2015 44th International Conference on Parallel Processing

GPGPU Benchmark Suites: How Well Do They Sample the Performance Spectrum?

Jee Ho Ryoo, Saddam J. Quirem, Michael LeBeane, Reena Panda, Shuang Song and Lizy K. John
The University of Texas at Austin
{jr45842, saddam.quirem, mlebeane, reena.panda, songshuang1990}@utexas.edu, ljohn@ece.utexas.edu

Abstract

Recently, GPGPUs have positioned themselves in the mainstream processor arena with their potential to perform a massive number of jobs in parallel. At the same time, many GPGPU benchmark suites have been proposed to evaluate the performance of GPGPUs. Both academia and industry have been introducing new sets of benchmarks each year while some already published benchmarks have been updated periodically. However, some benchmark suites contain benchmarks that are duplicates of each other or use the same underlying algorithm. This results in an excess of workloads in the same performance spectrum.

In this paper, we provide a methodology to obtain a set of new GPGPU benchmarks that are located in the unexplored region of the performance spectrum. Our proposal uses statistical methods to understand the performance spectrum coverage and uniqueness of existing benchmark suites. Later we show techniques to identify areas that are not explored by existing benchmarks by visually showing the performance spectrum coverage. Finding unique key metrics for future benchmarks to broaden its performance spectrum coverage is also explored using hierarchical clustering and ranking by Hotelling's T^2 method. Finally, key metrics are categorized into GPGPU performance related components to show how future benchmarks can stress each of the categorized metrics to distinguish themselves in the performance spectrum. Our methodology can serve as a performance spectrum oriented guidebook for designing future GPGPU benchmarks.

1. Introduction

GPU architects and designers need diverse GPGPU benchmarks in order to guide their designs. As of today, there are more than 100 GPGPU benchmarks from different suites available on the web for the public. The number of benchmarks in each suite vary from a few to more than fifty. Some suites are geared toward testing one domain of applications while others attempt to address a large range of applications [13, 30, 31, 37, 38]. The number of GPGPU benchmarks have been rising at a fast pace as newer GPGPU benchmarks are released every year at various research venues [3, 6, 9, 13, 36].

On one hand, this large number of benchmarks can help researchers stress various components of their proposed future microarchitectures, covering a large amount of variance in the performance spectrum [15, 16, 32]. However, many benchmarks may exhibit similar behaviors from performance perspectives [8]. In CPU benchmarking such as SPEC CPU2006, each time a new suite is released, a committee ensures that benchmarks show distinct behaviors and that the suite has a large coverage in performance spectrum [35]. However, GPGPU benchmarks have not been gone through such a rigorous process prior to release. Thus, they are prone to being duplicates of existing benchmarks in some other suites or very similar to each other. In this situation, some methodological guidance can be beneficial to indicate which benchmarks are redundant and which ones form a unique group of workloads to evaluate GPGPUs.

This paper uses various statistical methods to evaluate existing benchmarks in the GPGPU domain. We show the distribution of existing benchmarks using Principal Component Analysis to locate them in the performance spectrum and suggest where future benchmarks can explore. In order to achieve this, we present methods to identify unique benchmarks and their key metrics that exercise extreme ends of the spectrum. Then, we categorize these metrics into different GPGPU performance related components as this will illustrate how a new benchmark with metrics in one isolated GPGPU component can be used to stress an unexplored performance spectrum region. Using data collected from real hardware, we investigate which direction new benchmarks should stress. This paper makes the following contributions:

1. We provide a methodology for future GPGPU benchmarks to increase the performance spectrum coverage.
2. We perform various statistical analyses to show the suite-wise coverage on the performance spectrum and identify key performance coverage areas that are not explored by existing suites.
3. We use hierarchical clustering and ranking to identify unique benchmarks that are located at extremes of the performance spectrum. These benchmarks contain many distinctive features that can guide the future benchmark suites to be unique.
4. We analyze a very large set of benchmarks (88 benchmarks from 5 different benchmark suites) that are widely used in the GPGPU performance evaluation domain. To our

0190-3918 2015

DOI 10.1109/ICPP.2015.41

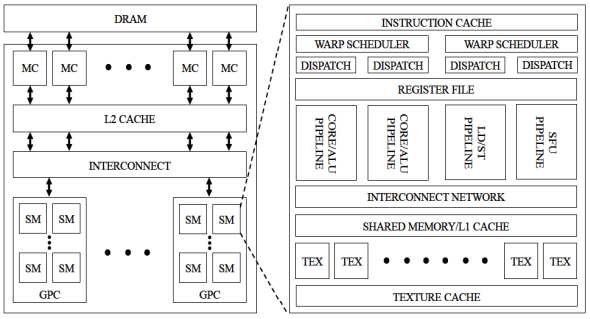


Figure 1: GPU Architecture Overview

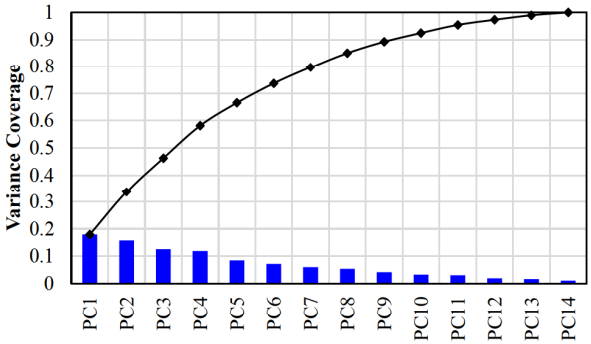


Figure 2: Variance Coverage of PCs

knowledge, no prior work has explored such a complete benchmark space due to lengthy simulation time as well as infrastructure issues.

The rest of this paper is organized as follows. Section 2 provides the background of GPU microarchitectural and programming model. Section 3 provides the overview of our experimental setup and methodology while we evaluate our results in Section 4. We perform the component-wise study in Section 5. Section 6 discusses prior work done in this area, and we provide the concluding remarks in Section 8.

2. Background

2.1. GPU Architecture and Programming Model

Figure 1 displays an overview of a typical GPU's microarchitecture [24]. The processing cores of the GPU are organized in Graphics Processing Clusters (GPCs), each of which contains a set of Streaming Multiprocessors (SMs) [12]. All SMs share a common L2 cache, which is the Last Level Cache (LLC). The SM contains dual warp-schedulers where a warp, a group of 32 threads, executes in SIMT fashion [24, 25]. Warp instructions can be dispatched to the core/ALU pipeline, load/store pipeline, or Special Function Unit (SFU) pipeline as shown in the right side of Figure 1. The memory subsystem has a unique type of memory known as shared memory. The name refers the fact that this memory is shared by all threads in the same Cooperative Thread Array (CTA). The size of the CTA is a programmer configurable parameter. In some architectures, the programmer has the capability to sacrifice L1 cache capacity in favor of a larger shared memory or vice-versa. Besides the L1 cache, a texture cache is used for memory accesses with high spatial locality while a constant cache is used for memory accesses with high temporal locality.

Such an architecture is coupled with heterogeneous languages such as OpenCL and CUDA [1, 25]. The language creates an abstraction of hardware structures present in the GPU. The programmer has control over the number of threads that are launched for a particular kernel by passing the number of CTAs, known as thread blocks in CUDA and work-groups in OpenCL. Then, the runtime hardware determines the number of warps in each CTA. The order in which the CTAs execute is non-deterministic [18]. Besides the fact that they are able to share the memory, threads within a CTA are able to synchronize with other threads in the CTA using a barrier instruction. The GPU will attempt to schedule as many warps as possible in the same SM. The ratio between number of warps executing on an SM and the maximum number of warps which can execute on an SM is known as warp occupancy. Threads within a warp can diverge from each other resulting in inactive SIMD lanes. There exists two types of divergence in GPU SIMD architectures, control-flow divergence and memory divergence [33]. Control-flow divergence is the result of a thread within a warp taking a different path than the other threads in the same warp. Memory divergence is the result of an uncoalesced memory operation where the memory locations accessed by the threads in a warp are unaligned and/or not adjacent.

2.2. Principal Component Analysis

In this section, we explain the Principal Component Analysis (PCA) technique, which is used throughout the paper to show the distribution of workloads among different benchmark suites in the performance spectrum [11]. PCA converts i variables $X_1, X_2, \ldots, X_i$ into j linearly uncorrelated variables $\hat{X_1}, \hat{X_2}, \ldots, \hat{X_j}$, called Principal Components (PCs). Each Principal Component is a linear combination of the various variables (characteristics) weighted with a certain coefficient, known as the loading factor, as shown in Equation 1. It has an interesting property that the first PC covers the most variance (information from the original data) while the second PC covers the second most variance (PC1 contains the most amount of information about the data). In this work, we have a total of 14 PCs with the individual and the cumulative variance coverage shown in Figure 2.

$$\hat{X_1} = \sum_{k=1}^{i} a_{1k} X_k \qquad \hat{X_2} = \sum_{k=1}^{i} a_{2k} X_k \qquad \cdots \tag{1}$$

3. Methodology

The applications were selected from a set of suites which are widely used for GPGPU performance evaluation. In this

Table 1: GPGPU Workload Sources

Workload Suite	Version	# of Applications
NVIDIA SDK	6.0	39
Rodinia	3.0	21
Parboil	2.5	9
Mars	Initial Version	7
GPGPU-Sim	Initial Version	12

Table 2: Experimental Platform Specifications

Name	Value
#SM	7
Processor Cores	384
Graphics Clock (MHz)	822
Processor Clock (MHz)	1645
L2 Cache Capacity (KB)	512
DRAM Interface	GDDR5
DRAM Capacity (GB)	1.28
DRAM Bandwidth (GB/s)	128
Peak GFLOPs	1263.4

section, we discuss these GPGPU application suites in terms of how the various application metrics are collected, how the benchmarks are selected, and what program characteristics are selected for workload characterization.

3.1. Workloads

We use 5 benchmark suites listed in Table 1. For completeness of the analysis, every application in all suites is included in the analysis regardless of whether one suite has a similar benchmark to another suite. For example, we run two versions of the histogram-generating applications from both the *CUDA_SDK* and *Parboil* suite. Our workload suites include the widely used *Rodinia* benchmark suite, the *Parboil* benchmark suite, the *GPGPU-Sim* suite presented at ISPASS-2009, the *Mars* MapReduce framework sample applications, and the sample applications included with the *CUDA_SDK* [3, 6, 7, 13, 22, 36]. The largest number of workloads come from the *CUDA_SDK* sample applications which cover the graphics (*MAND*), image-processing (*DXTC*), financial (*BS*), and simulation (*NBODY*) domains. The *Rodinia* and *Parboil* suites aim for diversity in the benchmarks. The *GPGPU-Sim* and *NVIDIA_SDK* suites are designed for demonstration purposes, while at the same time, providing diversity in their workloads. The *Mars* sample applications are very similar to each other in terms of their code and algorithms because they all depend on a map reduce framework.

A large body of simulation based workload characterization uses reduced input set sizes or shorter runs due to slower simulation speed. In our work, all workloads are executed until completion with representative input sizes. Figure 3 shows the number of warp instructions in all benchmarks that are used in our study. The number of warp instruction counts range from 36,822 in *DWTHAAR* to 43 billion in *CFD*. Due to a large variance in the number of instructions executed, we use a logarithmic scale in Figure 3. Our evaluation was done on hardware, so the total number of instructions are significantly higher than the prior work done on a simulator [12]. Longer execution time with more representative input sizes can accurately depict the performance characteristics of these GPGPU applications.

Table 3: Description of Metrics

Metric	Description
REPLAY_OVERHEAD	Average number of replays for each 1K instructions executed
BRANCH_EFF	Ratio of non-divergent branches to total branches
WARP_EXEC_EFF	Ratio of the average active threads per warp to the maximum number of threads per warp supported on a multiprocessor expressed as percentage (SIMD efficiency)
LOAD_STORE_INST	Compute load/store instructions executed by non-predicated threads per total instructions executed
CONTROL_INST	Control-flow instructions executed by non-predicated threads per total instructions executed
FP_INST	Floating point instructions executed per total instructions executed
MISC_INST	Miscellaneous instructions executed per total instructions executed
SHARED_LDST_INST	Shared load/store instructions executed per total instructions executed
DP_FLOP_PKI	Double-precision floating-point operations executed by non-predicated threads per 1K instructions executed
TEX_MPKI	Texture cache misses per 1K instructions executed
DRAM_WRPKI	Device memory write transactions per 1K instructions executed
L1_MISS_RATE	Miss rate at L1 cache
L2_L1_MISS_RATE	Miss rate at L2 cache for all read requests from L1 cache
TEX_MISS_RATE	Texture cache miss rate

3.2. Experimental Setup

The program metrics listed in Table 3 are collected using with NVIDIA's CUDA (6.0) profiler, known as *nvprof*, on an NVIDIA GTX 560 Ti GPU [23, 26]. Table 2 lists detailed specifications of our underlying hardware platform. The program metrics are collected from reproducible runs of each application. The profiler is capable of retrieving multiple metrics by replaying GPU kernels.

4. Evaluation

4.1. GPGPU Workload Characteristics

The complexity and diversity of GPGPU workloads continue to increase as they now are tackling larger problems and expanding across various fields. In order to analyze performance in such workloads, we perform analysis on a comprehensive set of program characteristics. Table 3 provides a list of the performance metrics used in this paper. The total number of dynamic instructions are used to scale many metrics in order to eliminate effects caused by a different number of instructions. For example, the total number of texture cache misses is divided by 1000 instructions to show the miss rate per kilo-instruction (TEX_MPKI).

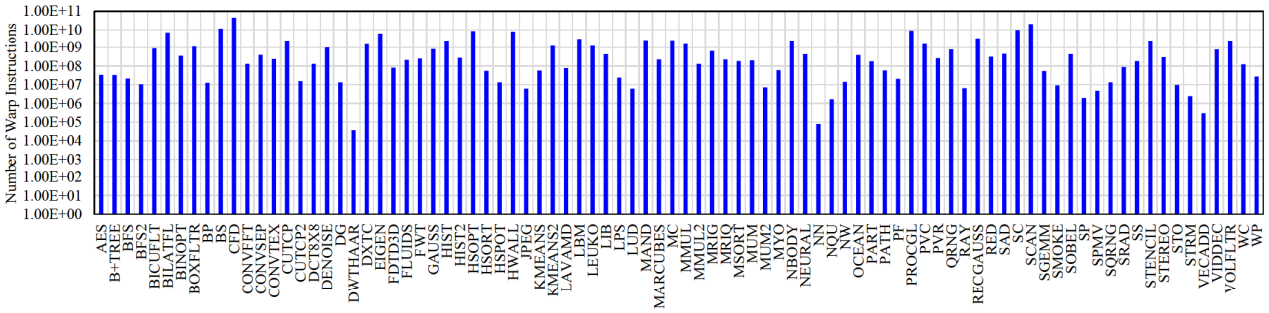


Figure 3: Number of Executed Instructions (Logarithmic Scale)

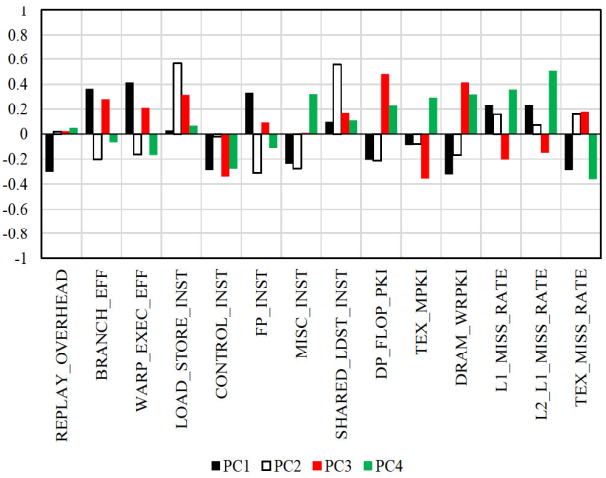


Figure 4: Factor Loading

The PCA algorithm projects each data point onto the new axis, the PC. This is done by multiplying each metric by an appropriately weight, called factor loading. The weighted metrics are then summed to compute each PC. The weights can show which metrics have significance in each PC. Figure 4 shows the weight of each metric on first four PCs. For example, WARP_EXEC_EFF, BRANCH_EFF and FP_INST are the metrics that affect PC1. Those metrics with the high PC1 factor loading are key metrics that distinguish benchmarks among each other since PC1 covers the most variance. In subsequent sections, we will use this background knowledge to explain various results in detail.

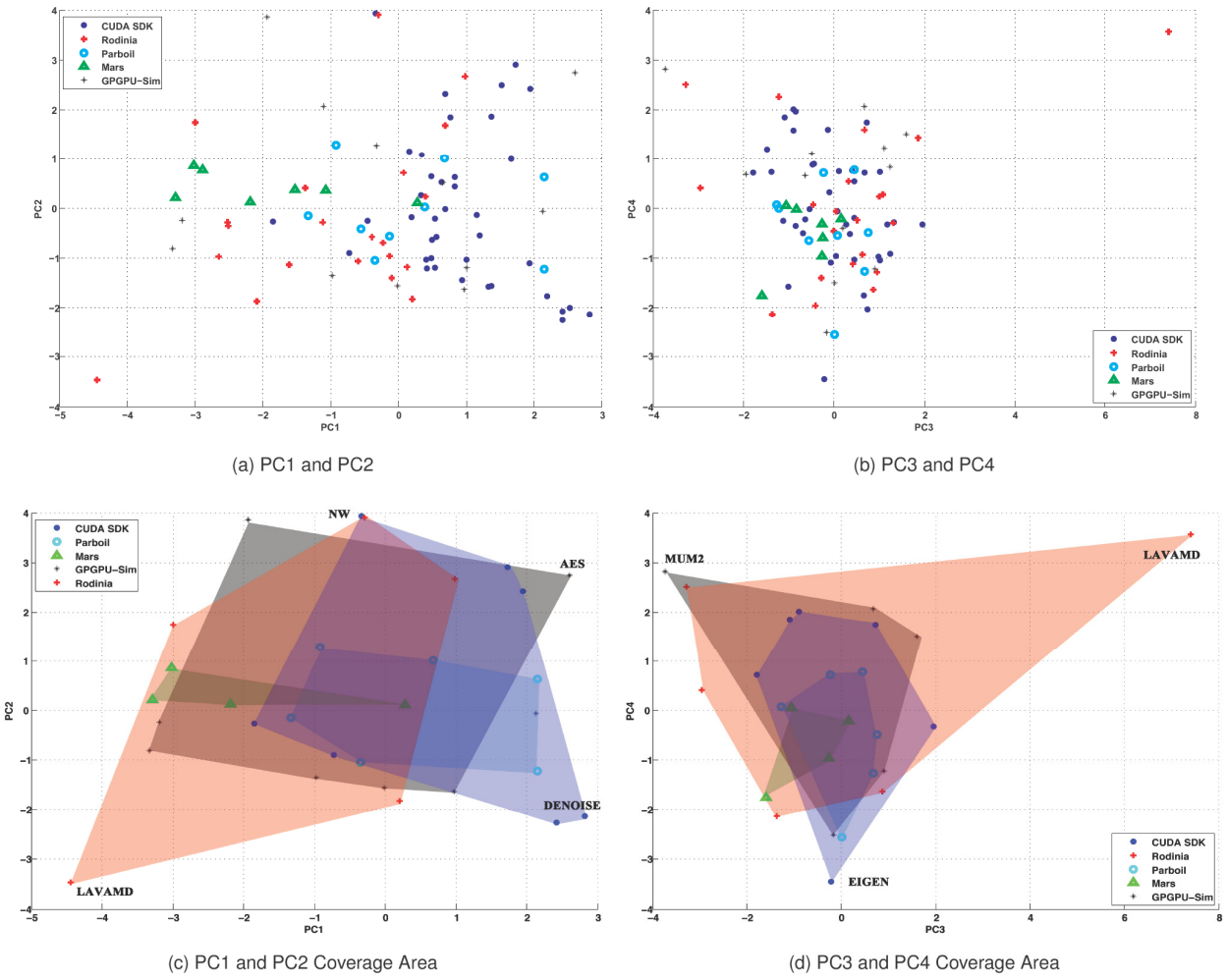
4.2. Performance Spectrum Coverage

We evaluate with the PC scatterplot where each benchmark is located in the two-dimensional performance spectrum. Figure 5 visually shows the position of each benchmark and the suite-wise coverage map in the performance spectrum. We do not label each data point to avoid cluttering the plot. However, we will explicitly mention the name of some interesting benchmarks in subsequent sections. Figure 5a shows the scatterplot for PC1 and PC2. These two PCs cover approximately 35% of the variance as previously shown in Figure 2.

The *CUDA_SDK* suite has the most number of data points with the largest number of benchmarks. Many benchmarks lie on the right side of the scatterplot. The PC1 axis, where WARP_EXEC_EFF has the highest weight, controls the horizontal placement, meaning the benchmark suite has many benchmarks with high SIMD efficiency. Other metrics with high PC1 factor loading are BRANCH_EFF and FP_INST, which are common factors that result in high performance in GPGPUs. This suite is a good representative of GPGPU applications with optimized control flow patterns as they reduce inefficiencies caused by not fully utilizing the available SIMD lanes. Also, a high number of floating point operations usually indicate a compute intensive workload, and GPGPUs process these operations very efficiently. At the same time, this means that the suite is missing a set of workloads that are memory intensive as well as highly control divergent. Figure 5b shows that this suite is distributed in a narrow region on the PC3 axis, which is dominated by DP_FLOP_PKI and DRAM_WRPKI. The suite does not have a set of benchmarks that extensively exercise the DRAM. Figure 5d shows the coverage map of each suite. It is noticeable that the width of the *CUDA_SDK* suite is rather narrow.

Rodinia, on the other hand, covers the left half of PC1 in Figure 5a, which almost complements the missing area of the *CUDA_SDK*. This is a good example of how two different suites can yield different results when both suites do not have a large coverage. For example, if a future architecture with mores registers were proposed to improves the warp occupancy, *CUDA_SDK* would not see much benefit as most benchmarks already have high warp occupancy. Yet, the *Rodinia* will see significant benefits as the suite is composed of many benchmarks with low warp occupancy. In this case, depending on which suite is used, the outcome can be drastically different and the decision for future architectures may change. *Rodinia* covers a wide spectrum in Figure 5d with many benchmarks being located at different extreme points of the coverage envelop. These PC axis show the memory intensiveness such as cache misses and DRAM accesses, illustrating that this suite has a variety mix of benchmarks that stress the memory subsystem in different ways.

GPGPU-Sim only has 12 benchmarks in the suite, yet the PC1 and PC2 coverage in Figure 5c is rather large. Those



(a) PC1 and PC2

(b) PC3 and PC4

(c) PC1 and PC2 Coverage Area

(d) PC3 and PC4 Coverage Area

Figure 5: Scatterplot Showing 4 PCs

benchmarks are well spread out in the control flow spectrum since one benchmark has high SIMD efficiency while another shows low SIMD efficiency. This suite not only covers a large spectrum in the control flow space, but also in the memory space as in Figure 5d. L2_L1_MISS_RATE and L1_MISS_RATE have high weights in the PC4 factor loading, representing memory intensive workloads. This suite stretches vertically with a good mix of workloads that show both high and low cache misses.

Parboil has a small number of benchmarks, yet their performance spectrum lies completely inside the *CUDA_SDK*'s spectrum as seen in Figure 5c and Figure 5d. The suite tries to cover a large area as the coverage spectrum is not narrow in one particular axis direction. This attempt is shown in Figure 5a and Figure 5b as all data points are not clustered in one region, but rather, set apart out from each other. Yet, it does not exercise any component to the extreme, so the area that this suite covers in the spectrum is relatively small. With future processors that improve such bottlenecks as cache misses, it is possible that the spectrum coverage will shrink further since metrics such as L1_MISS_RATE will decrease. However, it is possible that the existing benchmarks can increase the coverage area with some improvements. For instance, the metrics that affect Figure 5a are efficiency metrics such as the SIMD efficiency. The lack of hardware resources within the SM usually affects these metrics, so if the program uses more registers per thread or has more divergent branches, this will drag a few of the *Parboil* benchmarks to the lower end of the spectrum in PC1. Similarly, the input size of the application can be made larger so that it can generate more memory related events such as cache misses. This will eventually lead some benchmarks to extreme points in Figure 5d.

The *Mars* benchmark suite is designed to use the same map-reduce framework, so their coverage is expected to be small as shown in Figure 5. This suite is enclosed completely by the *Rodinia* suite. If this suite can cover an area that is not covered by any other suites, then it can stand out as a very unique benchmark suite. This suite lies near the border of the *Rodinia* coverage envelop, so if it can be stretched little farther towards one extreme, it can cover its own unique space. In the

case of Figure 5c, this will be towards the negative end of the PC1 axis. This means that if one of the benchmarks contains slightly lower SIMD efficiency, it can be located outside the *Rodinia* envelop.

In Figure 5, many areas are still left to be explored. Especially, the area with positive PC3 and negative PC4 is one example. This area corresponds to high floating point instructions, high DRAM read accesses, and high texture cache misses. A new benchmark to explore this area can be proposed with a high number of floating point instructions. The texture and L2 misses can be increased by increasing the texture cache working set size. Therefore, if a new benchmark is written with these in mind, it can have a large number of read accesses to the texture cache with a large footprint, which eventually reaches the DRAM. In addition, the lower PC1 and high PC2 area is another unexplored area, which corresponds to low SIMD efficiency, high load/store instructions and shared load-/store instructions. For this area, the future benchmark can be written with low SIMD efficiency. Yet, the warp instruction mix in this benchmark can be mainly composed of memory instructions. Then, it can eventually fill the missing space in the performance spectrum.

4.3. Similarity/Dissimilarity of Benchmarks

With many data points in the multidimensional PC scatterplot, it is difficult to see similarity between benchmarks. More importantly, if many existing benchmarks are similar to each other, then future benchmarks should be designed to be dissimilar to those ones in order to place themselves in unexplored areas. The dendrogram can help this process by showing how similar/dissimilar each benchmark is among 88 benchmarks used in this study. A dendrogram is a graphical representation of the hierarchical clustering method to represent the similarity among benchmarks. First, all metrics in Table 3 are drawn in the multidimensional space. Then, the pairwise distance of all benchmarks is computed and a tree is constructed based on the distance of each pair as shown in Figure 6. If a pair of two benchmarks is very similar, it will have a short distance, and thus, will be linked at the leafs of the tree (left side of the figure). At the end, a group of most dissimilar benchmarks are linked at the root of the tree. Subsetting a large number of benchmarks can easily be done with the dendrogram. If one wants to determine a subset of 2 benchmarks, then draw a vertical line which has two intersections [21]. In our figure, a dotted line drawn at the linkage distance value of 12 corresponds to this point. One intersection corresponds to the *Rodinia-LAVAMD* benchmark, and the other intersection corresponds to 87 other benchmarks [27]. This means the *Rodinia-LAVAMD* is the most unique program, and the program at the center of the other subset with the 87 programs will be used to represent that subset.

In Figure 6, we see that all of the *Mars* benchmarks are found to be similar, suggesting that a single map-reduce application is sufficient when used in benchmarking. A similar

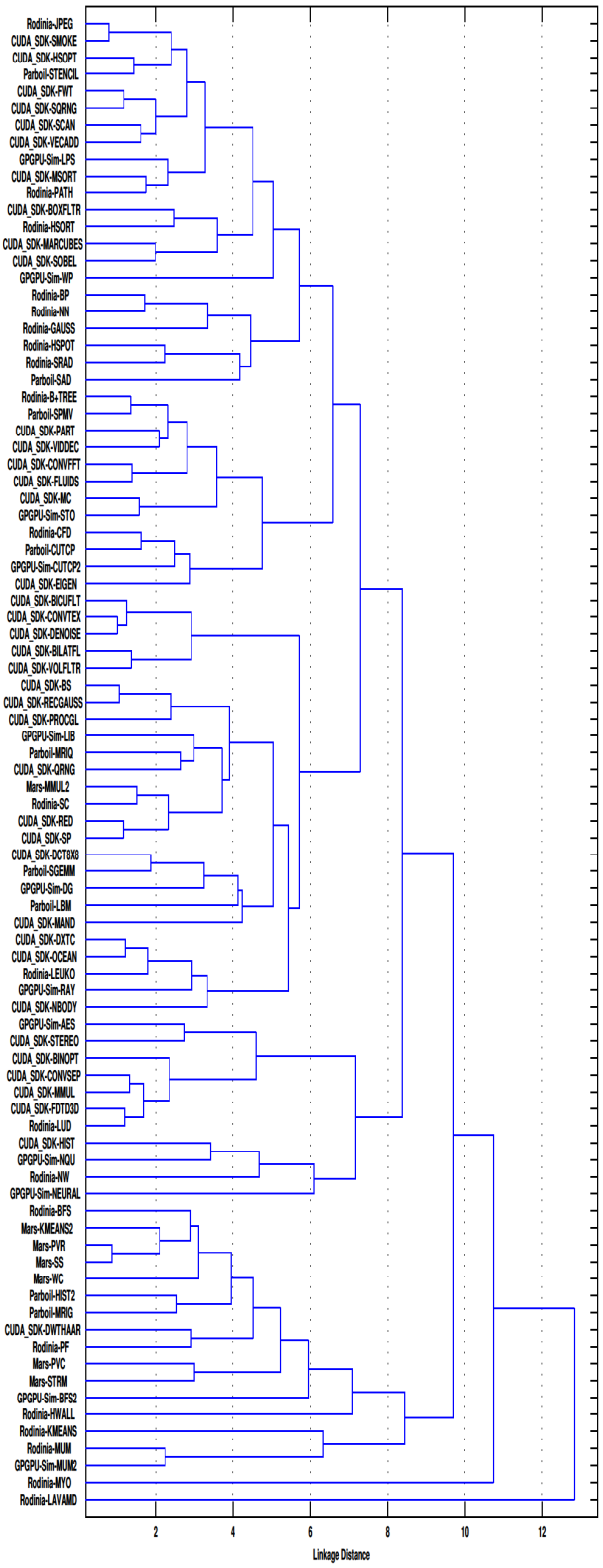


Figure 6: Dendrogram

Table 4: Uniqueness Rank of Benchmarks

Benchmark	Rank	Benchmark	Rank
Rodinia-LAVAMD	1	Rodinia-MUM	9
Rodinia-MYO	2	GPGPU-Sim-NQU	10
GPGPU-Sim-NEURAL	3	GPGPU-Sim-WP	11
Rodinia-NW	4	Mars-STRM	12
Rodinia-HWALL	5	GPGPU-Sim-AES	13
GPGPU-Sim-MUM2	6	GPGPU-Sim-STO	14
Rodinia-KMEANS	7	CUDA_SDK-MAND	15
GPGPU-Sim-BFS2	8	Parboil-LBM	16

result is shown with the image processing benchmarks of the *CUDA_SDK* suite, specifically *BICUFLT*, *BILATFL*, *DENOISE*, and *CONVTEX*. Besides those benchmarks found in the same domain, some were found to be similar due to the usage of a common algorithm, namely scalar-produce(*SP*) and reduction(*RED*). Parallel reduction is a major phase of scalar-product calculation.The results indicate that future benchmarks need to have workloads that are distinct from image processing or map-reduce to be unique in the performance spectrum. Also, future benchmarks should avoid using existing benchmarks' algorithms to eliminate redundancy.

A general goal in benchmarking is to increase the performance spectrum coverage. A good benchmark suite will have a large number of benchmarks that are connected at the root of the tree. The dendrogram in Figure 6 can indicate that more types of benchmarks such as *Rodinia-LAVAMD* and *GPGPUSim-MUM2* are needed. Not surprisingly, those are benchmarks that are located in their unique spaces in the PC scatterplot presented in Section 4.2. Therefore, an effective visual representation of the dendrogram can drive where new benchmark efforts should go in the performance spectrum.

4.4. Ranking Benchmarks

A new benchmark should target a unique space in the performance spectrum in order not to overlap with existing benchmarks or suites. The dendrogram can be effective in showing a few unique benchmarks, but with such a large number of benchmarks, it is hard to illustrate how unique one benchmark is relative to another. Especially in our study with 88 benchmarks, choosing unique benchmarks just by looking at the dendrogram can be difficult. Hotelling's T^2 method [17] can help identifying unique programs. It is a statistical method to find the most extreme data point in multivariate distribution. The benchmark that is farthest away from the center of the distribution is marked as the most unique benchmark in this study. Ranking benchmarks based on Hotelling's T^2 helps us select unique benchmarks when the subsetting the dendrogram results in that one subset contains too many benchmarks. Table 4 lists the top 16 unique benchmarks. The top two benchmarks agree with the dendrogram as they had the largest linkage distance from other benchmarks. We found that the top 36 benchmarks cover 90% of the total variance in the performance spectrum.

In Figure 5, we saw that the *Rodinia* suite had many benchmarks that are located at extreme PC values. The table aligns

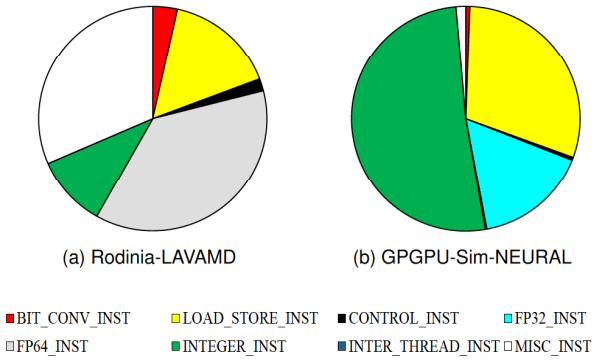
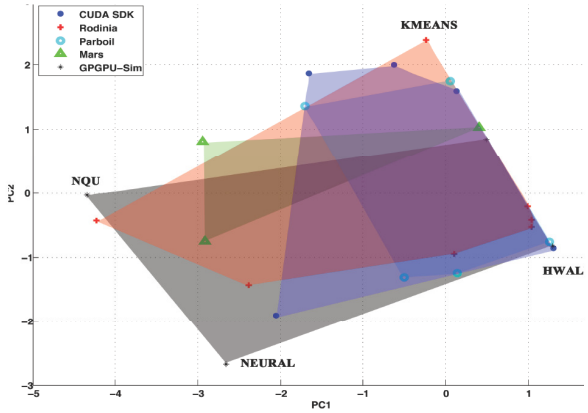


Figure 7: Instruction Mix of 2 Most Dissimilar Programs

with the prior finding since the 5 *Rodinia* benchmarks made it in the top 8 list. Although the ranking itself does not show which end of the extremes the benchmark exists, the methodology we have developed throughout the paper, including the PC scatterplot, can clearly show where high ranking benchmarks exist in the scatterplot. The top 5 *Rodinia* benchmarks are located at the edges of the coverage envelop in Figure 5c. In addition, the *GPGPU-Sim* suite also has 7 benchmarks in the top 16 list. Although the *Parboil* and *Mars* suite have a relatively small number of benchmarks, a few benchmarks such as *Parboil-LBM* and *Mars-STRM* are included in the top 16 list as they are located near the edge of the envelop.

Unique benchmarks can help future benchmark designers with important architectural components. They can offer insights into the source of underlying features that make them unique. Here, we will dive into the three most unique benchmarks, *LAVAMD*, *MYO*, and *NEURAL*. *LAVAMD* is unique among the *Rodinia* suite as it is the only benchmark representing the N-Body dwarf [2, 27]. This benchmark features a high branch and warp execution efficiency as well as efficient shared memory access patterns. Figure 7a explains another unusual characteristic of *Rodinia-LAVAMD* as it has a high number of miscellaneous instructions, which include barrier instructions. Also, the problem size is partitioned to fit in constant memory, so the computation is done only within each thread block, achieving high SIMD efficiency. However, even though *MYO* is a floating-point benchmark just like *LAVAMD*, it features very low memory efficiency. Each instance of this workload is assigned to a thread, so this workload does not orchestrate memory accesses to take advantage of caches such as constant memory. Therefore, this benchmark incurs significantly more cache misses than others. *NEURAL* operates on floating-point data, but over half of its instructions are integer operations which are used for address calculation as shown in Figure 7b. Although this benchmark has a high cache miss rate, it shows a low cache misses per kilo-instruction because a large portion of memory loads are hits from the constant memory. From our example using *LAVAMD*, *MYO*, and *NEURAL*, we can illustrate which algorithm or programming style



(a) PC1 and PC2 Coverage Area

Figure 8: Divergence PC Scatterplot

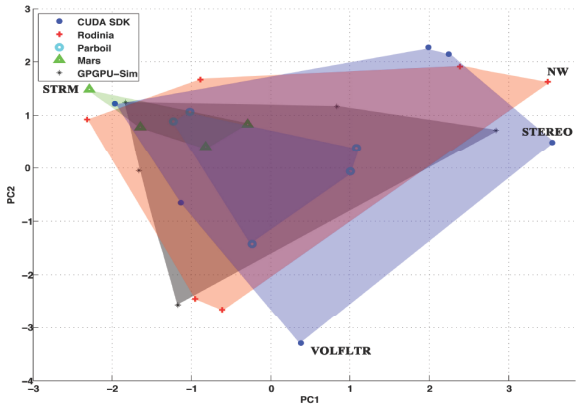
to use in order to design new benchmarks that are unique and do not overlap with existing ones.

5. Component-Wise Analysis

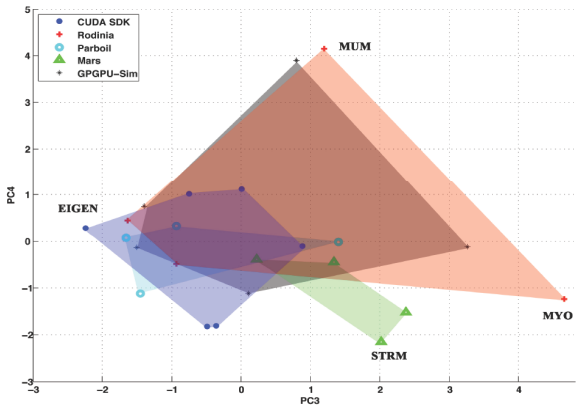
Until now, we have used all metrics that are either computation or memory related. Working with all at the same time to understand the performance spectrum can be difficult as the GPGPU architecture is sophisticated. In this section, we perform a component-wise analysis to show where future benchmark efforts should go to target major components of the GPGPU architecture.

5.1. Control Divergence

We perform the spectrum coverage analysis with only control divergent metrics as they are key GPU performance factors. Our initial analysis in Section 2.2 showed that BRANCH_EFF and WARP_EXEC_EFF affect PC1 significantly, so we conduct the performance spectrum coverage study with BRANCH_EFF, WARP_EXEC_EFF, and CONTROL_INST metrics. PC1 and PC2 are shown in Figure 8. It shows an interesting result where *CUDA_SDK* does not cover the second largest area in the scatterplot anymore. Especially, in our study where each suite has a different number of benchmarks, it is advantageous for *CUDA_SDK* as it potentially has more data points to cover a large area. Yet, Figure 8 shows that this suite with 39 benchmarks, covers a similar area as the *Parboil* suite, which only has 9 benchmarks. PC1 and PC2 covers 84% of the variance in our control divergent study, so they are considered significant in our statistical space. Most benchmarks written for heterogeneous systems aim for SIMD efficiency. However, some benchmarks such as *NEURAL* has lower SIMD efficiency due to a high amount of divergent predication. Furthermore, other benchmarks such as *NQU* also have lower SIMD efficiency, but as a result of high branch divergence. Many benchmark suites are clustered in the upper right region corresponding to the area with high SIMD efficiency. With the characteristics of those benchmarks at



(a) PC1 and PC2 Coverage Area



(b) PC3 and PC4 Coverage Area

Figure 9: Memory PC Scatterplot

the performance spectrum extremes, the missing space corresponds to extremely low SIMD efficiency. This can lead to a suggestion that future benchmarks with a large number of map reduce operations can fill the space since these operations usually have many unused SIMD lanes.

5.2. Memory

The GPGPU memory subsystem is one of core areas with performance bottlenecks as well as performance improvement potentials. Uncoalesced memory accesses or codes not written to take advantage of caches affect performance significantly. There is a high demand of research interests in this area, and we show which benchmark suites exercise the memory subsystem in various ways. Figure 9 shows the area coverage plot of different suites. Interestingly, most benchmarks cover a large area in both the PC1 and PC2 space. PC1 is largely dominated by memory related instructions whereas PC2 is dominated by cache misses. However, some benchmark suites span more of the spectrum in PC3 and PC4. In Figure 9b, the *Rodinia* and *GPGPU-Sim* suite cover a large area. The first 4 PCs cover 74% of the variance in the memory subsystem performance space. Although the *Mars* suite covers a rela-

tively small area, it exercise areas that are not covered by any other benchmark, placing itself in a unique position. It can be inferred that most benchmarks have a similar distribution of memory related instructions and cache misses, which explains why all benchmark suites are well spread out in Figure 9a. However, the texture cache miss behaviors diverge from each other as some benchmarks do not exercise the texture cache heavily. As a result, some benchmarks start to dominate the area coverage in the PC3 and PC4 space.

Uniqueness in memory performance behaviors among the benchmarks stems from the instruction mix. While applications such as *MMUL2* represent the bulk of the workloads where around 15% of instructions are load/store instructions, applications such as *STEREO* are composed of 40% load/-store instructions, increasing the impact of cache misses. If the percentage of load/store instructions is low, the impact of cache misses is relatively low (MPKI of the L1, L2, and texture caches will be low). Image filtering benchmarks such as *VOLFLTR* are unique in memory access behavior because they contain very few load/store applications. In this benchmark, each thread only needs to reference an input image once and continuously performs computation with this image. Therefore, future benchmarks can be formed with either extremely high and low memory instruction mix as they will be able to exercise both ends of the untouched memory performance spectrum. Even existing benchmark suites can have different input sizes that stress the memory subsystem towards both extremes to increase the area coverage in the performance spectrum.

6. Related Work

Kerr et al. have performed earlier work on characterizing the PTX kernels of the benchmarks using Ocelot dynamic compiler [20] *Parboil* and *CUDA_SDK* [10]. They later characterized heterogeneous workloads using a wider set of metrics, including both static and dynamic instruction counts [19]. Yet, their work is focused on finding the program characteristic similarities between the GPU and CPU processors. Goswami et al. performed the workload characterization of selected GPGPU workloads using GPGPU-Sim on different hardware configurations [3, 12]. However, many benchmarks including those using the texture cache cannot be executed on the simulator, thereby leaving an entire class of workloads out of the analysis. Input sizes are also limited so that the simulations would complete within a reasonable time frame. Che et al. have also utilized PCA analysis to characterize their GPGPU workloads when constructing the *Rodinia* suite itself, but their work was limited on *Parsec*, *SPLASH-2* and their own benchmark suite [4, 6, 39]. Unlike our methodology where we use various statistics techniques for guidance purpose, they use PCA only for validation. Phansalkar et al. performed the PCA analysis work for the SPEC CPU2000 and CPU2006 benchmark suites to detect redundant benchmarks [28, 29, 34]. They utilized a mixture of microarchitecture-independent characteristics (i.e., instruction mix) as well as microarchitecture-dependent ones (i.e., cache misses per kilo-instruction) with the results from 5 different machines and 4 different ISAs. Their work was focused on the CPU domain, yet used similar statistical approaches as our study. Heirman et. al performed a bottleneck analysis on the multi-threaded *SPLASH-2*, *PARSEC* and *Rodinia* benchmark suites using cycle stacks, which show the breakdown of execution cycles [4, 14, 39]. Che and Skadron performed experiments on correlating and predicting the performance of GPGPU benchmarks using a hardware profiler [8]. In their study, key performance metrics such as warp occupancy and computation-to-memory access ratio were used in order to correlate selected benchmarks. Unlike our work, they used statistics to predict performance using correlation metrics. Carrington et al. performed HPC application workload characterization using the entire benchmark suites as synthetic metrics [5].

7. Acknowledgement

This work has been supported and partially funded by SRC under Task ID 2504 and National Science Foundation under grant numbers 1337393 and 1117895. Any opinions, findings, conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation or other sponsors.

8. Conclusion

In this study, we have developed a methodology that analyzes existing benchmark suites to understand what they cover in the performance spectrum and to provide guidance for future benchmarks. We have used statistical methods such as PCA and Hotelling's T^2 to identify unexplored areas in the performance spectrum and unique benchmarks among five popular public-domain GPGPU benchmark suites. When building a new benchmark suite or improving the existing benchmark suite, this paper provides a formalized approach for performance architects to identify areas to focus on for future benchmarking purposes. They can use the performance spectrum coverage map to find unexplored areas and investigate the sources of uniqueness in benchmarks that are located at the edges of this map using the Hotelling's T^2 and the dendrogram. In the end, our study has found that 16 of 88 benchmarks from five widely used suites cover 90% of the variance, which indicate that they overlap each other in many areas in the spectrum, leaving some areas to be explored. Our methodology can guide future benchmarks towards these unexplored areas, which can ultimately create benchmarks with much larger coverage map in performance spectrum.

References

[1] AMD, "OpenCL Programming Guide," 2013. http://www.amd.com
[2] K. Asanovic *et al.*, "A view of the parallel computing landscape," *Commun. ACM*, vol. 52, no. 10, pp. 56–67, Oct. 2009.
[3] A. Bakhoda, G. Yuan, W. Fung, H. Wong, and T. Aamodt, "Analyzing CUDA workloads using a detailed GPU simulator," in *Performance*

Analysis of Systems and Software, 2009. ISPASS 2009. IEEE International Symposium on, April 2009, pp. 163–174.

[4] C. Bienia, S. Kumar, J. P. Singh, and K. Li, "The PARSEC benchmark suite: Characterization and architectural implications," in *Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques (PACT)*. New York, NY, USA: ACM, 2008, pp. 72–81.

[5] L. C. Carrington, M. Laurenzano, A. Snavely, R. L. Campbell, and L. P. Davis, "How well can simple metrics represent the performance of HPC applications?" in *Proceedings of the 2005 ACM/IEEE Conference on Supercomputing (SC)*. Washington, DC, USA: IEEE Computer Society, 2005, pp. 48–.

[6] S. Che *et al.*, "Rodinia: A benchmark suite for heterogeneous computing," in *Workload Characterization, 2009. IISWC 2009. IEEE International Symposium on*, Oct 2009, pp. 44–54.

[7] S. Che, J. Sheaffer, M. Boyer, L. Szafaryn, L. Wang, and K. Skadron, "A characterization of the Rodinia benchmark suite with comparison to contemporary CMP workloads," in *Workload Characterization (IISWC), 2010 IEEE International Symposium on*, Dec 2010, pp. 1–11.

[8] S. Che and K. Skadron, "Benchfriend: Correlating the performance of GPU benchmarks," *IJHPCA*, vol. 28, no. 2, pp. 238–250, 2014.

[9] A. Danalis *et al.*, "The Scalable Heterogeneous Computing (SHOC) benchmark suite," in *Proceedings of the 3rd Workshop on General-Purpose Computation on Graphics Processing Units*. ACM, 2010, pp. 63–74.

[10] G. F. Diamos, A. R. Kerr, S. Yalamanchili, and N. Clark, "Ocelot: A dynamic optimization framework for bulk-synchronous applications in heterogeneous systems," in *Proceedings of the 19th International Conference on Parallel Architectures and Compilation Techniques (PACT)*. New York, NY, USA: ACM, 2010, pp. 353–364.

[11] G. H. Dunteman, *Principal Ccomponents Analysis*. Sage, 1989, no. 69.

[12] N. Goswami, R. Shankar, M. Joshi, and T. Li, "Exploring GPGPU workloads: Characterization methodology, analysis and microarchitecture evaluation implications," in *Proceedings of the IEEE International Symposium on Workload Characterization (IISWC'10)*. Washington, DC, USA: IEEE Computer Society, 2010, pp. 1–10.

[13] B. He, W. Fang, Q. Luo, N. K. Govindaraju, and T. Wang, "Mars: A MapReduce framework on graphics processors," in *Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques (PACT)*. New York, NY, USA: ACM, 2008, pp. 260–269.

[14] W. Heirman, T. E. Carlson, S. Che, K. Skadron, and L. Eeckhout, "Using cycle stacks to understand scaling bottlenecks in multi-threaded workloads," in *Proceedings of the 2011 IEEE International Symposium on Workload Characterization (IISWC)*. Washington, DC, USA: IEEE Computer Society, 2011, pp. 38–49.

[15] S. Hong and H. Kim, "An analytical model for a GPU architecture with memory-level and thread-level parallelism awareness," *SIGARCH Comput. Archit. News*, vol. 37, no. 3, pp. 152–163, Jun. 2009.

[16] S. Hong and H. Kim, "An integrated GPU power and performance model," *SIGARCH Comput. Archit. News*, vol. 38, no. 3, pp. 280–289, Jun. 2010.

[17] H. Hotelling, *The generalization of Student's ratio*. Springer, 1992.

[18] H. Jooybar, W. W. Fung, M. O'Connor, J. Devietti, and T. M. Aamodt, "GPUDet: A deterministic GPU architecture," *SIGPLAN Not.*, vol. 48, no. 4, pp. 1–12, Mar. 2013.

[19] A. Kerr, G. Diamos, and S. Yalamanchili, "Modeling GPU-CPU workloads and systems," in *Proceedings of the 3rd Workshop on General-Purpose Computation on Graphics Processing Units (GPGPU'10*.

[20] A. Kerr, G. Diamos, and S. Yalamanchili, "A characterization and analysis of PTX kernels," in *Proceedings of the 2009 IEEE International Symposium on Workload Characterization (IISWC)*. Washington, DC, USA: IEEE Computer Society, 2009, pp. 3–12.

[21] A. Likas, N. Vlassis, and J. J. Verbeek, "The global K-means clustering algorithm," *Pattern Recognition*, vol. 36, no. 2, pp. 451 – 461, 2003, biometrics.

[22] NVIDIA, "CUDA Code Samples." http://www.nvidia.com

[23] NVIDIA, "GeForce GTX 560 Ti." http://www.geforce.com

[24] NVIDIA, "NVDIA's Next Generation CUDA Compute Architecture: Fermi," 2009. http://www.nvidia.com

[25] NVIDIA, "NVIDIA CUDA C Programming Guide," 2014. http://www.nvidia.com

[26] NVIDIA, "Profiler User's Guide," 2014. http://www.nvidia.com

[27] D. A. Oliveira, C. B. Lunardi, L. L. Pilla, P. Rech, P. O. Navaux, and L. Carro, "Radiation sensitivity of high performance computing applications on kepler-based GPGPUs," in *Dependable Systems and Networks (DSN), 2014 44th Annual IEEE/IFIP International Conference on*, June 2014, pp. 732–737.

[28] A. Phansalkar, A. Joshi, L. Eeckhout, and L. John, "Measuring program similarity: Experiments with SPEC CPU benchmark suites," in *Performance Analysis of Systems and Software, 2005. ISPASS 2005. IEEE International Symposium on*, March 2005, pp. 10–20.

[29] A. Phansalkar, A. Joshi, and L. K. John, "Analysis of redundancy and application balance in the SPEC CPU2006 benchmark suite," *SIGARCH Comput. Archit. News*, vol. 35, no. 2, pp. 412–423, Jun. 2007.

[30] E. Phillips and M. Fatica, "Implementing the Himeno benchmark with CUDA on GPU clusters," in *Parallel Distributed Processing (IPDPS), 2010 IEEE International Symposium on*, April 2010, pp. 1–10.

[31] X. Ren, Y. Tang, G. Wang, T. Tang, and X. Fang, "Optimization and implementation of LBM benchmark on multithreaded GPU," in *Data Storage and Data Engineering (DSDE), 2010 International Conference on*, Feb 2010, pp. 116–122.

[32] S. Ryoo, C. I. Rodrigues, S. S. Baghsorkhi, S. S. Stone, D. B. Kirk, and W.-m. W. Hwu, "Optimization principles and application performance evaluation of a multithreaded GPU using CUDA," in *Proceedings of the 13th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP)*. New York, NY, USA: ACM, 2008, pp. 73–82.

[33] J. Sartori and R. Kumar, "Branch and data herding: Reducing control and memory divergence for error-tolerant GPU applications," in *Proceedings of the 21st International Conference on Parallel Architectures and Compilation Techniques (PACT)*. New York, NY, USA: ACM, 2012, pp. 427–428.

[34] SPEC, "SPEC CPU2000 and CPU2006." http://www.spec.org

[35] Standard Performance Evaluation Corporation, "SPEC CPU2006," 2006. http://www.spec.org/cpu2006.

[36] J. A. Stratton *et al.*, "Parboil: A revised benchmark suite for scientific and commercial throughput computing," *Center for Reliable and High-Performance Computing*, 2012.

[37] R. Taylor and X. Li, "A micro-benchmark suite for AMD GPUs," in *Parallel Processing Workshops (ICPPW), 2010 39th International Conference on*, Sept 2010, pp. 387–396.

[38] G. Wang, T. Tang, X. Fang, and X. Ren, "Program optimization of array-intensive SPEC2k benchmarks on multithreaded GPU using CUDA and Brook+," in *Parallel and Distributed Systems (ICPADS), 2009 15th International Conference on*, Dec 2009, pp. 292–299.

[39] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta, "The SPLASH-2 programs: Characterization and methodological considerations," in *Proceedings of the 22nd Annual International Symposium on Computer Architecture (ISCA)*. New York, NY, USA: ACM, 1995, pp. 24–36.