Fermi L1-cache misses; (2) Lagrange generates around 28% on the Phi™ and 40% (due to local memory transactions) on the Fermi, while on the Phi™Time Constraints and Calc & Apply Accel together generate less than 1% of the L1 data cache misses; (3) as shown in Figure 7, 25% of the Fermi L1-cache misses generated by Lagrange are due to global memory transactions; and (4) in contrast, the L1-cache misses generated by LULESH 1.0 on the Kepler are only generated by the Lagrange phase.

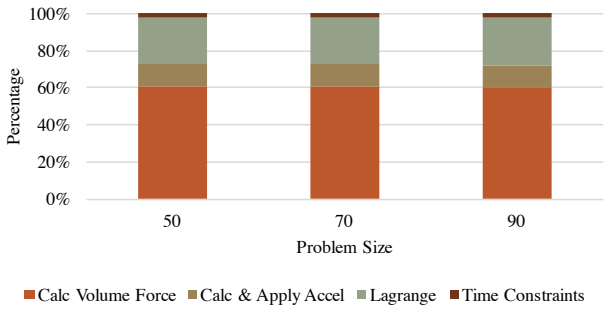

Figure 7: FGPU/F L1-cache misses due to global memory transactions.

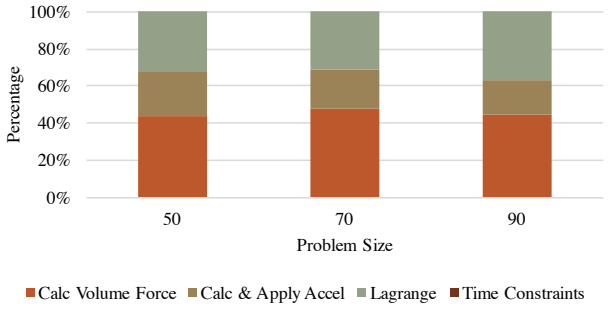

Figure 8: KGPU/K L2-cache misses.

Calc Volume Force and Lagrange also generate the vast majority of L2-cache misses on the Xeon Phi™, i.e., 66%-70% and 26%-27%, respectively. As was the case for the L1 data cache, the number of L2-cache misses generated by Time Constraints and Calc & Apply Accel is minimal, ranging from 3% to 6%. Similar to the Xeon Phi™, over 90% of the Fermi L2-cache misses (due to L1-cache misses) are attributable to the two most time-consuming phases of LULESH. However, only 53% of the misses are generated by Calc Volume Force, while 43% are generated by Lagrange. Finally, 4% of the L2-cache misses are generated by Calc & Apply Accel and Time Constraints. The L2-cache behavior of the Kepler, which is illustrated in Figure 8, is similar to that of the Fermi. However, the distribution of the Kepler L2-cache misses across the four phases of execution of LULESH mirrors the distribution of the execution time.

3.4 IPC and Vectorization Usage

Vectorization usage gauges the ability of a computing device to simultaneously compute one operation over multiple pairs of operands. The Phi/DL compiler vectorization reports indicate that 30% of LULESH's loops were successfully vectorized, but 26.667% of them were not completely vectorized. FGPU/F achieves an occupancy of .378, .369, and .376 for the 50^3, 70^3, and 90^3 problem sizes, respectively; and these occupancies are .428, .433, and .435 for

KGPU/K. Considering the occupancy to be a percentage and comparing it to the upper bound of vectorization for the version of the code executed on the Xeon Phi™: (1) KGPU/K does best in terms of vectorization usage, followed by FGPU/F, and then Phi/DL. (2) And, comparing the occupancy achieved on the Fermi and the percentage of the loops that were vectorized for the Xeon Phi™, the difference between their vectorization usage is less than 15%. This behavior maps to the execution-time performance of these accelerator/code pairs, i.e., for the three problem sizes: (1) the execution times of KGPU/K are less than those of FGPU/F and Phi/DL, and (2) although the execution times of Phi/DL are less than those of FGPU/F, the differences between their execution times do not exceed 15% percent.

With respect to the IPC of the three accelerator/code pairs, which is presented in Table 3: (1) Coinciding with the fact that across the three problem sizes KGPU/K has the shortest execution times, it also has the highest IPC. (2) Unlike FGPU/F and Phi/DL, the IPC of KGPU/K increases, albeit slightly, with the problem size. (3) Although the execution times of FGPU/F and Phi/DL are similar in magnitude across the three problem sizes, the IPC achieved by Phi/DL is consistently higher by about 20%.

Table 3: Instructions per Cycle (IPC).

Architecture/Code Pair	50^3	70^3	90^3
KGPU/K	1.161	1.175	1.178
FGPU/F	0.306	0.309	0.307
Phi/DL	0.360	0.361	0.361

Table 4: Xeon Phi™ IPC per phase of LULESH.

Code Section	50^3	70^3	90^3
Calc Volume Force	0.348	0.348	0.342
Calc & Apply Accel	0.510	0.520	0.522
Lagrange	0.398	0.372	0.358
Time Constraints	0.560	0.614	0.588

Table 5: Fermi IPC per phase of LULESH.

Code Section	50^3	70^3	90^3
Calc Volume Force	0.269	0.272	0.270
Calc & Apply Accel	0.333	0.331	0.330
Lagrange	0.341	0.339	0.339
Time Constraints	1.088	1.243	1.402

Table 6: Kepler IPC per phase of LULESH.

Code Section	50^3	70^3	90^3
Calc Volume Force	0.916	0.924	0.933
Calc & Apply Accel	0.332	0.333	0.334
Lagrange	1.645	1.690	1.681
Time Constraints	1.898	1.933	1.936

Tables 4 - 6 present the IPC of the three accelerator/code pairs for each execution phase of LULESH. As is the case with the overall IPC: (1) the Kepler delivers the highest IPC for three of the four phases, the exception being Calc & Apply Accel, for which the Phi™'s IPC is greater; and (2) for two of the four phases, the Kepler's IPC increases, albeit slightly, with the problem size. Since the Fermi's IPC remains fairly constant for three of the four phases, it cannot be said that LULESH 1.0 will achieve good performance and scalability on GPUs. Measurement of the IPC of each execution phase allows us to understand why the execution time of Phi/DL is smaller than that of FGPU/F. With the exception of Time Constraints for the 50^3 problem size, Phi/DL attained a higher IPC. Clearly, the IPC and vectorization usage of LULESH indicate that its execution-time performance is linked to an accelerator's ability to exploit the inherent parallelism in LULESH 1.0.

3.5 Parallel Overhead

With the exception of the 50^3 problem size, Phi/DL outperforms SB/DL by 5.17% and 4.30% for the 70^3 and 90^3 problem sizes, respectively. Referring to Figure 9 and noting that computation time is the total execution time minus the parallel overhead, the reason for this is twofold: (1) the computation times of Phi/DL are 19%, 17%, and 18% smaller than those of SB/DL, while (2) the parallel overhead of Phi/DL is 142% and 5% larger for the two smaller problem sizes and then 13% smaller for the largest.

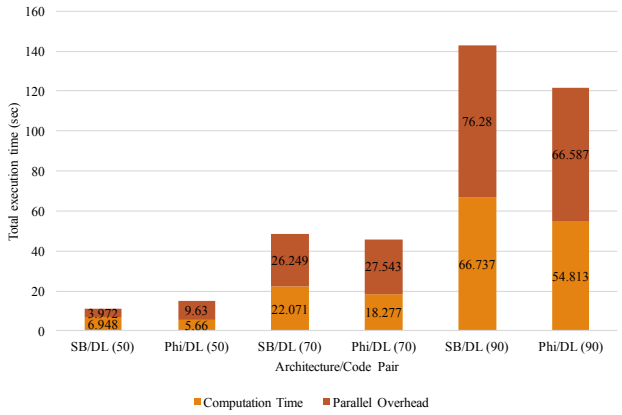

Figure 9: Computation time vs. parallel overhead.

Clearly, this behavior causes SB/DL to initially outperform Phi/DL and then allows Phi/DL to outperform SB/DL for the two larger problem sizes. Nonetheless, of concern is that for both, with the exception of SB/DL for the 50^3 problem size, the parallel overhead consumes more than 50% of the total execution time! Also, as the problem size increases, the SB/DL overhead grows faster than that of Phi/DL. In contrast, the computation time of Phi/DL grows slower than that of SB/DL. Thus, it appears that the gap in execution times will grow with the problem size, along with the superiority of Phi/DL over SB/DL.

In terms of scalability, for both SB/DL and Phi/DL the parallel overhead grows with both the problem size and the number of threads employed. As shown in Figures 10 and 11, the parallel overhead of executing Phi/DL with 60 or 120 threads is larger than that of executing SB/DL with two, four, or eight threads. However, when SB/DL is executed with 16 threads, the parallel overhead exceeds that of Phi/DL executed with 60, 120, or 240 threads. Phase analysis indicates that this is due to the larger overhead associated with the execution of Calc & Apply Accel, Lagrange, and

Time Constraints on the Sandy Bridge.

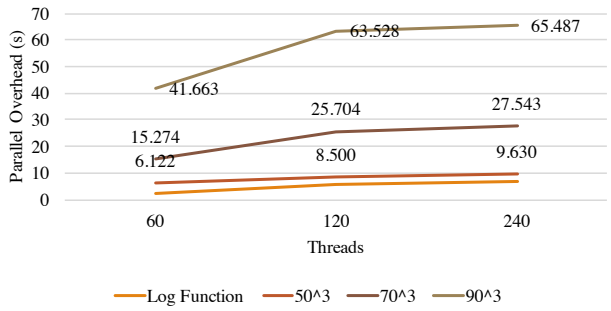

Figure 10: Scalability of Phi/DL parallel overhead.

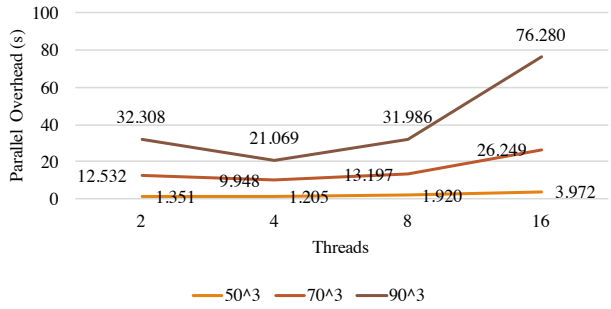

Figure 11: Scalability of SB/DL parallel overhead.

4. RELATED WORK

As new computing devices materialize, it is common for studies to emerge that exploit their features. And, an effective way to study their performance capabilities is through the use of benchmarks. Established benchmarks for studying multi- and many-core processors are the Rodinia [1], SHOC [2], and OpenDwarfs [6] benchmark suites. These benchmarks, which use algorithms in various domains to stress different processor components, have been used in several studies of accelerators. For example, [9] compares the many-core Intel® Xeon Phi™ to the Intel® Sandy Bridge Xeon E5-2620 multi-core processor and the many-core NVIDIA Tesla c2050 GPU (which employs the Fermi architecture). The SHOC benchmarks are used to compare the Phi™ with the Tesla in terms of power consumption and execution time, while the Rodinia benchmarks are used to compare the Phi™ to the Sandy Bridge in terms of execution time. The Phi™ outperforms the Tesla in terms of execution time for both compute- and memory-bound benchmarks, but it consumes more power. For compute-bound benchmarks, the Phi™ has shorter execution times than the Sandy Bridge, but for memory-bound benchmarks, the Sandy Bridge performs better. Although these findings can help determine which types of algorithms will perform better on a particular type of processor, they provide little insight as to why this is the case.

The need for a common set of metrics to characterize the performance and power efficiency of applications executed on heterogeneous systems inspired not only our study, but also the work presented in [11]. The set of metrics presented in [11] are based on the roofline model and measure: achieved bandwidth, execution time, power consumption, operational intensity, energy, and performance per Watt. First, using a set of micro-benchmarks, the roofline model for a given device is created and used to measure the device's peak performance and memory bandwidth. Next, these measurements are collected for several implementations of OpenCL kernels executed on the device and are compared to the established roofline model. Although it was demonstrated that this methodology is useful for comparing how well an algorithm utilizes the computing resources of a given set of devices, this particular study focuses only on the OpenCL programming model.

The use of peak performance to compare computing devices is not novel. In [14], it is used along with the time-to-solution of molecular dynamics (MD) applications to rank different types of supercomputers. The peak performance of the entire computing system is considered and the time-to-solution is specific to MD applications, i.e., it is defined as the time for one MD integration step to complete. Although of use for this particular purpose, this methodology is not applicable to understanding why an application performs as it does on a specific computing system.

Another property that has been used to compare different computing devices is programmability, which quantifies the complexity of using a device's programming model. Since ease-of-use is not a quantifiable metric, determining a device's programmability is a difficult task. Despite this, attempts have been made to assess the programmability of the NVIDIA Kepler and Intel® Xeon Phi™ MIC architectures. For example, in [7] the programmability of the Sandy Bridge (used as a baseline) was measured by counting the number of lines of source code that were introduced to develop a parallel code. In addition, common optimization techniques were implemented in the code and their impact on application performance was measured by calculating their achieved speedup. The highest speedup, i.e.,1,020.88X, was achieved using the Kepler, followed by the Xeon Phi™ with a speedup of 885.18X. However, the Kepler required much more manual tuning than did the Phi™. While these types of studies can be useful for determining if the effort required to port a code to a device is commensurate with the expected performance gains, they do not identify the causes of performance bottlenecks.

While there is a growing interest to establish a set of metrics that can be used to compare the performance of multi- and many-core processors, the differences in device architectures introduce several limitations. This is due, at least in part, to the fact that different processors do not expose the same metrics, and if they do, they may not be directly comparable. In this work, we address these challenges by comparing metrics that dissimilar devices expose, taking into consideration their architectural differences.

5. CONCLUSIONS AND FUTURE WORK

Our comparative study of the performance of versions of LULESH 1.0 executed on the Intel® Xeon Phi™ (Phi/DL) and the NVIDIA Fermi and Kepler GPUs (FGPU/F and KGPU/K) shows that KGPU/K runs up to 7X faster than FGPU/F and Phi/DL, which have comparable execution times. But, despite the variations in the architectural designs of these three accelerators, the distribution of their

execution times across the four main phases of execution of LULESH is similar. Phi/DL has the highest power and energy consumption, while KGPU/K has the lowest. However, unlike FGPU/F and Phi/DL, the power draw of KGPU/K grows similarly to how the problem size grows.

Just as the execution-time and power/energy performance of Phi/DL and FGPU/F are comparable, so are their memory behaviors, i.e., the Calc Volume Force and Lagrange phases of LULESH generate similar miss rates (from 85%-98%). Unfortunately, the limited number of available metrics that can be used to quantify a program's memory behavior on the three accelerators is not sufficient to explain its effect on their execution-time performance. Nonetheless, our IPC and vectorization usage data provide some insights: (1) KGPU/K has the best vectorization usage and highest IPC across the three problem sizes. (2) The IPC of KGPU/K increases with the problem size and appears to correlate with its lower execution times. (3) In contrast, although the execution times of FGPU/F and Phi/DL are comparable, the vectorization usage of FGPU/F is roughly 10% higher than that of Phi/DL, while the IPC of Phi/DL is consistently about 20% higher than that of FGPU/F. (4) Phi/DL and FGPU/F are similar w.r.t. IPC, vectorization usage, and memory behavior, and the clock rates of the Phi™ (1.1 GHz) and Fermi (1.15 GHz) are similar.

Currently, commonly used metrics like IPC and cache hit/miss rates cannot be directly compared across accelerators because of their architectural differences and the limited number of available hardware event counters. Nonetheless, we were able to identify reasons why LULESH 1.0 executed faster on the Kepler GPU, as compared to the Fermi GPU and the Xeon Phi™. And, in doing so, we introduced possible ways to circumvent the challenges of comparing the performance of disparate accelerators.

To expand and improve the contributions of this work, additional runtime execution configurations must be explored and other metrics must be considered, e.g., the transfer time between the accelerators and host processors should be considered. Also, the results of this study should be validated using additional applications executed on the studied computing devices. Finally, the power experiments should be refined to include finer-grain measurements and identify how the power draw of the GPUs is affected by their ability to spin up and down as they are loaded.

6. ACKNOWLEDGMENTS

This work was funded by the National Science Foundation Stampede grant through The University of Texas at Austin (UT-Austin) grant no. UTA13-000072 and the Department of the Army High Performance Research Center grant through the Stanford University sub award no. 60300261-107307-B. The authors wish to thank Edgar Leon and Ian Karlin of the Lawrence Livermore National Laboratories, as well as James Browne of UT-Austin, for suggesting this topic and providing guidance during our research.

7. REFERENCES

[1] 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 2009 IEEE Int. Symp. on Workload Characterization*, IISWC '09, pages 44–54, Washington, DC, USA, 2009. IEEE Computer Society.

[2] A. Danalis, G. Marin, C. McCurdy, J. S. Meredith, P. C. Roth, K. Spafford, V. Tipparaju, and J. S. Vetter. The Scalable Heterogeneous Computing (SHOC) Benchmark Suite. In *Proc. of the 3rd Workshop on General-Purpose Computation on Graphics Processing Units*, GPGPU-3, pages 63–74, New York, NY, USA, 2010. ACM.

[3] E. Gallardo. A Case Study of Accelerator Performance. Master's thesis, University of Texas at El Paso, El Paso, TX, 2015.

[4] J. Jeffers and J. Reinders. *Intel Xeon Phi Coprocessor High-Performance Programming*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1st edition, 2013.

[5] I. Karlin, A. Bhatele, B. L. Chamberlain, J. Cohen, Z. Devito, M. Gokhale, R. Haque, R. Hornung, J. Keasler, D. Laney, E. Luke, S. Lloyd, J. McGraw, R. Neely, D. Richards, M. Schulz, C. H. Still, F. Wang, and D. Wong. LULESH Programming Model and Performance Ports Overview. Technical Report LLNL-TR-608824, December 2012.

[6] K. Krommydas, W. C. Feng, C. D. Antonopoulos, and N. Bellas. OpenDwarfs: Characterization of Dwarf-Based Benchmarks on Fixed and Reconfigurable Architectures. *Journal of Signal Processing Systems*, pages 1–20, 2015.

[7] K. Krommydas, T. Scogland, and W. C. Feng. On the Programmability and Performance of Heterogeneous Platforms. In *Proc. of the 19th IEEE Int. Conf. on Parallel and Distributed Systems*, ICPADS, pages 224–231, Washington, DC, USA, 2013. IEEE Computer Society.

[8] J. LaGrone, A. Aribuki, and B. Chapman. A Set of Microbenchmarks for Measuring OpenMP Task Overheads. In *Proc. of the Int. Conf. on Parallel and Distributed Processing Techniques and Applications*, pages 594–600. Citeseer, 2011.

[9] 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 2014 IEEE Int. Parallel & Distributed Processing Symp. Workshops*, IPDPSW '14, pages 1448–1456, Washington, DC, USA, 2014. IEEE Computer Socieity.

[10] P. J. Mucci, S. Browne, C. Deane, and G. Ho. PAPI: A Portable Interface to Hardware Performance Counters. In *Proc. of the Dept. of Defense HPCMP Users Group Conf.*, pages 7–10, 1999.

[11] S. Muralidharan, K. O'Brien, and C. Lalanne. A Semi-Automated Tool Flow for Roofline Analysis of OpenCL Kernels on Accelerators. In *Proc. of the 1st Int. Workshop on Heterogeneous High-performance Reconfigurable Computing*, H2RC'15, 2015.

[12] NVIDIA. NVIDIA System Management Interface. Retrieved from: https://developer.nvidia.com/nvidia-system-management-interface.

[13] NVIDIA. NVIDIA Visual Profiler. Retrieved from: https://developer.nvidia.com/nvidia-visual-profiler.

[14] V. V. Stegailov, N. D. Orekhov, and G. S. Smirnov. HPC Hardware Efficiency for Quantum and Classical Molecular Dynamics. In *Parallel Computing Technologies*, pages 469–473. Springer, 2015.