

BenchFriend: Correlating the Performance of GPU Benchmarks

Shuai Che[†] and Kevin Skadron[‡]

Shuai.Che@amd.com and skadron@cs.virginia.edu

AMD Research[†] and Computer Science, University of Virginia[‡]

Abstract—Graphics processing units (GPUs) have become an important platform for general-purpose computing, thanks to their high parallel throughput and high memory bandwidth. GPUs present significantly different architectures from CPUs and require specific mappings and optimizations to achieve high performance. This makes GPU workloads demonstrate application characteristics different from those of CPU workloads. It is critical for researchers to understand the first-order metrics that most influence GPU performance and scalability. Furthermore, methodologies and associated tools are needed to analyze and predict the performance of GPU applications and help guide users' purchasing decisions.

In this work, we study an approach of predicting the performance of GPU applications by correlating them to existing workloads. One tenet of benchmark design, also a motivation of this paper, is that users should be given capabilities of leveraging standard workloads to infer the performance of applications of their interest. We first identify a set of important GPU application characteristics and then use them to predict performance of an arbitrary application by determining its most similar proxy benchmarks. We demonstrate the prediction methodology and conduct predictions with benchmarks from different suites to achieve better workload coverage. The experimental results show that we are able to achieve satisfactory performance predictions, although errors are higher for outlier applications. Finally, we discuss several considerations for systematically constructing future benchmark suites.

I. INTRODUCTION

GPUs have become increasingly popular for general-purpose computing (GPGPU). GPUs offer a large number of SIMD cores and high memory bandwidth; they achieve significant speedups for many data-parallel applications. With GPGPU use increasing, research challenges include understanding GPU application behaviors, identifying first-order metrics that capture GPU performance, and designing methodologies for predicting performance. A better understanding of these issues is useful for analyzing and comparing different hardware platforms, and can guide users to choose the platforms that best serve their computation needs.

Previous research explored the issues of analyzing and predicting application performance for CPUs [5], [12], [16], [29]. However, they mostly concentrate on single-threaded applications. Recently, researchers have started to study performance prediction for GPUs. Some researchers build analytical models with detailed GPU hardware parameters as inputs [1], [14]. However, one challenge is keeping up with rapid GPU evolution, which invalidates highly tuned analytical models. Other works use regression methods to construct empirical

models [20]. However, machine-learning-based approaches often make it difficult for researchers to draw conclusions.

An alternative way of helping understand GPU performance behaviors is through benchmarking. Because users' applications may not exist in standard benchmark suites, one important goal of benchmark design is allowing users to predict their applications' performances based on the benchmark performance on different platforms. However, predicting performance of arbitrary applications using benchmarks remains an open problem, especially for new hardware. This is important, especially because—due to the finite benchmark mix offered by vendors—customers may need to purchase sample hardware, and port and characterize performance of applications before making large-scale purchases [12]. This is costly and time-consuming.

Our hypothesis is that, effective performance prediction is possible with a set of key metrics and a well-designed benchmark repository, without the need to purchase hardware, by sampling and interpolating the benchmark space, and thus taking advantage of data that vendors make available for standardized benchmarks. Our framework determines the most similar GPU benchmarks as proxies for an application of interest based on their mutual similarity in the workload space, using characterization on available hardware or a simulator. The predicted performance for the target platform is then determined by a linear interpolation of the performance of the proxy benchmarks on the target platform. Our work focuses on manycore architectures (e.g., GPUs), and uses a similar approach to the study by Hoste et al. [16] for single-threaded CPU applications. We also examine how well the applications included in today's GPU benchmark suites can represent the characteristics of real workloads, which we believe will facilitate a more scientific approach for future benchmark suite construction.

Our paper makes the following contributions:

- We identify a set of simple, first-order application characteristics for the GPU platform, and analyze their impacts on performance. We then demonstrate the entire flow of the performance prediction framework based on program similarity.
- We use the Rodinia benchmark suite and workloads from other benchmark suites for performance prediction.
- We evaluate the effectiveness of our prediction approach using different processor configurations, program inputs and numbers of nearest neighbors.
- We discuss important directions for future benchmark construction. We point out that future benchmark design should adopt a holistic approach to improve overall feature coverage. One metric to evaluate how well a suite

[†]Most of this work was completed while Shuai Che was with the University of Virginia

is designed can be its workload space coverage and how effectively it can be used for performance projection.

The framework can accurately predict GPU application performance except for applications that are isolated in the workload space. Our experiment result shows an arithmetic mean of 21.6% prediction error (14.6% excluding outliers) using simulation, and an arithmetic mean of 24% prediction error in predicting performance of an NVIDIA K20 based on characterization on a C2050 (Fermi). In addition, the predicted performance shows a strong correlation to actual performance based on the Spearman’s rank analysis. This suggests that our overall prediction is accurate, even with certain outliers whose absolute errors are somewhat high.

II. MOTIVATIONS AND BACKGROUND

A. Motivations

Users’ applications are their best benchmarks. However, because most of these applications are not included in standard benchmark suites, users sometimes need to predict the performance of their applications by referring to benchmark scores of standard benchmarks [16]. Therefore, researching mutual relationships among applications becomes important. One challenge is that it is almost impossible for users to run their applications on all the systems available in the market [12] due to accessibility and costs. Users sometimes have to rely on hardware specifications or white papers to estimate roughly the performance of a platform, which tends to be less accurate than well-designed performance prediction models. Furthermore, most users do not have the experience or skills to configure and change architectural simulators to model hardware of interest. Also, simulators are time-consuming and prone to their own inaccuracies.

These issues motivate the need to research methodologies that correlate the performance of a particular application with that of existing benchmarks to predict the performance of the target application accurately. Gustafson et al. [12] did an early study of performance correlations among benchmarks. Hoste et al. [16], [18] pioneered the research of benchmark suite coverage and used standardized benchmarks to conduct performance prediction for single-thread CPU workloads. Carrington et al. [5] predicted application performance using single, simple synthetic metrics (i.e., diverse compute kernels) and a linear combination of these simple metrics. For all of these works, one common requirement is first to build a benchmark repository covering diverse application behaviors. As far as we know, there is no previous work studying this issue for manycore architectures such as GPUs.

B. GPUs and Programming Models

GPUs differ from CPUs significantly in hardware architecture, programming models, and middleware support. For example, GPUs possess many simple, light-weight scalar cores focused on improving instruction throughput and hiding memory latency through deep multi-threading. In contrast, the heavy-weight out-of-order cores in CPUs target improving instruction-level parallelism with pipelining, speculation and

latency-hiding through cache hierarchies. This suggests that researchers are required to identify a set of effective metrics unique to GPUs and understand their implications on application performance.

GPU programming models (e.g., OpenCL [25] and CUDA [10]) represent the GPU as a co-processor capable of running a large number of threads. For example, threads are managed by representing parallel tasks as compute kernels mapped over a domain indexable by *work-groups IDs* (*block IDs* in CUDA) and *work-items* (*thread IDs* in CUDA). With CUDA terms, GPUs typically consist of multiple streaming multiprocessors (SMs), each with multiple streaming processors (SPs), executing instructions in a SIMT (Single Instruction Multiple Thread) fashion [21]. The threads in a thread block are time-sliced onto multiple SPs within a SM in groups of 32 called warps. Each warp of 32 threads operates in lockstep. Within a warp, divergent threads that follow different execution paths due to branches (e.g., *if* statements) are handled using hardware masking until they reconverge. Warp divergence will lead to an underutilization of GPU compute resources. Threads process data stored in the GPU device memory (*global memory*) concurrently; data is transferred from the main memory to the device memory via the PCIe bus in a system with a discrete GPU. Data can be loaded into shared, constant and texture memory spaces for improving data locality and optimizing different access patterns. In addition, efficient global memory accesses can be achieved when threads within a warp access contiguous memory locations; multiple accesses can be coalesced into fewer memory transactions (*memory coalescing*). Synchronization primitives are provided within each thread block and entirely managed in hardware. A simple GPU execution model is illustrated in Figure 1. Previous works [24], [37] showed that GPU performance is highly sensitive to effective memory bandwidth utilization and degree of branch divergence.

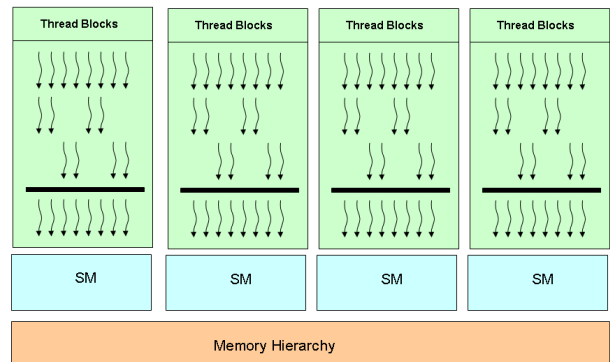


Fig. 1. Thread blocks (*work-groups* in OpenCL) are dispatched onto multiple SMs for execution. Due to branch divergence, only some threads within a warp may be active. Barrier synchronizations are supported for threads within a thread block. GPUs have per-thread-block shared-memory space (local space in OpenCL) and global constant and texture memory spaces.

C. Prediction for GPU Kernels

A classic debate for CPU benchmarks is whether today’s benchmarks represent the real workloads being used. This issue is well-documented in the literature [12] for CPU benchmarking. One major reason is that benchmark designers usually pick relatively smaller problems (e.g., small applications and small inputs) so the benchmarks can fit all kinds of machines and achieve general adoption. However, simple problems sometimes lose important characteristics of real applications [12]. This brings the challenge of predicting the performance of “big” applications with “small” ones; in other words, predicting the performance of real applications with kernels [5].

This is less an issue for benchmarking and predicting GPUs. Today’s GPU programming models like CUDA and OpenCL require programmers to map algorithms and data structures explicitly to their domain-based compute models. GPU applications themselves are a set of well-defined, small compute kernels accelerating compute-intensive loops. In addition, developers sometimes have to divide a large logical function into constituent smaller kernels due to the global synchronization requirement. Thus, for a big GPU application with multiple kernels, its application behavior can be predicted through analyzing and combining individual kernels. Furthermore, the GPU scaling behavior relative to input size is relatively easy to predict once the input is sufficiently large to fill the GPU; processing of the entire dataset will be divided into batches of thread blocks, distributed onto the GPU’s processing elements (i.e. a roughly linear and stepwise relationship between input size and execution time). These features make accurate performance prediction for GPU kernels possible and practical.

III. HIGH-LEVEL FRAMEWORK

In this section, we discuss the overall flow of performance prediction and the metrics we use to determine program similarity.

A. The Flow of Performance Prediction

Figure 2 illustrates our high-level prediction framework. This approach accords with the framework proposed by Hoste et al. [16] for CPU performance prediction. This paper differs from their work by targeting manycore architectures and identifying a set of first-order program metrics for GPU applications.

As shown in the diagram, performance prediction is achieved through correlating the characteristics of the application of interest with those of existing benchmarks in the workload space, whose performance scores are known before prediction. Program statistics are collected to construct a workload space and calculate pair-wise similarity values across different applications. Given an application of interest, we profile it with exactly the same metrics, map the application into the workload space and use the similarity information to determine its nearest proxy benchmarks from the benchmark repository [27]. Finally, the performance of the target application is predicted by interpolating the performance of proxy benchmarks. For example, many computer system vendors

report performance scores of their systems by instrumenting the SPEC benchmark suite [30]; these scores can be used for prediction and reference purposes for CPUs. The SPEC HPG group has been developing a GPGPU standard benchmark suite [33], and we anticipate that vendors may use it similarly to report benchmark scores for their GPU platforms.

Two issues are of particular importance to accurate prediction in our approach:

- How effectively can the chosen metrics capture the major behaviors of the GPU and represent similarity?
- How diverse are the benchmarks included in today’s benchmark suites and how well do they cover the workload space?

The first issue is important for accurately determining if two applications are similar, while the second issue is important to ensure the existence of appropriate proxy benchmarks to the application of interest. The rest of this paper endeavors to address these two problems.

B. Application Profiling and Metrics

To determine the similarity among different applications, we use a set of metrics that play a major role in determining GPU performance. Our choice of metrics is based on observations of previous experiences of benchmarking GPUs [2], [9], [19]. These metrics have proved capable of effectively capturing the program behaviors of both NVIDIA and AMD GPUs. However, this does not preclude other metrics, that could make the performance prediction more accurate. Furthermore, a comprehensive and rigorous characterization of the most suitable metrics would require statistical analysis (e.g., principal component analysis) and/or genetic algorithms [16], [18] to select from a large pool of potential metrics. The following list illustrates the metrics used in this study. They represent degrees of compute intensity, memory locality, branch divergence, etc.

- **Instruction throughput** This metric demonstrates the aggregate throughput of an application. We use instruction per cycle (IPC) in this work. Different applications can achieve a similar level of IPC but quite different scaling behaviors, which suggests additional metrics are needed to present other aspects of application behaviors.
- **Computation-to-memory access ratio** A higher ratio implies that an application stresses more arithmetic units, while a low ratio suggests more stresses on memory interface. This metric is a widely used to determine if an application is compute-bound or memory-bound.
- **Memory instruction mix** A GPU application may take advantage of multiple memory spaces on the GPU. Significant global memory accesses lead to poor performance and scalability. A high ratio of memory accesses to GPU caches (scratchpad, constant, and texture) generally means a better usage of data locality, fewer off-chip accesses and better program scalability.
- **Memory efficiency** One important performance optimization for GPUs is the coalescing of global memory accesses [10]. A high ratio of uncoalesced memory ac-

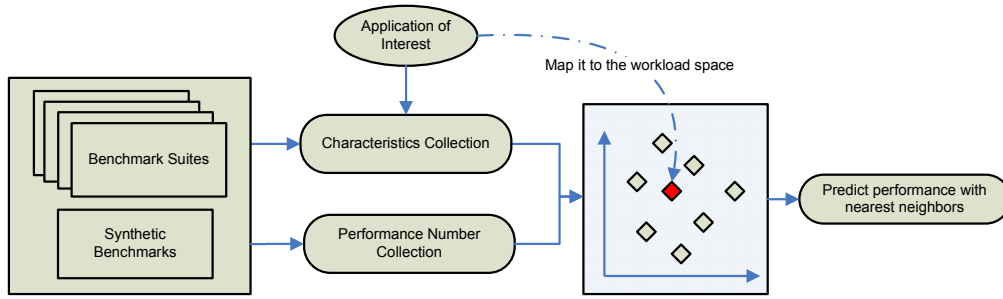


Fig. 2. First, benchmark statistics are collected to construct a workload space. An application of interest is profiled with exactly the same set of metrics and mapped into the workload space. Prediction is achieved with the nearest proxy benchmarks to the application of interest [16].

cesses suggests a waste of effective memory bandwidth and thus degrades the overall application performance.

- **Warp occupancy** Warp occupancy captures the average number of active threads over all issued warps over the entire runtime of the benchmarks [2]. We classify this metric into four buckets, namely [1–8], [9–16], [17–24] and [25–32]. A higher warp occupancy means a better utilization of GPU computation resources.

IV. METHODOLOGY

In this section, we discuss our methodology, including the experiment environment, the workloads used in this paper and how we calculate similarity among benchmarks and further use it for performance prediction.

A. Experiment Setup

To measure program characteristics, we use GPGPU-Sim [2] from the University of British Columbia. GPGPU-Sim provides a detailed simulation model of contemporary GPUs. We use GPGPU-Sim 2.1.1b to report program statistics such as IPC, instruction mix, warp occupancy, uncoalesced memory accesses, etc., which will be used for subsequent performance predictions. We also use GPGPU-Sim to simulate the performance of GPU systems with different numbers of SMs and memory channels. For hardware prediction, we use an NVIDIA Tesla C2050 and a Kepler K20 GPU. The program characteristics are collected with NVIDIA’s CUDA (5.0) profiler *nvprof*.

The program characteristics needed to be collected once, and the constructed benchmark space can be re-used for future performance prediction. A GPU application usually consists of CPU parts, CPU-GPU PCIe transfers and GPU kernels. We restrict our performance prediction work to GPU kernels and leave CPU execution time as a constant variable. In addition, time spent on PCIe transfers is proportional to the data size transferred across the CPU and the GPU. Therefore, the transfer time can be predicted easily by dividing data size with the PCIe bandwidth. In addition, we do not yet consider applications in which computations and PCIe transfers are overlapped.

B. Workloads

Accurate performance prediction requires the workload repository to include a sufficient number of benchmarks to

TABLE I
APPLICATIONS.

Application	Abbrev	Suite	Input Size
Back Propagation	BP	Rodinia	65,536 input nodes
CFD Solver	CFD	Rodinia	97,000 elements
Heart Wall Tracking	HW	Rodinia	609×590 pixels/frame
HotSpot	HS	Rodinia	500×500 grid
LU Decomposition	LUD	Rodinia	256×256 matrix
Needleman-Wunsch	NW	Rodinia	2,048×2,048 data points
SRAD	SRAD	Rodinia	512×512 data points
Stream Cluster	SC	Rodinia	2048×2048 data points
LIBOR Monte Carlo	LIB	GPGPU-Sim	4,096 paths, 15 options
Neural Network	NN	GPGPU-Sim	28 digits
NQueen Solver	NQU	GPGPU-Sim	10×10 grid
Ray Tracer	RAY	GPGPU-Sim	256×256 image
Weather Prediction	WP	GPGPU-Sim	10 time-steps
BlackScholes	BLK	NVIDIA SDK	4 million options
DXTC	DXTC	NVIDIA SDK	512×512 image
Matrix Multiply	MM	NVIDIA SDK	80×48, 48×128
Fast Walsh Transform	FWT	NVIDIA SDK	32,000 data points
MersenneTwister	MT	NVIDIA SDK	24,000,000 samples

construct a training set. These benchmarks also need to be diverse in application characteristics and ideally distributed evenly in the workload space. We use real workloads with diverse characteristics for training. For this study, we first evaluate the effectiveness of using only Rodinia [6] for performance prediction, and then we will take other workloads into account, which provides a richer workload space.

The input sizes we choose for all the benchmarks can make full use of the GPU. We also restrict our study to applications that do not take advantage of a GPU’s texture memory space. CUDA and OpenCL allows programmers to bind big read-only data structures to texture memory, which is cached and optimized for arbitrary memory access patterns. Texture units present a unique access pattern, that we leave for future work.

C. Similarity and Proxy Benchmarks

We define similarity between two benchmarks as their mutual Euclidean distance in the n-dimensional workload space. Each benchmark is represented by a data vector with multiple dimensions; each dimension of the workload space represents one characteristic. We calculate pair-wise distances for all the benchmarks. For each benchmark, we search the entire workload space to find k ($k \geq 1$) closest benchmarks for the interpolation purpose. We use the k -Nearest Neighbors algorithm for searching [27]. K benchmarks will be used as the proxy benchmarks for the application of interest. The values of different metrics vary in the range. Therefore,

when calculating the Euclidean distances among benchmarks, normalization is applied first to raw data for all characteristics across all benchmarks.

The choice of the value k is also important to prediction accuracy, which we will discuss in details in section V-D. We use MATLAB [34] to process data for collected characteristics, calculate similarities among benchmarks (i.e. $pdist$) and search nearest neighbors for individual benchmarks.

D. Scaling Prediction

$$Weight(i) = \frac{1}{\sum_{i=1}^n \frac{1}{dist(i)}} \quad (1)$$

$$SpeedupPred = \sum_{i=1}^n Weight(i) * Speedup_i \quad (2)$$

$$Error = \frac{|SpeedupPred - SpeedupReal|}{SpeedupReal} * 100\% \quad (3)$$

After we determine the most similar benchmarks to an application of interest, we predict its relative performance: the speedup of running an application on one platform against another platform. The predicted speedup is calculated with Equations (1) and (2). In each equation, n represents the number of proxy applications. The predicted speedup of the application of interest is represented by a linear combination of its nearest neighbors, each contributing a component to the overall predicted value. The weights of the neighboring applications are assigned to be inversely proportional to the distances to the application of interest [16]. This means that the more similar a benchmark is to the application of interest, the more weight it should be given for prediction. If we can obtain the run-times on one platform, we can also predict the run-times on another platform by multiplying the speedups. To evaluate how accurate the prediction is, we compare the predicted speedup with the real speedup achieved. The absolute prediction error can be calculated with Equation (3). In addition, we choose the arithmetic mean of absolute prediction errors because we are reporting results that are samples from the overall population of application behaviors.

V. PERFORMANCE PREDICTION AND RESULTS

In this section, we report performance prediction results with different numbers of benchmarks and across multiple processor configurations and multiple application inputs. We also study the impact of the number of proxy benchmarks and verify the statistical significance of prediction accuracy.

A. Performance Prediction with Existing GPGPU Benchmarks

In real-world practice, users may rely on standard benchmarks with widely-accessible information and performance scores. There are several benchmark suites released for GPGPU [2], [6], [8], [32]. However, how representatively they can be used as references has not been sufficiently understood by prior works.

In the first experiment, we consider eight benchmarks from the Rodinia benchmark suite. To predict the performance of a particular application, the rest of applications are deemed as the training set. We use a “leave-one-out” approach for each application [16], [27], and calculate its pair-wise distances with the rest of Rodinia applications. We subsequently calculate the nearest neighbors of the target application and predict that application’s scaling behavior with determined proxies. Figure 3 shows the predicted and actual speedups for eight Rodinia applications switching from an 8-SM to 28-SM configuration. For each application, the predicted speedup is calculated by taking a weighted sum of the speedups of its proxy applications. The prediction result for this experiment tends to be inaccurate. This is attributed to the small application space covered by these benchmarks both in number and feature. In addition, the experiment result suggests that to evaluate the prediction framework, we should consider more benchmarks enriching the application coverage.

In the second experiment, we consider some other applications from the GPGPU-Sim suite and NVIDIA’s CUDA SDK. We report the results for the benchmarks we are able to simulate successfully and exclude those making use of the texture memory. We also did not consider simple kernels such as vector add and matrix transpose from the CUDA SDK. Figure 4 shows the new prediction results comparing the predicted speedups to the actual speedups. Including new applications improved the prediction accuracy except for some outlier applications. The results show an average 21.9% absolute errors. In particular, *NQU* (50.3%) and *LUD* (133.3%) are poorly predicted. These two applications tend to have low warp occupancies and limited parallelism [2], [6]. On the other hand, as proxy benchmarks they contribute to the accurate predictions of other benchmarks (e.g., *NN* and *NW*). Excluding *NQU* and *LUD*, the average absolute error of the rest applications is 14.6%, which is close to that of the GPU analytical model [14] constructed with a set of detailed GPU parameters (13% error). This suggests that our approach has a potential for accurate performance prediction if the benchmark repository is well constructed, covering most of the application space.

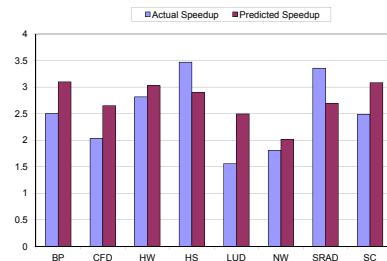


Fig. 3. The predicted speedups (28 SMs vs. 8 SMs) and measured speedups. We use only Rodinia benchmarks in this experiment.

To illustrate better similarity/dissimilarity among benchmarks, we conduct a principal component analysis (PCA) across all benchmarks for all characteristics. PCA transforms a

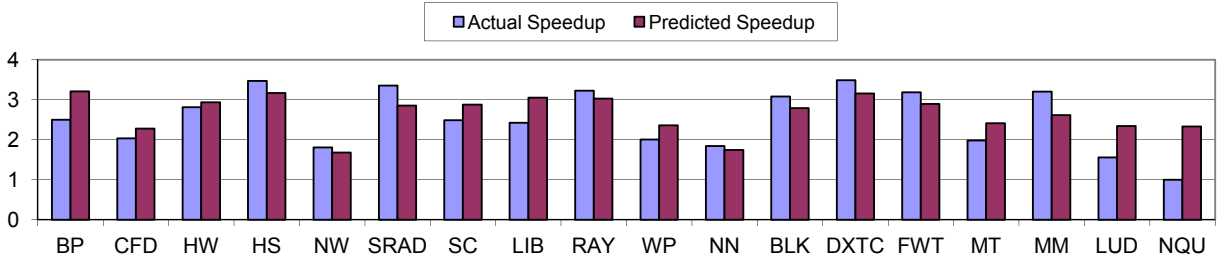


Fig. 4. The predicted speedups (28 SMs vs. 8 SMs) v.s. measured speedups for all the applications. Prediction is conducted with real workloads from Rodinia, GPGPU-Sim and NVIDIA CUDA SDK.

number of possibly correlated variables (or dimensions) into a set of uncorrelated variables, called principal components [18]. PCA has the ability to describe a big data set along a limited number of dimensions while still capturing the essence of the entire data set. For illustration, we plot all the benchmarks in the 3-D PCA space (the max dimension we can show) in Figure 5. However, three PCs already represent 78% of total variance, which accounts for most differences among benchmarks. It is interesting that some outlier benchmarks (e.g., *NN* and *NW*) in the workload space are not badly predicted. Similar to their scaling behaviors, the nearest neighbors determined by the framework (see Table III) also exhibit relatively poor scaling, making prediction relatively accurate. From our experiment, the worst-predicted benchmarks (*LUD* and *NQU*), though also belonging to the category of outlier benchmarks, are located at the “transition” region in the workload space; the nearest neighbors to *LUD* and *NQU* both include benchmarks that scale well (i.e., over-prediction).

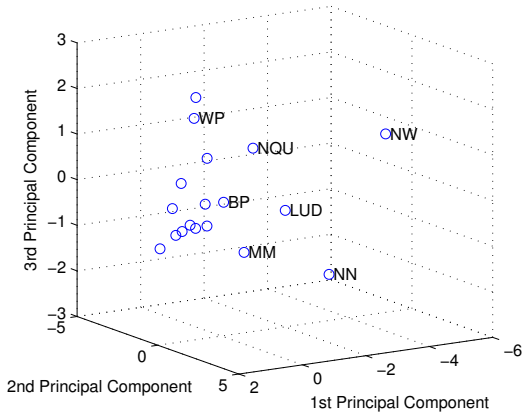


Fig. 5. Applications in the 3-D PCA space. The big contributors to each principal component are: warp occupancy [24-32] and IPC for the first PC, cached memory accesses and memory efficiency for the second PC, and computation/memory ratio for the third PC.

B. Spearman’s Rank

Using the metric of absolute error alone is not sufficient to justify accurate prediction; the evaluation of correlation between predicted and real values is needed. We use Spearman’s rank correlation, a non-parametric statistical measure of mutual relationship and dependence between two variables. Furthermore, it does not require that samples meet specific

distribution requirements (e.g., normal distribution in Pearson correlation coefficient). This statistical method was also used by the work [16] to evaluate the accuracy of performance prediction. The *rank* of a given value is its position in the ascending order of the sample values. For instance, for a set of four values $\{0.3, 0.1, 0.5, 0.2\}$, their corresponding ranks are $\{3, 1, 4, 2\}$. We convert the actual speedup value A_i and predicted speedup value P_i into ranks a_i and p_i . ρ is calculated with the Equation $1 - \frac{6 \sum d_i^2}{n(n^2-1)}$, where $d_i = a_i - p_i$ and n represents the sample size. Figure 6 illustrates 2-D views with the x-axis representing the predicted speedup while the y-axis represents the actual speedup. The predicted speedups show a strong correlation with the actual speedups across all applications. Ideally, all the points should be on the $y = x$ line (i.e. perfect prediction). For instance, to evaluate the significance of prediction for the experiment shown in Figure 4, we rank all the predicted and actual speedups and calculate Spearman’s rank correlation coefficient, which results in a high ρ value of 0.738. Then we determine the “critical value”. We use the common significance level of $\alpha = .05$. The corresponding significance value is 0.429 (0.582 when $\alpha = .01$). Our calculated correlation efficient, 0.738, is much higher than this value, which means that our approach is reasonably accurate.

C. Prediction with Different GPU Configurations

We also predict the scaling of applications under different GPU configurations. Table II shows the prediction results for the speedups of a 28-SM GPU (8 memory channels) against GPUs with 4, 8 and 16 SMs. These three configurations are associated with 4, 6 and 8 memory channels respectively. When we refer to X vs. Y (e.g., 28 vs. 8) in this paper, it means we compare the predicted speedup with the actual speedup of an application, going from a Y-SM (e.g., 8-SM) to X-SM (e.g., 28-SM) GPU configuration. The arithmetic mean of the absolute prediction errors ranges from 15.8% to 27.3% and the ρ values range from 0.620 to 0.774. All three cases show similar patterns to Figure 4, suggesting that the proposed approach is capable of predicting performance accurately except for outlier applications. The 28 vs. 4 SM case (27%) shows a relatively higher average prediction error but a higher ρ . The 28 vs. 16 SM case shows a relatively lower average absolute error as well as a lower ρ . Interestingly, ρ decreases as the prediction error decreases. This sounds contradictory at first glance, but is due mostly to outlier applications. For instance, for *NQU*, a highly serialized application, its performance is insensitive to the changing number of SMs; in the 28 SM vs.

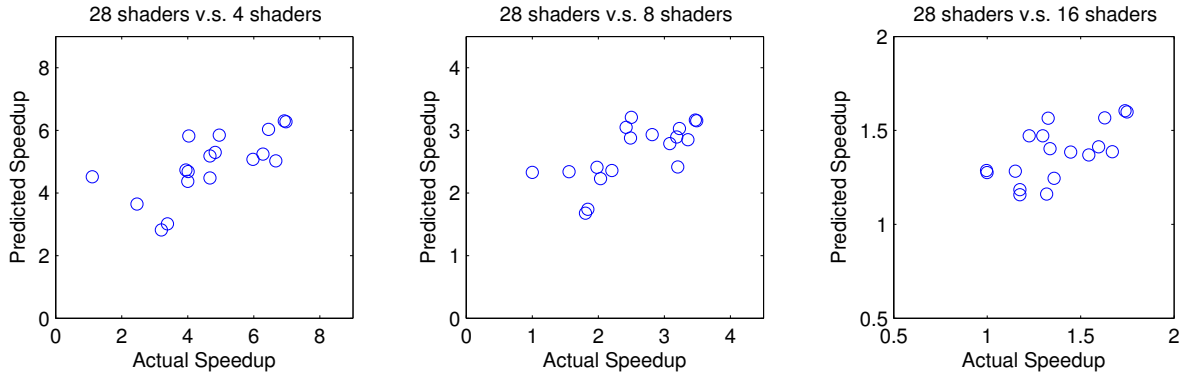


Fig. 6. Actual speedups (x-axis) and predicted speedups (y-axis). Each dot in the graph represents one benchmark.

4 SM case, its actual speedup is approximately 1.1 while the predicted speedup (calculated by its neighboring applications that benefit from parallelism) is 4.5, which contributes a significant portion to the average prediction error. On the other hand, a high degree of correlation is observed for the overall set of applications. Similarly, in the 28 SM vs. 16 SM case, the error contribution of the outlier benchmarks becomes smaller because the individual predicted value does not exceed 1.75 (i.e. 28/16), which explains a smaller prediction error. Figure 6 shows the prediction results for three GPU configurations. Each dot in the graph represents one benchmark and most benchmarks show up along the $y = x$ line.

GPUs	4 SM-4 MEM	8 SM-6 MEM	16 SM-8 MEM
Avg. Errors	27.3%	21.9%	15.8%
Significance: ρ	0.774	0.738	0.620

TABLE II
PREDICTION ERRORS AND SPEARMAN’S RANK RESULTS ACROSS DIFFERENT GPU CONFIGURATIONS. X SM-Y MEM MEANS THE SIMULATOR IS CONFIGURED TO USE X STREAMING MULTIPROCESSORS AND Y MEMORY CHANNELS.

D. Number of Proxies

The number of proxies is an important parameter critical to accurate prediction. We compute the speedup for the application of interest using a number of proxies. The actual proxies, as discussed in Section IV-C, are determined by the k-Nearest Neighbors algorithm. It is not feasible for the sample space to include all the points (i.e., essentially all the theoretical applications); therefore, our assumption of the framework is that we will need more than one nearest neighbor ($k > 1$) for interpolation purpose. On the other hand, if k is too large, it will adversely degrade the accuracy of prediction; using a large k risks that some applications dissimilar to the application of interest may be treated as proxies.

Figure 7 quantifies the sum prediction error as a function of number of proxies. The prediction error improves when increasing the number of proxies and then starts to degrade with larger numbers, which proves our hypothesis. The best prediction happens when we use three proxies in our experiments: the prediction results are thus reported with three proxy benchmarks in this paper. If switching to a different

benchmark repository, an appropriate proxy number needs to be re-determined. Additionally, deciding how many proxy benchmarks are needed as a function of their closeness to the target application and their distribution in the workload space is an open research question.

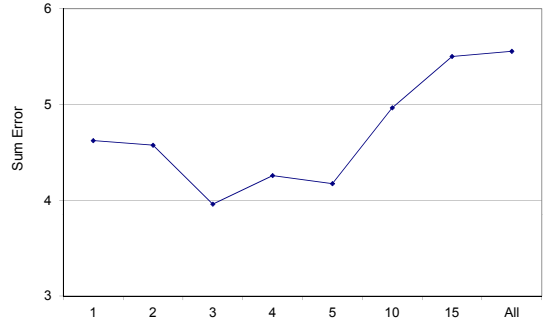


Fig. 7. The changes of prediction errors while increasing the number of nearest neighbors (i.e. proxies).

E. Different Input Sizes

In this section, we predict performance using different application input sizes. The GPU schedules thread execution by dispatching thread blocks on multiple SMs. As discussed in Section II-C, when the number of thread blocks are sufficiently large to fill the GPU, further increasing the input size will not influence the scaling behavior much (i.e. there is an approximately linear relationship between run-time and input). We show two applications, *SRAD* and *Needleman-Wunsch*, as examples and vary their input sizes from 512×512 to $2,048 \times 2,048$ and from $1,024 \times 1,024$ to $2,048 \times 2,048$ respectively. We measure the execution time in cycles on a 8-shader GPU and calculate the execution time of different inputs on a 28-shader GPU with the predicted speedup. Figure 8 shows the results comparing the actual cycles versus the predicted cycles. *SRAD* achieves an average 19.3% absolute prediction error while *Needleman-Wunsch* achieves an average 11.6% error. Of course, this is restricted to only those applications whose behaviors are not data-dependent.

F. Prediction on Real Hardware

We verify the robustness of our prediction approach by predicting performance for a “future” GPU. We predict ap-

TABLE III
THE THREE PROXIES AND THEIR WEIGHTS FOR EACH BENCHMARK

	First proxy		Second proxy		Third proxy	
	Benchmark	Weight	Benchmark	Weight	Benchmark	Weight
Needleman-Wunsch	LU Decomposition	0.3423	N-Queen Solver	0.3330	Back Propagation	0.3247
HotSpot	DXTC	0.4522	Back Propagation	0.2925	SRAD	0.2553
Back Propagation	SRAD	0.3755	Heartwall	0.3417	DXTC	0.2828
SRAD	Heartwall	0.4004	Fast Walsh Transform	0.3572	Monte Carlo	0.2424
Streamcluster	Heartwall	0.3477	SRAD	0.3429	Monte Carlo	0.3094
CFD Solver	MersenneTwister	0.4733	Monte Carlo	0.2735	Streamcluster	0.2532
LU Decomposition	Matrix Multiply	0.3916	Streamcluster	0.3208	N-Queen Solver	0.2876
Heartwall	Monte Carlo	0.4023	SRAD	0.3317	Fast Walsh Transform	0.2660
Monte Carlo	Heartwall	0.4769	Fast Walsh Transform	0.2849	SRAD	0.2382
Neuro Network	N-Queen Solver	0.3549	Needleman-Wunsch	0.3265	Back Propagation	0.3186
Ray Tracer	HotSpot	0.3921	CFD Solver	0.3096	DXTC	0.2983
Weather Prediction	CFD	0.3682	MersenneTwister	0.3440	Ray Tracer	0.2878
BlackScholes	Monte Carlo	0.3757	Fast Walsh Transform	0.3326	Heartwall	0.2917
DXTC	HotSpot	0.3794	SRAD	0.3362	Back Propagation	0.2844
N-Queen Solver	CFD Solver	0.3574	Streamcluster	0.3311	Back Propagation	0.3115
Fast Walsh Transform	SRAD	0.3689	Heartwall	0.3315	Monte Carlo	0.2996
MersenneTwister	CFD Solver	0.4790	BlackScholes	0.2661	Monte Carlo	0.2549
Matrix Multiply	LU Decomposition	0.3915	Streamcluster	0.3147	DXTC	0.2938

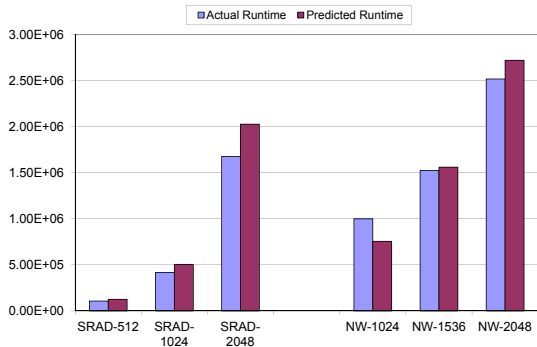


Fig. 8. The actual and predicted execution times (in cycles) of *SRAD* and *Needleman-Wunsch* with different input sizes.

plication speedups upgrading from an NVIDIA Tesla C2050 to a Kepler K20 GPU and compare the results to the actual speedups achieved for all the applications. When identifying program similarity, we collect application characteristics on the Tesla C2050 using hardware counters with NVIDIA’s CUDA profiler *nvprof* and use counter values to calculate pairwise distances. The metrics we evaluated include instructions per cycle (IPC), percentage of divergent branches to total branches, percentage of memory instructions to total instructions, percentage of cached instructions (e.g. shared memory), L2 cache miss rate and warp execution efficiency [11]. The profiler supports only per-kernel profiling, so for each application we report numbers for each individual kernel. Some applications in the simulator study are not reported here due to that (e.g., *MT* and *WP* failed to successfully execute and compile respectively on our system). Figure 9 shows the predicted speedups and actual speedups across all application kernels. Our prediction approach can achieve an arithmetic mean of 24% error. The Spearman’s ranking coefficient is 0.4308, higher than the critical value of 0.353 ($N = 26$, $\alpha = .05$), which suggests that our approach is reasonably accurate and the results are statistically significant.

G. Discussion

Our analysis shows that one challenge of this approach to performance prediction is the difficulty of predicting an application of interest that is isolated in the workload space (discussed in Section V-A). The same observation has been made by Hoste et al. [16]. For these applications, their proxy benchmarks determined by the framework are relatively farther away from the application of interest than those applications in “richer” areas of workload space, which means the benchmarks might not accurately represent the application behavior of the target application. We have shown that predictions can be poor with inappropriately determined nearest neighbors. Therefore, constructing a comprehensive benchmark repository for training is critical to accurate prediction.

Based our analysis, the workload space in this study can be enriched further by adding applications with several features. These include applications with diverse warp occupancies, especially applications with low SIMD utilizations (e.g., *NW*), limited parallelism (e.g., *NN* and *NQU*) and significant off-chip memory accesses (e.g., *WP* and *NN*). These regions of application space are underrepresented, with only a few applications. Additional benchmarks are needed because accurate prediction for a given application requires sufficient number of benchmarks with similar characteristics as nearby proxies.

This raises important questions of benchmark suite design in general: What programs we should select for inclusion in a standard benchmark suite [16] and how many benchmarks are sufficient in terms of both feature coverage and cost? Several prior works have examined similar issues for today’s research workloads. Phansalkar et al. [26] make the argument that the SPEC CPU benchmark suite only covers a subset of application behaviors and its workloads exhibit redundant behaviors. Hoste et al. [15] use microarchitecture-independent characteristics and PCA to characterize single-threaded workloads. Heirman et al. [13] use a cycle-stack approach to compare multithreaded CPU workloads including SPLASH2 [36], PARSEC [4] and Rodinia. Goswami et al. [9] and Che et al. [7] analyze GPU workloads and their coverage. Their findings agree with our observations: a thorough exam-

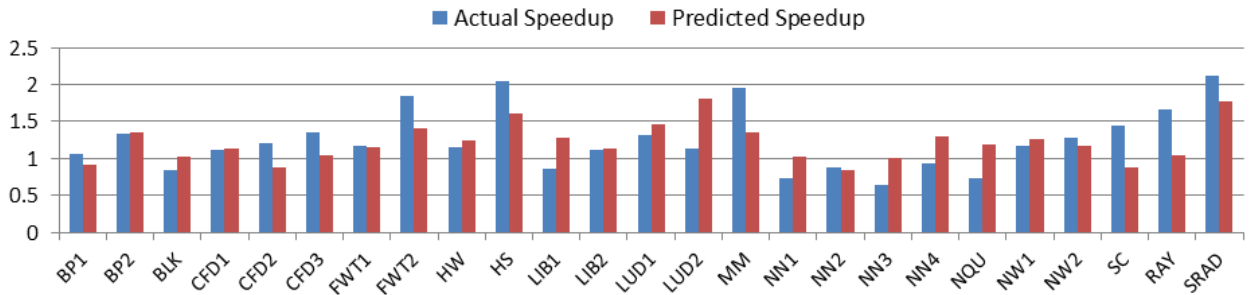


Fig. 9. Actual and predicted speedups on real hardware (a Kepler K20 GPU vs. an NVIDIA Tesla C2050)

ination for benchmark construction requires a comprehensive evaluation and comparison of all the current benchmark suites to establish a single set of workloads with sufficient coverage and little redundancy.

Phansalkar et al. [26] propose using hierarchical clustering to guide benchmark construction (also used in prior work [7], [9], [13]). With the help of the clustering tree (e.g., dendrogram), users can choose appropriate benchmarks to meet their needs (e.g., benchmark construction) - selecting the N most diverse benchmarks for any N . One benchmark can be chosen from each cluster by tracking down the clustering tree. If a cluster consists of more than two benchmarks, the benchmark closest to the center of the cluster is chosen as a representative.

Our prediction framework also can be used as a method to test how well the benchmark suite is constructed, in terms of covering diverse workload behaviors, by following the prediction process discussed in this paper and determining how well outliers are predicted. The size and workload coverage of a suite can be modified to make tradeoff between prediction accuracy and instrumentation cost. Benchmark vendors (e.g., SPEC) may consider adopting similar approaches to ensure the diversity of their provided benchmark suites meets different needs, including performance prediction. In addition to reporting metrics such as SPECratio, SPECrate, power, etc., it also would be helpful to report program characteristics so the benchmark space can be constructed easily. These save users from needing to determine a reasonable benchmark set independently as well as the effort of application profiling. Another interesting open research question is whether some real-world applications exist to cover those “desert” areas of application space. This issue might partially be resolved by developing synthetic benchmarks to mimic diverse application behaviors.

In this work, we assign weights to proxy benchmarks merely based on their distances to the target benchmark (i.e., the shorter the distance, the larger the weight; see Section IV-D). However, it is possible that different metrics may contribute different weights to the overall predicted value, which may potentially make the prediction more accurate. In this work, we assume all the metrics have equal weights. We leave the understanding of the contribution of each metric as future work.

VI. OTHER POTENTIAL METRICS

Our work has shown promising trends of predicting GPUs’ application performance with existing benchmarks. We hope to address some limitations in future work. In this paper, we consider only a few first-order characteristics that affect GPU performance. Other factors might make performance prediction more accurate. We plan to study metrics capturing memory access efficiency of other GPU caches (e.g., texture and Fermi’s hardware caches). We also plan to consider synchronization overhead within each thread block (e.g., `syncthread()`) and across thread blocks (i.e., global synchronization). On the other hand, in certain cases, some of these factors might not influence prediction results significantly (e.g. first-order metrics). For instance, when `syncthread()`s are used at the points of exchanging data between global memory and shared memory, interleaved warp executions may hide most synchronization overheads. Global synchronization overhead is caused by different finishing times of thread blocks dispatched to multiple SMs. If there are a sufficiently large number of thread blocks evenly distributed to SMs, synchronization overhead is negligible – the execution time of one or a few thread blocks. Also, part of the program characteristics can be directly collected through GPU hardware counters; we leave instrumentation and analysis of GPU hardware for future work.

VII. RELATED WORK

Gustafson and Todi [12] performed an early work in correlating the performance of benchmarks with others. Hoste et al. conducted a benchmark suite coverage study [18] and they used standardized benchmarks to conduct performance prediction [16]. The framework they proposed for prediction is based on principal component analysis and genetic algorithms. Snively et al. [29] studied an approach to predict parallel application performance on HPC systems. They collected machine profiles and application signatures, and combined them for prediction with a convolution method. Carrington et al. [5] predicted application performance using single, simple synthetic metric (i.e. compute kernel) and a linear combination of these simple metrics as well. The individual “metrics” are small synthetic benchmarks such as LINPACK [22], STREAM [31] and HPC Challenge benchmarks [3], etc. All of these prior works mainly focused on CPU workloads.

Hong et al. [14] proposed an analytical model that estimates the execution time of massively parallel programs on GPUs.

Similarly, Bagsorkhi et al. [1] proposed a performance model capturing performance effects of major GPU microarchitecture features using an approach based on the program dependence graph (PDG). Meng et al. [23] proposed GROPHECY, a GPU performance projection framework that can estimate the performance benefit of GPU acceleration without actual GPU programming or hardware. Users only need to skeletonize pieces of CPU code as targets for GPU acceleration, which are transformed further in various ways to tune GPU code in the optimization space. The code characteristics are used by an analytical model to project GPU performance. Kerr et al. evaluated a set of metrics for GPU workloads [19] and used them to analyze the behavior of GPU programs. They further used PCA and regression modeling to predict GPU performance [20]. Another regression-based approach was proposed by Jia et al. [17] to build application-specific models for performance prediction and identification of significant architectural parameters. In contrast to these works based on analytical and regression models for GPUs, our approach predicts GPUs' application performance by taking advantage of the characteristics and performance numbers of existing benchmarks.

Wong et al. [35] developed a set of microbenchmarks to detect various microarchitectural parameters for NVIDIA GPUs. Zhang et al. [38] developed a microbenchmark-based performance model that allows programmers and architects to identify GPU program bottlenecks and predict the benefits of potential program optimizations and architectural improvements. Our work focuses on real GPU applications instead of microbenchmarks.

Piccart et al. [28] proposed another approach for CPU performance prediction. They exploited machine similarity and used a data transposition technique to identify a predictive machine that is most similar to the target machine of interest for predicting the performance of an application of interest. It assumed that benchmark results for a sufficient number of machines are available. Different from their work, this paper uses benchmarks as a training set and determines the benchmarks that are most similar to the target benchmark for performance prediction.

VIII. CONCLUSION

In this paper, we study an approach of using existing benchmarks to predict performance of arbitrary GPU applications. This is helpful when performance of these benchmarks on the target GPU is available, but user access to the GPU is not possible, such as when making purchasing decisions. Given a target application, prediction is conducted by collecting a set of important GPU characteristics for all the benchmarks in the repository, identifying the proxy benchmarks in the workload space that are most similar to the GPU application to be predicted, and using the performance of the proxy benchmarks to predict that of the target application. We predict performance speedups of various applications across different GPU configurations. The predicted value for a particular benchmark is determined by a weighted sum of the speedups of its proxy benchmarks. We allocate the contribution of each benchmark

to be inversely proportional to its distance to the target application. We consider real workloads from Rodinia, GPGPU-Sim and NVIDIA CUDA SDK to construct a diverse and representative workload space. The results show that accurate performance prediction is possible with the proposed metrics and the methodology based on nearest neighbors. We are able to achieve an arithmetic mean of 21.9% prediction error using simulation and an arithmetic mean of 24% prediction error on real GPU hardware. Much of the error is due to a few outlier applications in the workload space. This problem can be reduced by improving the baseline set of benchmarks to cover these outlying areas more effectively. In addition, the predicted performance shows a strong correlation with the actual performance according to a Spearman's rank analysis.

Users can take advantage of performance scores of a variety of standard GPU benchmarks provided by GPU vendors (e.g. CPU vendors report performance scores for standardized CPU benchmarks such as SPEC [30]). Achieving this goal requires a well-constructed GPU benchmark suite with sufficient diversity and feature coverage. Furthermore, studying mutual relationships among benchmarks allows users to focus on understanding and analyzing the most important and relevant proxy benchmarks, which is useful in helping them make appropriate design and purchasing decisions.

Future work will study the robustness of this prediction approach with different programming styles and the impact of hardware ISA and architecture changes (e.g., across AMD and NVIDIA GPUs, different core configurations, and memory sub-systems). We plan to compare our approach to analytical models based on detailed GPU parameters. We also will study the application space coverage with benchmarks from a much larger set of workloads drawn from real applications. Using a benchmark correlation-based approach to predict power remains an interesting research question.

IX. ACKNOWLEDGEMENTS

This work was supported in part by NSF grants CNS-0916908 and CCF-1116673, and by a grant from AMD Research. We thank the reviewers for their constructive comments.

REFERENCES

- [1] S. S. Bagsorkhi, M. Delahaye, S. J. Patel, W. D. Gropp, and Wenmei W. Hwu. An adaptive performance modeling tool for GPU architectures. In *Proceedings of the 15th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, Jan 2010.
- [2] A. Bakhoda, G. L. Yuan, W. L. Fung, H. Wong, and T. M. Aamodt. Analyzing CUDA workloads using a detailed GPU simulator. In *Proceedings of 2009 IEEE International Symposium on Performance Analysis of Systems and Software*, April 2009.
- [3] HPC Challenge Benchmarks. Web resource. <http://icl.cs.utk.edu/hpcc/>.
- [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*, Oct 2008.
- [5] L. Carrington, M. Laurenzano, A. Snively, R. Campbell, and L. Davis. How well can simple metrics represent the performance of HPC applications? In *Proceedings of the 2005 ACM/IEEE Conference on Supercomputing*, Nov 2005.
- [6] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, Lee S-H, and K. Skadron. Rodinia: A benchmark suite for heterogeneous computing. In *Proceedings of the IEEE International Symposium on Workload Characterization*, Oct 2009.

- [7] S. Che, J. W. Sheaffer, M. Boyer, L. G. Szafaryn, L. Wang, and K. Skadron. A characterization of the rodinia benchmark suite with comparison to contemporary cmp workloads. In *Proceedings of the IEEE International Symposium on Workload Characterization*, Dec 2010.
- [8] 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 *Proceedings of Third Workshop on General-Purpose Computation on Graphics Processing Units*, Mar 2010.
- [9] N. Goswami, R. Shankar, M. Joshi, and T. Li. Exploring GPGPU workloads: Characterization methodology, analysis and microarchitecture evaluation implication. In *Proceedings of the IEEE International Symposium on Workload Characterization*, Dec 2010.
- [10] NVIDIA CUDA Programming Guide. Web resource. <http://developer.nvidia.com/object/gpucomputing.html>.
- [11] NVIDIA Profiler User's Guide. Web resource. <http://docs.nvidia.com/cuda/profiler-users-guide/index.html>.
- [12] J. Gustafson and R. Todi. Conventional benchmarks as a sample of the performance spectrum. In *Proceedings of the Thirty-first Hawaii International Conference on System Sciences*, Jan 1998.
- [13] 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 IEEE International Symposium on Workload Characterization*, Nov 2011.
- [14] S. Hong and H. Kim. An analytical model for a GPU architecture with memory-level and thread-level parallelism awareness. In *Proceedings of the 36th International Symposium on Computer Architecture*, June 2009.
- [15] K. Hoste and L. Eeckhout. Microarchitecture-independent workload characterization. *IEEE Micro*, 27(3):63–72, 2007.
- [16] K. Hoste, A. Phansalkar, L. Eeckhout, A. Georges, L. K. John, and K. D. Bosschere. Performance prediction based on inherent program similarity. In *Proceedings of the 15th International Conference on Parallel Architectures and Compilation Techniques*, Sept 2006.
- [17] W. Jia, K. A. Shaw, and M. Martonosi. Stargazer: Automated regression-based gpu design space exploration. In *Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software*, April 2012.
- [18] A. Joshi, A. Phansalkar, L. Eeckhout, and L. K. John. Measuring benchmark similarity using inherent program characteristics. *IEEE Transactions on Computers*, 55(6):769–782, 2006.
- [19] A. Kerr, G. Damos, and S. Yalamanchili. A characterization and analysis of PTX kernels. In *Proceedings of the 2009 International Symposium on Workload Characterization*, Oct 2009.
- [20] A. Kerr, G. Damos, and S. Yalamanchili. Modeling GPU-CPU workloads and systems. In *Proceedings of the Third Workshop on General-Purpose Computation on Graphics Processing Units*, April 2010.
- [21] E. Lindholm, J. Nickolls, S. Oberman, and J. Montrym. NVIDIA Tesla: A unified graphics and computing architecture. *IEEE Micro*, 28(2):39–55, 2008.
- [22] LINPACK. Web resource. <http://www.netlib.org/linpack/>.
- [23] J. Meng, V. A. Morozov, K. Kumaran, V. Vishwanath, and T. D. Uram. GROPHECY: GPU performance projection from CPU code skeletons. In *Proceedings of the ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, Nov 2011.
- [24] J. Meng, D. Tarjan, and K. Skadron. Dynamic warp subdivision for integrated branch and memory divergence tolerance. In *Proceedings of the 37th ACM/IEEE International Symposium on Computer Architecture*, June 2010.
- [25] OpenCL. Web resource. <http://www.khronos.org/opencl/>.
- [26] A. Phansalkar, A. Joshi, and L. K. John. Analysis of redundancy and application balance in the SPEC CPU2006 benchmark suite. In *Proceedings of the 34th International Symposium on Computer Architecture*, June 2007.
- [27] A. S. Phansalkar. *Measuring program similarity for efficient benchmarking and performance analysis of computer systems*. PhD thesis, Austin, TX, USA, 2007. AAI3285977.
- [28] B. Piccart, A. Georges, H. Blockeel, and L. Eeckhout. Ranking commercial machines through data transposition. In *Proceedings of the IEEE International Symposium on Workload Characterization*, Nov 2011.
- [29] A. Snaveley, L. Carrington, N. Wolter, J. Labarta, R. Badia, and A. Purkayastha. A framework for performance modeling and prediction. In *Proceedings of the 2002 ACM/IEEE Conference on Supercomputing*, Nov 2002.
- [30] The Standard Performance Evaluation Corporation (SPEC). Web resource. <http://www.spec.org>.
- [31] STREAM. Web resource. <http://www.cs.virginia.edu/stream/>.
- [32] Parboil Benchmark Suite. Web resource. <http://impact.crhc.illinois.edu/parboil.php>.
- [33] SPEC Accelerator Benchmark Suite. Web resource. <http://www.spec.org/hpg/press/acceleratorbmk.html>.
- [34] MATLAB Statistics Toolbox. Web resource. <http://www.mathworks.com>.
- [35] H. Wong, M.-M. Papadopoulou, M. Sadooghi-Alvandi, and A. Moshovos. Demystifying gpu microarchitecture through microbenchmarking. In *Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software*, April 2010.
- [36] 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*, June 1995.
- [37] E. Z. Zhang, Z. Guo, Y. Jiang, K. Tian, and Xipeng Shen. On-the-fly elimination of dynamic irregularities for GPU computing. In *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems*, Mar 2011.
- [38] Y. Zhang and J. D. Owens. A quantitative performance analysis model for GPU architectures. In *Proceedings of the 17th IEEE International Symposium on High-Performance Computer Architecture*, February 2011.



Shuai Che is a researcher at AMD Research. He received his Ph.D. in Computer Engineering from the University of Virginia, advised by Prof. Kevin Skadron. His research interests include computer architecture, parallel and heterogeneous computing, parallel algorithms and GPU programming. His work includes the development of the Rodinia Benchmark Suite for Heterogeneous Computing. He is a member of ACM and IEEE.



Kevin Skadron is professor and department chair of the Department of Computer Science at the University of Virginia, where he has been on the faculty since 1999. His research focuses on heterogeneous architecture and physical constraints such as power, temperature, and reliability; and developed (together with Che) the Rodinia Benchmark Suite for Heterogeneous Computing. Skadron is an IEEE Fellow and ACM Distinguished Scientist, and recipient of the 2011 ACM SIGARCH Maurice Wilkes Award.