A Characterization of the Rodinia Benchmark Suite with Comparison to Contemporary CMP Workloads

Shuai Che
sc5nf@virginia.edu

Jeremy W. Sheaffer
jws9c@cs.virginia.edu

Michael Boyer
mwb7w@cs.virginia.edu

Lukasz G. Szafaryn
lgs9a@virginia.edu

Liang Wang
lw2aw@virginia.edu

Kevin Skadron
skadron@cs.virginia.edu

The University of Virginia
Department of Computer Science

***Abstract*—The recently released Rodinia benchmark suite enables users to evaluate heterogeneous systems including both accelerators, such as GPUs, and multicore CPUs. As Rodinia sees higher levels of acceptance, it becomes important that researchers understand this new set of benchmarks, especially in how they differ from previous work. In this paper, we present recent extensions to Rodinia and conduct a detailed characterization of the Rodinia benchmarks (including performance results on an NVIDIA GeForce GTX480, the first product released based on the Fermi architecture). We also compare and contrast Rodinia with Parsec to gain insights into the similarities and differences of the two benchmark collections; we apply principal component analysis to analyze the application space coverage of the two suites. Our analysis shows that many of the workloads in Rodinia and Parsec are complementary, capturing different aspects of certain performance metrics.**

I. Introduction

Computer systems are increasingly exposing a heterogeneous computing model consisting of accelerators—such as graphics processors (GPUs), media processors, and even reconfigurable hardware like FPGAs—combined with one or more conventional CPUs. GPUs, for instance, offer parallelism at scales unachievable with other processors and afford about an order of magnitude greater peak throughput than general-purpose, multicore CPUs, while the CPUs offer high single-thread performance and programmability.

A vision of heterogeneous computer systems that incorporate diverse accelerators and automatically select the best computational unit for a particular task is widely shared among researchers and many industry analysts; however, there are no agreed-upon benchmarks to support the research needed in the development of such a platform. There are many benchmark suites for parallel computing on general-purpose CPU architectures, but accelerators fall into a gap that is not covered by current benchmark suites or benchmark development. There is a dearth of publicly available code for heterogeneous platforms.

The Rodinia benchmark suite [8], a set of free and open benchmarks and associated methodologies, was developed to address these concerns. The Rodinia applications are designed for *heterogeneous* computing infrastructures, and, using OpenMP and CUDA, target both GPUs and multicore CPUs. The implementations for each distinct platform can also serve as independent suites to evaluate multicore and manycore architectures separately. The Rodinia suite is structured to span a range of parallelism and compute patterns, providing researchers with various feature options to identify architectural bottlenecks and to fine tune hardware designs.

Several multithreaded benchmark suites for multicore CPUs, including SPLASH-2 [35], Parsec [5], and SPEC OMP [29], are available. Rodinia was developed to address the issues of benchmarking heterogeneous systems, particularly those including a GPU. There is growing support for use of the Rodinia workloads [6], [8]–[10], [24], but there are some important questions yet to be answered:

- How much do those Rodinia workloads which are designed for heterogeneous platforms (those with GPU accelerators) differ from those of other suites designed for multicore CPUs?
- Do the workload designs of other suites demonstrate overlapping or orthogonal features?
- How well do the chosen applications span the workload space?
- How well can traditional, multithreaded CPU workloads map onto GPU platforms?

A better understanding of these issues will not only expand the knowledge of parallel benchmark construction, but could also inform decisions on workload scheduling and partitioning on different architectures and guide researchers to choose appropriate benchmarks for their research as well.

In this paper we make the following contributions:

- We present important extensions to the Rodinia benchmark suite that have been added since its initial publication at IISWC 2009 [8].
- We conduct a more detailed characterization of the Rodinia GPU workloads to aid researchers in understanding

the characteristics of Rodinia.
- We evaluate the Rodinia benchmarks on a recently released NVIDIA GTX480, which is based on the Fermi architecture with traditional L1 and L2 caches, identifying some bottlenecks of the new GPU architecture.
- We perform an application space study, comparing the multithreaded CPU implementations of Rodinia with those of Parsec, and evaluate the extent to which the program selections of the two suites overlap.
- We present analysis and discussion of important, open research topics, including the need for new parallel performance metrics, for an effective application taxonomy, and for a general application space study of multithreaded workloads, and we discuss the challenges that make porting existing suites difficult.

II. Overview of Rodinia

As apposed to Parsec and SPLASH-2, which target homogeneous platforms, Rodinia workloads are selected and designed for heterogeneous computing platforms including both CPUs and devices such as GPUs and FPGAs [10]. Rodinia not only covers applications from emerging domains such as bioinformatics, data mining, and image processing, but also includes the accelerator implementations of important, classical algorithms like LU decomposition and graph traversal. The Berkeley Dwarf taxonomy [1] was initially used as a guideline to choose applications for Rodinia in order to avoid missing important parallel patterns. Table I illustrates the Rodinia applications and their corresponding domains and Dwarves.

The Rodinia benchmarks are currently implemented in OpenMP and CUDA. As OpenCL provides an attractive alternative to CUDA, we are producing OpenCL ports, as well; these are not complete, nor as mature as the OpenMP and CUDA implementations. OpenCL and CUDA use very similar sets of abstractions, such that CUDA is sufficient for the characterization and diversity analysis presented in this paper. We expect that our reported results will transfer directly to the OpenCL ports when they are complete.

Rodinia has some important features that differentiate it from other benchmark suites:

- Rodinia implementations take advantage of non-traditional memory hierarchies, like scratchpad and texture units, for general purpose computation. Cell and ClearSpeed are two examples in a trend to use other types of memories as alternatives to hardware-managed cache. This trend in turn requires benchmark development to keep up with such an evolution.
- Rodinia provides multiple versions of some applications, with successive layers of optimization, allowing designers to evaluate the impact of multiple different implementations on their architecture or compiler designs.
- Rodinia's applications adopt an "offloading" model which assumes that accelerators use a memory space disjoint from main memory.
- Rodinia provides a set of applications from which it may be relatively hard for compilers to automatically generate accelerator code.

A. Rodinia Extensions

Since the release of the first version of Rodinia [8], we have been adding applications to enrich the workload set. The newly added applications include *Heartwall-Tracking*, *LU Decomposition*, *MUMmer* and *Computational Fluid Dynamics*. The major criterion applied in selecting these applications was their use of advanced data structures. These data structures allow the applications to demonstrate new types of parallelism and inter-thread communications not seen in other members of the Rodinia suite.

LU Decomposition (LUD): LU Decomposition is an algorithm to calculate the solutions of a set of linear equations. The LUD kernel decomposes a matrix as the product of a lower triangular matrix and an upper triangular matrix. This application has many row-wise and column-wise interdependencies and requires significant optimization to achieve good parallel performance.

Motivation for inclusion: LU Decomposition exhibits significant inter-thread sharing and row and column dependencies.

Heartwall Tracking (HW): The Heart Wall [31] application tracks the changing shape of the walls of a mouse heart over a sequence of 104 ultrasound images, each with a resolution of 609×590 pixels. In its initial stage, the program performs several image processing passes—edge detection, SRAD despeckling (part of Rodinia), morphological transformation, and dilation—on the first image in the sequence in order to detect partial shapes of inner and outer heart walls. To reconstruct approximated, full shapes of heart walls for tracking purposes, the application generates ellipses that are superimposed over the image and sampled to mark points on the heart walls. In its final stage, the program tracks the changing shapes of the two heart walls by detecting the movement of certain sample points throughout the sequence of images.

Motivation for inclusion: Heartwall Tracking presents a pattern of *braided parallelism*—a mixture of data and task parallelism—which is absent from other Rodinia benchmarks. The application is coarsely parallelized according to independent tasks (TLP); each task is then finely parallelized according to independent data operations (DLP). The processing of a frame is implemented as a single GPU kernel in order to successfully implement braided parallelism and avoid kernel launch overhead. This structure requires the inclusion of some non-parallel computation into the kernel, leading to a slight warp under-utilization but overall greater performance.

Computational Fluid Dynamics (CFD): The CFD solver is an unstructured-grid, finite-volume solver for the three-dimensional Euler equations for compressible flow. Effective GPU memory bandwidth is improved by reducing total global memory accesses and overlapping redundant computation, as well as by using an appropriate numbering scheme and data

TABLE I
RODINIA APPLICATIONS AND KERNELS ('*' DENOTES KERNEL).

Application	Dwarf	Domain	Problem Sizes
Kmeans	Dense Linear Algebra	Data Mining	204800 data points, 34 features
Needleman-Wunsch (NW)	Dynamic Programming	Bioinformatics	2048×2048 data points
HotSpot* (HS)	Structured Grid	Physics Simulation	500×500 data points
Back Propagation* (BP)	Unstructured Grid	Pattern Recognition	65536 input nodes
SRAD*	Structured Grid	Image Processing	512×512 data points
Leukocyte Tracking (LC)	Structured Grid	Medical Imaging	219×640 pixels/frame
Breadth-First Search* (BFS)	Graph Traversal	Graph Algorithms	1000000 nodes
Stream Cluster* (SC)	Dense Linear Algebra	Data Mining	65536 points, 256 dimensions
MUMmer (MUM)	Graph Traversal	Bioinformatics	50000 25-character queries
CFD Solver (CFD)	Unstructured Grid	Fluid Dynamics	97k elements
LU Decomposition* (LUD)	Dense Linear Algebra	Linear Algebra	256×256 data points
Heart Wall Tracking (HW)	Structured Grid	Medical Imaging	609×590 pixels/frame

layout. The CFD solver is released with two versions: one with precomputed fluxes, and the other with redundant flux computations. CFD is an implementation of the work by Corrigan *et al.* [11].

Motivation for inclusion: The CFD implementation applies data layout optimizations to reduce uncoalesced memory accesses to GPU memory. It also provides both single-precision and double-precision floating point implementations for the GPU, which allows users to analyze the trade-off between performance and computational precision. Also, computational fluid dynamics is widely regarded as a very important scientific workload.

MUMmerGPU (MUMmer): MUMmerGPU, developed by Schatz *et al.* [28], is an high-throughput, parallel, pairwise, local-sequence alignment program. It uses the GPU to simultaneously align multiple query sequences against a single reference sequence stored as a suffix tree encoded with 2D textures. The tree of the reference sequence is constructed on the CPU using Ukkonen's Algorithm [33] and transferred to the GPU along with the query sequences. The query sequences are then transfered to the GPU, and are aligned with the tree on the GPU.

Motivation for inclusion: The working set and code size of MUMmer is significantly larger than other benchmarks, stressing memory systems. Additionally, the suffix tree implementation presents the challenge of mapping Mummer's data structures to a GPU computational model while utilizing efficient data layouts.

TABLE II
GPGPU-SIM CONFIGURATIONS.

Parameter	Value	Parameter	Value
Clock Frequency	2 GHz	No. of CTAs/Core	8
No. of SMs	28	Number of Registers/Core	16384
Warp Size	32	Shared Memory/Core	32 kB
SIMD pipeline width	32	Shared Memory Bank Conflict	True
No. of Threads/Core	1024	No. of Memory Channels	8

III. CHARACTERIZATION OF RODINIA

In this section, we characterize Rodinia's applications in terms of instructions per cycle (IPC), memory instruction mix, and warp divergence. Our analysis shows that the Rodinia applications demonstrate good diversity, and the addition of new benchmarks enriches the application coverage of the suite beyond that of the previous Rodinia release [8]. We also use Rodinia to benchmark the NVIDIA GeForce GTX480 GPU (Fermi) targeting each of L1 and shared memory as preferred configurations.

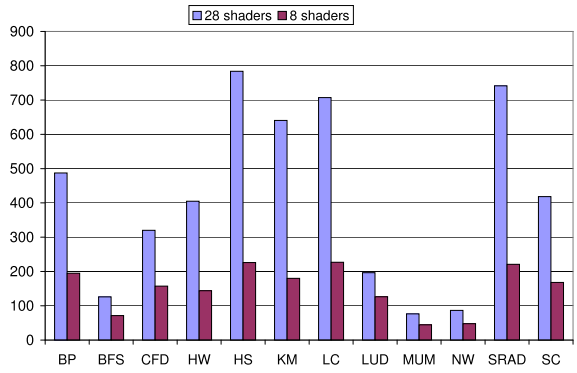

Fig. 1. IPCs are measured over 8-shader and 28-shader configurations.

A. Experiment Setup

To measure the execution characteristics of the Rodinia GPU benchmarks, we use GPGPU-Sim [2] from the University of British Columbia. GPGPU-Sim provides a detailed simulation model of a contemporary GPU capable of running CUDA and OpenCL workloads. Table II shows the parameters we used to configure the simulator.

Our GPGPU-Sim simulations did not use an L2 cache. Table I lists input details for the Rodinia applications we used in the simulations. In the Fermi benchmarking experiments, we use an NVIDIA GeForce GTX480 with 15 streaming multiprocessors (SMs) with a total of 480 1.4 GHz streaming processors (SPs), and a 768 kB L2. Each SM has a 64 kB, configurable, on-chip memory that can be configured as 48 kB shared + 16 kB L1 or as 16 kB shared + 48 kB L1. We use NVIDIA CUDA 2.2 for the GPGPU-Sim simulations (the simulator currently supports up to CUDA 2.3); for the GTX 480 experiments, we use CUDA version 3.0.

B. GPU Benchmark Results

Figure 1 shows the IPCs of each of the Rodinia benchmarks measured with 28-shader—the default configuration provided

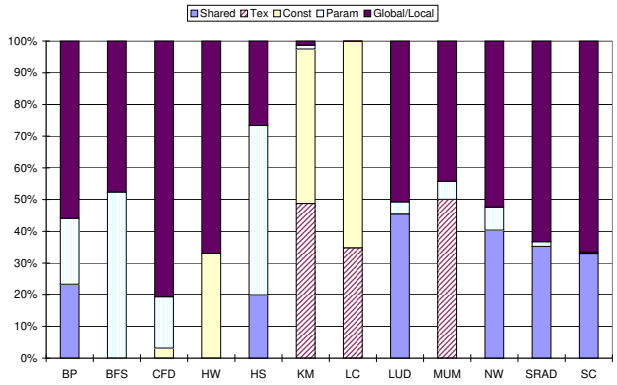

Fig. 2. Memory operation breakdown in terms of shared, texture, constant, parameter, and local or global memory instructions. "Param" memory refers to parameters passed through the GPU kernel call, which we always treat as cache hits [2].

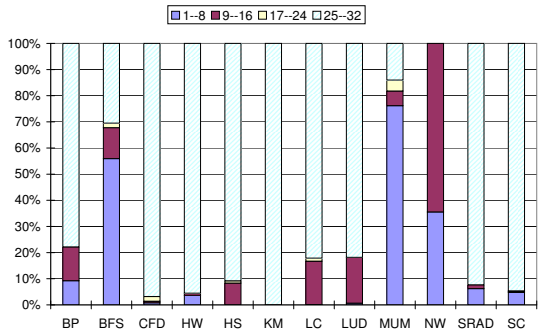

Fig. 3. Warp occupancies show the numbers of active threads in an issued warp over the entire runtime of the benchmark [2].

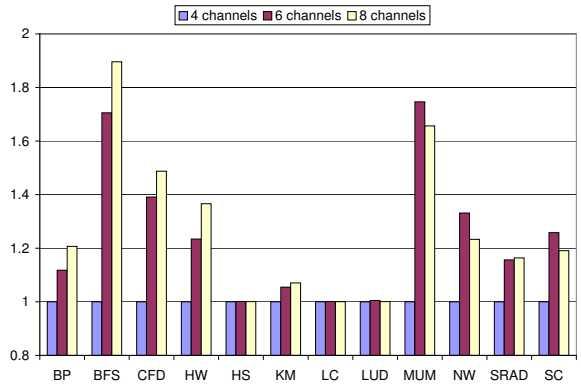

Fig. 4. Memory bandwidth improvements while increasing the number of memory channels. All data is normalized to the 4-channel case.

by the GPGPU-Sim package [2] with a SIMD width of 32—and 8-shader configurations. The IPCs with the 28-shader configuration range from less than 100 in *MUMmer* and *Needleman-Wunsch*) to more than 700 in *SRAD*, *HotSpot*, and *Leukocyte*. The highest IPCs are usually due to massive parallelism, better usage of memory locality, and good algorithmic optimization [6], [8], [24], [31]. Low IPC can be attributed to any of myriad faults: there is limited parallelism per iteration in *Needleman-Wunsch* due to the dependencies of processing data elements in a diagonal strip manner [9]; the overhead of the GPU's global memory accesses dominates *Breadth-First Search*; and some applications present many divergent branches. The benchmarks show high scalability across 8 and 28 shaders, except for those like *MUMmer* and *Breadth-First Search*, which are limited by the global memory access bandwidth, and like *LUD* with significant row and column dependencies.

Many Rodinia benchmarks take advantage of the GPU's specialized memory spaces by localizing data access patterns and inter-thread communication within thread blocks to take advantage of the SM's per-block shared memory. For read-only data structures, binding to cached constant or texture memory to reap the benefits of caching can provide significant performance improvements. Figure 2 shows a breakdown of different types of memory accesses. Applications such as *Back Propagation*, *HotSpot*, *Needleman-Wunsch* and *StreamCluster* make extensive use of shared memory. The performances of *Kmeans*, *Leukocyte* and *MUMmer* are improved by taking advantage of texture memory. Differing from *Kmeans* and *Leukocyte*, *Heartwall* uses constant memory to store large numbers of parameters which cannot be readily fit into shared memory.

Figure 3 shows warp occupancies [2]—the average number of active threads over all issued warps—over the entire runtime of the benchmarks. In a SIMT model [25], the cores will achieve the best performance when the threads within a SIMT group follow the same execution path. For example, because it must determine whether or not neighboring nodes have been visited, *Breadth-First Search* contains many control flow operations; hence the high number of low occupancy warps. *SRAD* does not have much control flow; what of it there is deals with the loading and processing data elements lying on the boundaries between data blocks. *Heartwall* must determine the specific operations to execute on the various regions of the image, but this requires a relatively small portion of the calculation, and the rest of the computation executes with little control flow.

For applications other than *Breadth-First Search*, *SRAD*, and *Heartwall*, unfilled warps are not due to branch divergence. Only some of the threads in *Back Propagation* are active, due to the parallel reduction; assuming a 16-element sum reduction, the number of active threads during the four iterations are 8, 4, 2 and 1. A similar situation occurs in *Needleman-Wunsch*, where, in each thread-block, the number of active threads is less than 16. *MUMmer*, in particular, experiences severe performance penalties because more than 60% of its warps have less than 5 active threads [2].

Figure 4 shows the bandwidth improvement as we increase the number of memory channels from 4 to 8. The benchmarks which benefit most from this change include *Breadth-First Search*, *CFD* and *MUMmer*. *LUD* and *HotSpot*, which take advantage of shared memory locality, benefit less from increased memory channels. For *Kmeans* and *Leukocyte*, little

improvement occurs with additional channels because we bind their main data structures to texture memory and make use of use constant memory.

C. Incrementally Optimized Versions

One important distinguishing characteristic of Rodinia is its support for multiple versions of individual benchmarks. These *incremental versions* are useful tools for architects and compiler developers because they allow analysis of the impact of hardware and software design choices on problems that are fundamentally the same but differ in certain specifics. Incremental versions can be used by programmers and compiler developers as "road maps" for similar problems, to aid them in getting from unoptimized to optimized or to evaluate their own optimizations.

We are preparing to release incremental code versions of *Leukocyte*, *LUD*, *Needleman-Wunsch* and *SRAD*. Table III shows sample characteristics of two different versions of *SRAD* and *Leukocyte*. We apply more shared memory optimization on the second version of SRAD, thus increasing the IPC from 404 to 748. Similarly, the performance of Leukocyte version 2 is improved by reducing the percentage of long latency global memory accesses through the use of persistent thread blocks. Boyer *et al.* provide a detailed study on the optimization of *Leukocyte* [6].

TABLE III
INCREMENTALLY OPTIMIZED VERSIONS OF SRAD AND LEUKOCYTE.

Benchmarks	Statistics	
SRAD	Version 1	IPC: 404, BW Utilization: 26% Shared: 9.7%, Global: 49.3% (Mem. inst. mix)
	Version 2	IPC: 748, BW Utilization: 34% Shared: 28.9%, Global: 51.9%
Leukocyte	Version 1	IPC: 656, BW Utilization: 8% Const: 54.1%, Tex: 22.7%, Global: 7.7%
	Version 2	IPC: 707, BW utilization: 3% Const: 65.1%, Tex: 34.7%, Global: 0.0%

D. Fermi Evaluation

Unlike the earlier G80 and Tesla products, NVIDIA's Fermi architecture includes traditional L1 and L2 caches. Each SM has 64 kB of on-chip memory that can be configured as 48 kB of shared memory and 16 kB of L1 (*shared bias*), the default configuration) or as 16 kB of shared memory and 48 kB of L1 (*L1 bias*. CUDA provides a new API function, `cudaFuncSetCacheConfig()`, to select the desired configuration [13]. A unified L2 cache handles all memory requests for data loads and stores, as well as all texture fetches.

Figure 5 shows the results obtained measuring the performance of the Rodinia CUDA implementations on an NVIDIA GeForce GTX480 GPU with each memory configuration. We compare to the results on a GTX280 GPU with 240 1.3 GHz SPs and 1 GB of device memory. All the measurements are kernel execution times normalized to the GTX280. Excepting *LUD* and *Leukocyte*, the total workload size of all benchmarks is larger than the aggregate L1 capacity.

The performances of *MUMmer* and *BFS*, which have large numbers of global memory accesses, improve by 11.6% and 16.7% respectively after switching the configuration from shared bias to L1 bias. Many Rodinia applications, including *SRAD*, *Needleman-Wunsch* and *Leukocyte*, which are designed to utilize shared memory well, expectedly prefer the shared bias setting. *LU Decomposition* and *StreamCluster* show very little performance variation between the two configurations.

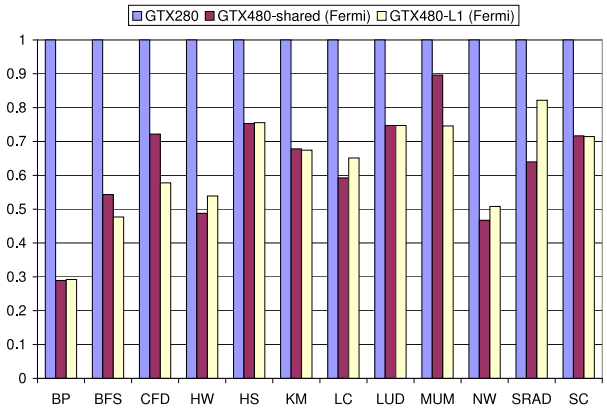

Fig. 5. Normalized kernel execution time of the GPU implementations on a GTX 280 and a GTX 480 (Fermi). Two configurations (L1 and shared bias) are used for the GTX 480 measurements.

E. GPU Sensitivity Study

To help architects achieve a desirable design, it is necessary to identify key architectural parameters and understand how benchmarks respond to changes in those parameters. Yi *et al.* [36] proposed using the Plackett-Burman (PB) design approach to determine the effect of a parameter on a processor's performance. Given n architectural parameters, PB requires only approximately $2n$ simulations instead of the 2^n required for brute-force full coverage. The downside of the PB methodology is its inability to effectively quantify the effects of all the interactions; however, Yi *et al.* show that if an interaction between parameters was significant, it was significant only because each of its constituent parameters was individually significant.

We use the PB approach to evaluate a subset of GPU architectural parameters that are important to performance. The parameters we choose include core clock rate (1.2 GHz–1.5 GHz), SIMD width (16–32), shared memory size (16 kB–32 kB), bank conflict (on or off), register file size (16384–32768 registers), number of threads (1024–2048), memory clock (800 MHz–1 GHz), number of memory channels (4–8), and DRAM bus width (4–8 bytes). We use GPGPU-sim to configure these parameters and evaluate the resulting changes of total execution cycles with the 11-column PB matrix.

SIMD width and the number of memory channels have the largest impacts on benchmark performance, often demonstrating more than an order of magnitude greater effect than other parameters; however, different applications show different characteristics. For instance, *SRAD* makes extensive uses of the shared memory; for this application, the configurations of shared memory and the number of memory channels have similar impact on performance. Similarly, shared memory bank

conflict, SIMD-width, and memory bandwidth demonstrate similar influence on performance for *Needleman Wunsch*. This is attributable to the fact that the current implementation processes diagonal strips on a 16×16 shared memory block, which leads to copious bank conflict and suggests that we have space for further optimization. The applications, including *Leukocyte* and *HotSpot*, which have better on-chip data locality or utilize texture memory units, are impacted only modestly by changes to the memory interfaces.

Note that some complex interactions between parameters cannot be captured by Plackett-Burman. For example, the number of thread blocks that can be issued is often limited by another resource limitation of the GPU (the number of registers or threads, or the shared memory size). It is possible that once limited by one constraint, providing extra resources for structures does not lead to any performance improvement; this is a question we plan to explore in future work.

IV. Rodinia and Parsec

This section answers several important questions: 1) How do workloads, like Rodinia, which are designed for heterogeneous platforms differ from those of other suites designed for multicore CPUs, like Parsec? 2) How well do the chosen applications span the workload space? 3) What aspects of Parsec and Rodinia are differentiating? We hope that examination of these questions may facilitate the improvement of workload construction for multicore CPU and accelerator performance analysis.

How to perform fair comparisons between accelerator and CPU workloads running on different architectures is an open research question, and one which we cannot adequately address in this paper. Among the difficulties in heterogeneous, parallel benchmarking are the questions of 1) algorithm choice: How alike are the underlying algorithms of two different implementations? 2) optimization: What does it mean to compare the quantity and quality of optimization across heterogeneous platforms? 3) effort: If performance is not the sole concern, the next item on the list is probably cost or programmer effort. How difficult is an application to implement [8]–[10]?

The Rodinia OpenMP and CUDA implementations are developed congruously, using same algorithms with similar levels of optimization; we reserve formal evaluation of their similarity for future work. We use the Rodinia OpenMP implementations to compare with the Parsec benchmarks in this study. We apply principal component analysis (PCA) to identify distinctions and also to characterize the workloads in terms of cache behavior, working set, and other, similar performance metrics.

A. Comparison of Rodinia and Parsec

Parsec, a benchmark suite jointly developed by Princeton University and Intel, has been gradually gaining popularity among users of multithreaded workloads. The suite includes some workloads from emerging application domains and uses some state-of-the-art software techniques. Bienia *et al.* [4] compare SPLASH-2 and Parsec to determine the extent of feature overlap, and conclude that the workloads have significant differences. Many Parsec workloads have larger working sets than those in SPLASH-2, useful in the face of the scientific trend toward massive data growth. Other work compares the communication characteristics of SPLASH-2 and Parsec [34] and examines the behavior of Parsec on real hardware [3].

TABLE IV
Comparison between Parsec and Rodinia.

Features↓ \ Suite→	**Parsec**	**Rodinia**
Platform	CPU	CPU and GPU
Programming Model	Pthreads, OpenMP, and TBB	OpenMP and CUDA
Machine Model	Shared Memory	Shared Memory and Offloading
Application Domains	Scientific, Engineering, Finance, Multimedia	Scientific, Engineering, Data Mining
Application Count	3 Kernels and 9 Applications	6 Kernels and 6 Applications
Optimized for...	Multicore	Manycore and Accelerator
Incremental Versions	No	Yes
Memory Space	HW Cache	HW and SW Caches
Problem Sizes	Small–Large	Small–Large
Special SW Techniques	SW Pipelining	Ghost-zone and Persistent Thread Blocks
Synchronization	Barriers, Locks, and Conditions	Barriers

Table IV provides a high-level overview of the differing design focuses of Parsec and Rodinia, while Table V provides some more specific details on Parsec. In the previous sections, we discussed several aspects of Rodinia which distinguish it from other benchmark suites; here we provide some more discussion on the topic, specifically with respect to Parsec.

Parsec provides a rich set of features that support fine-grained parallelism (locks), languages (TBB, OpenMP, and Pthreads), and large code bases. Rodinia currently focuses only on OpenMP workloads for the CPU implementations. The use of fine-grained parallelism in Rodinia, even in CPU implementations, is restricted by our desire to maintain algorithmic congruence with the CUDA ports given the fact that CUDA supports only barrier synchronization within a thread block [22] and global synchronization at kernel exit or when using a global synchronization primitive. In the construction of the Rodinia benchmark suite, we also consider Parsec workloads. We include *StreamCluster* in Rodinia, but find that those benchmarks relying on task pipelining, like *Ferret*, do not port well unless each stage is also heavily parallelizable.

B. Methodology

To compare Rodinia and Parsec, we adopt the methodology and metrics of Bienia *et al.* [4] in their SPLASH-2 and Parsec comparison, so that the reported results are cross-comparable. The points of comparison include instruction mix (including ALU, branch, and memory instructions), working set (cache misses per memory reference), and sharing behavior (the fraction of cache lines shared, and the number of accesses to shared lines per memory reference). Our experiments use eight cache sizes, ranging from 128 kB to 16 MB, and measure the sharing and the working set behavior. We adopt a similar cache structure to that used by Bienia *et al.* as well, an 8-core processor with a single cache shared by all cores. The cache is 4-way associative with 64 byte lines. All programs are compiled with gcc 4.2.1 with OpenMP or Pthreads.

All data is obtained with Pin [23]. Pin is a dynamic, binary instrumentation tool that instruments an application that executes on Intel processors. It provides an infrastructure for

TABLE V
PARSEC APPLICATIONS AND SIM-LARGE INPUT SIZES. [3], [4]

Application	Application Domain	Problem Size	Description
Blackscholes	Financial Analysis, Algebra	65,536 options	Portfolio price calculation using Black-Scholes PDE
Bodytrack	Computer Vision	4 frames, 4,000 particles	Computer vision, tracks 3D pose of human body
Canneal	Engineering	400,000 elements	Synthetic chip design, routing
Dedup	Enterprise Storage	184 MB	Pipelined compression kernel
Facesim	Animation	1 frame, 372,126 tetrahedrons	Physics simulation, models a human face
Ferret	Similarity Search	256 queries, 34,973 images	Pipelined audio, image and video searches
Fluidanimate	Animation	5 frames, 300,000 particles	Physics simulation, animation of fluids
Freqmine	Data Mining	990,000 transactions	Data mining application
StreamCluster	Data Mining	16,384 points per block, 1 block	Kernel to solve the online clustering problem
Swaptions	Financial Analysis	64 swaptions, 20,000 simulations	Computes portfolio prices using Monte-Carlo simulation
Vips	Media Processing	1 image, 26,625,500 pixels	Image processing, image transformations
X264	Media Processing	128 frames, 640,360 pixels	H.264 video encoder

writing program analysis tools, called *Pin tools*. Instruction mix is obtained using the *mix-mt* tool provided with the Pin package. We developed our own Pin tool, based on the cache tool in Pin, with support for multithreaded workloads to collect cache behavior characteristics.

C. Principal Component Analysis and Measuring Similarity

Principal components analysis (PCA) is a statistical, data analysis technique that reduces a data set's dimensionality and removes correlation from the data set while controlling the amount of information lost. PCA computes n new variables, called principal components, which are linear combinations of n original variables, such that all principal components are uncorrelated. The first of the resulting orthogonal principal components exhibits the largest variance, followed by the second, followed by the third, and so on [15]. After performing PCA, we cluster to find equivalence classes of programs with similar characteristics. PCA has been widely applied for benchmark comparison [4], [15], [18], [26] in similar contexts; however, the question of how to perform more fair and accurate evaluation and comparison of benchmarks is an open one and beyond the scope of this paper.

To measure the similarity among benchmarks, we use classical hierarchical clustering analysis. Similar approaches have been used in other recent performance analysis work. Clusters are formed in such a way that data objects in the same cluster are very similar and data objects in different clusters are very distinct. We use the MATLAB [32] statistics toolbox to process the data for the collected characteristic values for all the benchmarks. The algorithm involves finding the similarity or dissimilarity between every pair of data objects in the data set using distance functions and grouping the objects into a binary, hierarchical cluster tree. Dendrograms are used to illustrate our results.

V. ANALYSIS

Here we present the results of our principal component analysis.

A. Hierarchical Clustering

Figure 6 shows the overlap of the two program collections. In the figure, the magnitude of the link between any two nodes (or clusters of nodes) quantifies the measure of dissimilarity

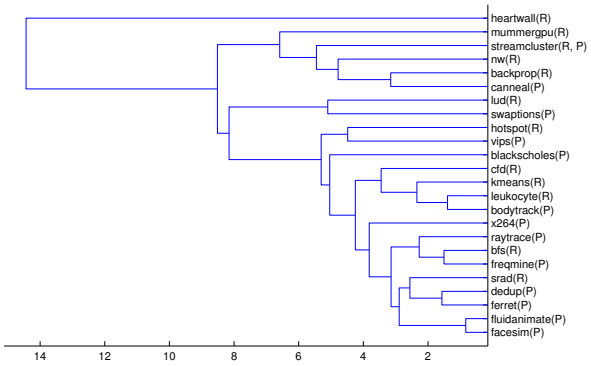

Fig. 6. A dendrogram showing the similarity between the Parsec (P) and Rodinia (R) workloads. The x axis represents the linkage distance in the PCA coverage space, which has no obvious physical analog.

between those nodes; thus, *Leukocyte* and *Bodytrack* are fairly similar, while *Heartwall* differs significantly from all other compared benchmarks; and *MUMmer* and *Swaptions*, while spatially close in the figure, are more dissimilar than *HotSpot* and *Facesim*. From this dendrogram, it is evident that the two benchmark suites cover similar application spaces, with most clusters containing both Rodinia and Parsec applications. Also note that the new applications added to Rodinia, namely *CFD*, *LUD*, *MUMmer*, and *Heartwall*, enrich the original application set, with the latter two significantly different from all others.

We also perform an analysis of some subsets of our characteristics: instruction mix, cache miss rate, and data sharing behavior. Instruction mix is an interesting metric because it represents a set of fundamental program characteristics and the utilization of various hardware components, while the cache miss rate characterizes data locality and reuse, and an application's sharing behavior is important to multithreading workloads and communications [4].

Figures 7, 8 and 9 show the instruction mix, working set and sharing behaviors of the programs as is similarly shown in the Parsec and SPASH-2 comparison [4]. In Figure 7, Parsec and Rodinia demonstrate disparate behavior, with *Breadth-First Search*, *Back Propagation*, and *HotSpot* from Rodinia, and *Raytrace*, *Ferret*, *Bodytrack*, and *StreamCluster* from Parsec

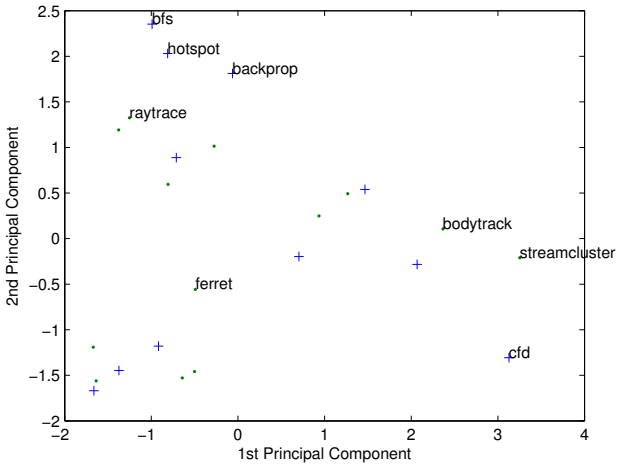

Fig. 7. The instruction mix plot using two PCA components for Parsec (dots) and Rodinia (crosses).

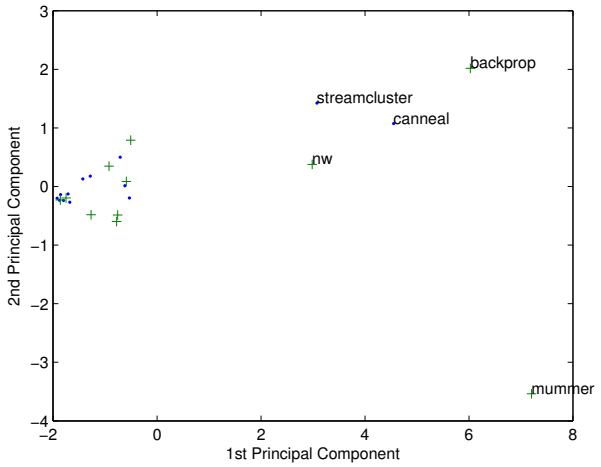

Fig. 8. The working set plot using two PCA components for Parsec (dots) and Rodinia (crosses).

tending to populate different areas in the space. In the working set plot of Figure 8, there are several Parsec and Rodinia benchmarks that are clear outliers from the main cluster; *MUMmer* is a significant outlier, which correlates with its high miss rates. The miss rates—given in cache misses per memory reference—of all the benchmarks under a 4 MB cache configuration are shown in Figure 10. Figure 9 shows similar behavior—data sharing, now, rather than working set size—with *Heartwall* significantly different from the rest. Looking back at Figure 6, *Heartwall* and *MUMmer* are the most disparate benchmarks in the suite; something which is backed up by this series of figures.

As is evident from these figures, Rodinia clearly provides a good workload mix for multicore CPUs. Additionally, Parsec and Rodinia demonstrate features that complement with each other, suggesting that researchers should consider both of them, possibly as well as other benchmark suites, to ensure a reasonable application coverage for their work.

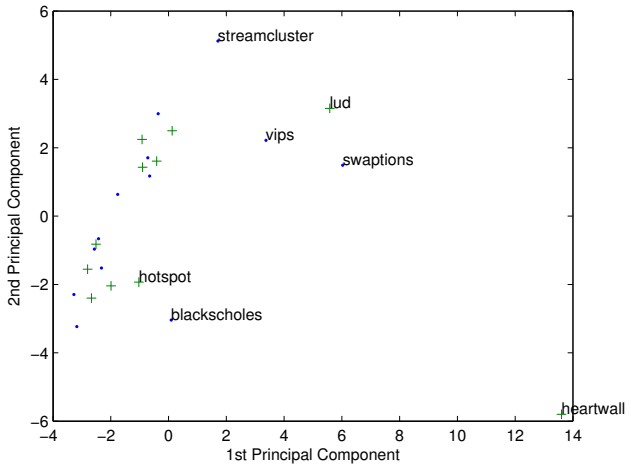

Fig. 9. The sharing plot using two PCA components for Parsec (dots) and Rodinia (crosses).

B. Clustering Discussion

How well is the application space covered by the two suites?

— Our clustering analysis shows that Parsec and Rodinia cover similar application spaces. This does not imply that using either or both of them is sufficient for research. Consider the blank regions in the PCA coverage spaces of Figures 7, 8, and 9; it is unclear whether these regions can be covered by other real-world workloads or benchmark suites. This implies that a thorough examination requires a comprehensive evaluation and comparison of all the current multithreaded benchmark suites, including SPLASH-2 and various domain-specific workloads, to establish a single set of workloads with sufficient coverage and little redundancy. Previous work performs such studies but only on single-threaded benchmarks [18], [26], while this is an open problem in heterogeneous environments.

The metrics evaluated in this work are important for multithreaded program behavior [4]. On the other hand, other potentially important metrics may indicate other crucial differences between benchmarks. It is an area of ongoing research to develop a set of metrics which are able to capture most behaviors of multithreaded workloads. Host *et al.* [15] propose a set of microarchitecture-independent workload characteristics to profile single-threaded applications, and which are also useful for performance prediction [16]. A set of metrics for multithreaded workloads are needed.

Can the Parsec workloads be effectively mapped to heterogeneous platforms?

— Our clustering results indicate that, except for a few

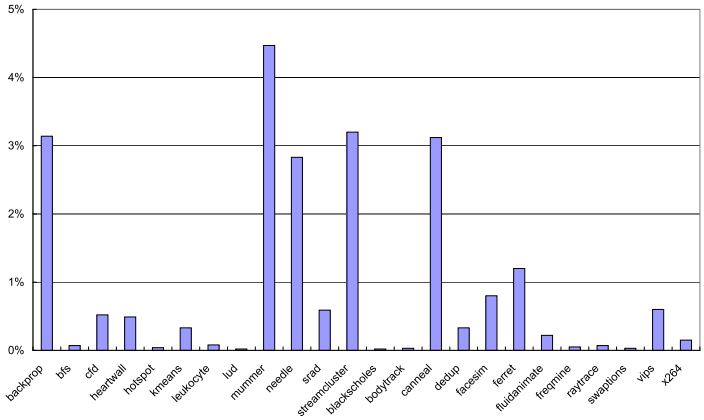

Fig. 10. The miss rates of Rodinia and Parsec [4] benchmarks under a 4 Mb cache configuration.

outliers, the heterogeneous workloads we developed for Rodinia are not fundamentally different from those of Parsec, developed for multicore CPUs. This, however, does not imply that since all the Rodinia benchmarks map well to the GPU platform, the same will be true of the Parsec benchmarks. We have found many challenges in the task of porting traditional, multithreaded, CPU workloads onto heterogeneous platforms. Some issues which make this less than straightforward:

- **Library Modules**: Application development depends upon libraries and reuse for productivity and maintainability. This poses a potentially large challenge in porting CPU applications to accelerators. Though it is possible to implement each library module on the GPU, for example, the cost of maintaining modularity is the possibly resultant overhead of GPU kernel call invocation and memory transfer between the CPU and the GPU. To achieve better performance for GPU applications, optimization sometimes requires cross-function algorithmic reorganization, or the division of a logical function into multiple kernels.
- **Synchronization**: Many Parsec applications heavily rely on fine-grained synchronization primitives, such as mutexes [5]. For some applications, like *StreamCluster*, it is relatively easy to reorganize for the GPU, while for others, it is non-trivial; especially in those applications using the software pipelining model, including *Dedup* and *Ferret*, which require significant algorithmic reorganization. The difficulty in supporting these primitives is directly attributable to the GPU's limited synchronization capabilities. On the GPU, synchronization within a thread block is provided, and global synchronization is achieved via a barrier primitive. The latest CUDA versions also provide a primitive for an on-chip, global memory fence which, unfortunately, requires restructuring of applications such that thread blocks are persistent during the entire program execution. Locks across thread blocks are non-trivial to implement, and performance benefits are not guaranteed.

Are existing application classification taxonomies sufficient to differentiate application characteristics?

Several approaches have been proposed for classifying applications based on their memory access and execution patterns, including the Berkeley *Dwarves* [1] and Intel's *Recognition, Mining and Synthesis* (RMS) [21]. Rodinia and Parsec were designed with the Dwarves and RMS as guidelines, respectively. Although these taxonomies are defined at a high level of abstraction to provide useful guiding principals and to allow users to effectively reason about program behavior, our work, often with multiple instances of a single Dwarf, suggests that the Dwarf taxonomy alone may not be sufficient to ensure adequate diversity, and that some important behaviors may not be captured by the Dwarves.

As shown in Figure 6, for *Structured Grid* applications, stencil-type workloads, such as *SRAD* and *Fluidanimate*, are quite similar. However, applications such as *HotSpot*, *Leukocyte*, and *Heartwall* are located in different clusters, with *Heartwall* significantly different from the others. *Back Propagation* and *CFD* are both from the *Unstructured Grid* Dwarf and show significant differences. The *Graph Traversal* applications, *MUMmer* and *Breadth-First Search*, are also very dissimilar.

Even applications from the same application domain are quite different; for example, the two fluid dynamics applications, Parsec's *Fluidanimate* and Rodinia's *CFD* differ more than *Fluidanimate* and *Facesim*, the latter members of different Dwarves. Also, two data mining benchmarks, *Kmeans* and *StreamCluster*, both of which rely on distance-based clustering, lie far apart in the binary clustering tree.

C. Instruction and Data Footprints

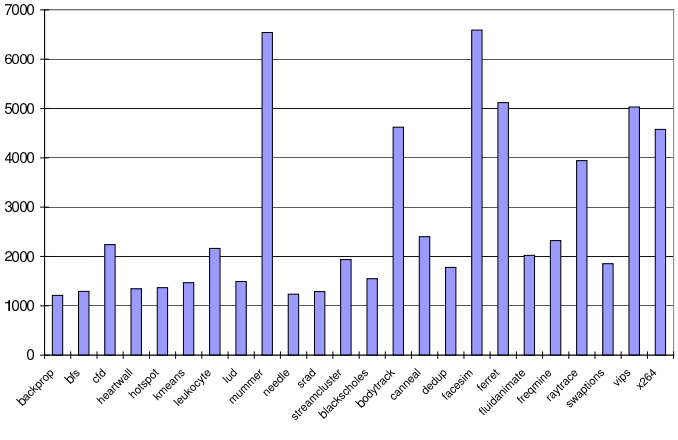

Fig. 11. The numbers of 64-byte instruction blocks touched during the program execution.

Figures 11 and 12 illustrate the instruction and data footprints of Parsec and Rodinia. The figures show the number of 64-byte instruction blocks and 4 kB data blocks touched during the entire program execution [18]. Figure 12 shows that both Parsec and Rodinia use large working sets, but, with the exception of *MUMmer*, Parsec applications tend to have larger instruction footprints, or code sizes, than Rodinia workloads.

There is a related, open question in workload characterization, that of the difference between "big" applications and "small" ones, or, in other words, between applications and kernels. Better understanding this issue requires finding the "building blocks" of the applications and a method to correlate applications with constituent kernels. Carrington *et al.* [7] did this in the HPC domain, but some more sophisticated approaches are needed for higher prediction accuracy.

VI. Related Work

The Parsec benchmark suite [5] includes emerging applications from finance, multimedia, and data mining. Parsec benchmarks utilize relatively large working sets and are developed with state-of-the-art software techniques such as software pipelining. Some earlier benchmark suites include SPLASH-2 [35] and SPEC OMP2001 [29], consisting of general-purpose workloads focusing on science, engineering, and graphics. BioParallel [17], ALPBench [20], and MineBench [27] target specific application domains.

Parboil [30] and SHOC [12] are two efforts to benchmark GPUs, but the former does not provide any diversity analysis and the latter targets systems with multiple GPU nodes. Bakhoda *et al.* developed GPGPU-Sim [2] and use it to analyze various CUDA programs. Hong *et al.* developed an analytical model to predict GPU performance [14] and proposed metrics to represent degree of warp-level parallelism. Kerr *et al.* proposed a set of metrics for GPU workloads [19] and use these metrics to analyze the behavior of GPU programs. They also use these metrics combined with PCA and regression modeling to predict GPU performance. Rodinia is distinct from these works primarily in that it is designed to provide implementations with diverse parallel execution patterns, optimizations, and software mappings, in addition to its ability to compare platforms, a crucial capability for tackling the design challenges of future parallel and heterogeneous systems.

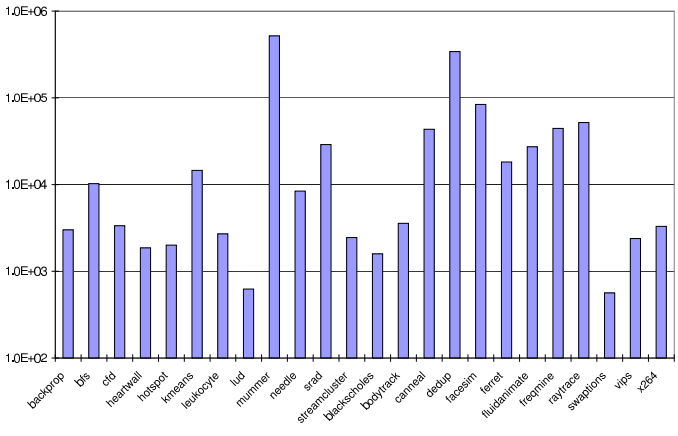

Fig. 12. The number of 4 kB data blocks touched during the program execution.

VII. Conclusions and Future Work

In this paper, we performed a detailed characterization of Rodinia, designed to let researchers better understand this collection of benchmarks and to show that the new applications—*Heartwall*, *CFD*, *LUD*, and *Mummer*—enrich the diversity of Rodinia.

We also compared Rodinia with Parsec. Some important differences we observe show the importance of measuring how well existing suites span the design space and the importance of using applications from different suites together.

Our experimental results show that Rodinia applications demonstrate a good mixture of diversity in terms of both basic program characteristics and how they stress accelerator architectures. Our applications demonstrate multiple degrees of data parallelism, branch divergence, and sensitivity to memory constraints.

Directions for future work include:

- conducting more detailed characterizations on the Rodinia GPU implementations, such as branch divergence sensitivity, data sharing among threads, and the impact of hardware thread scheduling mechanisms.
- identifying a set of metrics to quantify the extent to which the same algorithm exhibits different properties when implemented on different architectures.
- performing an application-space coverage study of existing multithreaded workloads.
- correlating program characteristics across the CPU and the GPU, as well as across big applications and small kernels.
- adding new features to the suite, including support for OpenCL and simultaneous kernel execution.

Acknowledgements

This work is supported by NSF grant nos. IIS-0612049, CNS-0916908 and CNS-0615277, a grant from the SRC under task no. 1607, and grants from NVIDIA Research and NEC labs. We would like to acknowledge George Mason University who allowed us to use their CFD application, and the University of Maryland who contributed their MUMmerGPU implementation.

References

[1] K. Asanovic et al. The landscape of parallel computing research: A view from Berkeley. Technical Report UCB/EECS-2006-183, EECS Department, University of California, Berkeley, Dec 2006.

[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] M. Bhadauria, V. M. Weaver, and S. A. McKee. Understanding PARSEC performance on contemporary CMPs. In *Proceedings of the IEEE International Symposium on Workload Characterization*, Oct 2009.

[4] C. Bienia, S. Kumar, and K. Li. PARSEC vs. SPLASH-2: A quantitative comparison of two multithreaded benchmark suites on chip-multiprocessors. In *Proceedings of the IEEE International Symposium on Workload Characterization*, Sep 2008.

[5] 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.

[6] M. Boyer, D. Tarjan, S. T. Acton, and K. Skadron. Accelerating Leukocyte tracking using CUDA: A case study in leveraging manycore coprocessors. In *Proceedings of the 23rd International Parallel and Distributed Processing Symposium*, May 2009.

[7] L. Carrington, M. Laurenzano, A. Snavely, 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.

[8] 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.

[9] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, and K. Skadron. A performance study of general purpose applications on graphics processors using CUDA. *Journal of Parallel and Distributed Computing*, 68(10):1370–1380, 2008.

[10] S. Che, J. Li, J. W. Sheaffer, K. Skadron, and J. Lach. Accelerating compute intensive applications with GPUs and FPGAs. In *Proceedings of the 6th IEEE Symposium on Application Specific Processors*, June 2008.

[11] Andrew Corrigan, Fernando Camelli, Rainald Löhner, and John Wallin. Running unstructured grid cfd solvers on modern graphics hardware. In *19th AIAA Computational Fluid Dynamics Conference*, June 2009.

[12] 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.

[13] NVIDIA CUDA Programming Guide. Web resource. http://developer.nvidia.com/object/gpucomputing.html.

[14] S.P Hong and H.S 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. De Bosschere. Performance prediction based on inherent program similarity. In *Proceedings of the 15th International Conference on Parallel Architectures and Compilation Techniques*, Sept 2006.

[17] A. Jaleel, M. Mattina, and B. Jacob. Last level cache (LLC) performance of data mining workloads on a CMP - a case study of parallel bioinformatics workloads. In *Proceedings of the 12th International Symposium on High-Performance Computer Architecture*, Feb 2006.

[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. Diamos, and S. Yalamanchili. A characterization and analysis of PTX kernels. In *Proceedings of the 2009 International Symposium on Workload Characterization*, Oct 2009.

[20] M. Li, R. Sasanka, S. V. Adve, Y. Chen, and E. Debes. The ALPBench benchmark suite for complex multimedia applications. In *Proceedings of the 2005 IEEE International Symposium on Workload Characterization*, Oct 2005.

[21] B. Liang and P. Dubey. Recognition, mining and synthesis moves computers to the era of Tera. *Technology@Intel Magazine*, Feb 2005.

[22] E. Lindholm, J. Nickolls, S. Oberman, and J. Montrym. NVIDIA Tesla: A unified graphics and computing architecture. *IEEE Micro*, 28(2):39–55, 2008.

[23] C. Luk, R. Cohn, R. Muth, H. Patil, A. Klauser, G. Lowney, S. Wallace, V. Janapa, and K. Hazelwood. Pin: Building customized program analysis tools with dynamic instrumentation. In *Proceedings of the 2005 ACM SIGPLAN Conference on Programming Language Design and Implementation*, June 2005.

[24] J. Meng and K. Skadron. Performance modeling and automatic ghost zone optimization for iterative stencil loops on GPUs. In *Proceedings of the 23rd Annual ACM International Conference on Supercomputing*, June 2009.

[25] J. Nickolls, I. Buck, M. Garland, and K. Skadron. Scalable parallel programming with CUDA. *ACM Queue*, 6(2):40–53, 2008.

[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] J. Pisharath, Y. Liu, W. Liao, A. Choudhary, G. Memik, and J. Parhi. NU-MineBench 2.0. Technical Report CUCIS-2005-08-01, Department of Electrical and Computer Engineering, Northwestern University, Aug 2005.

[28] M. C. Schatz, C. Trapnell, A. L. Delcher, and A. Varshney. High-throughput sequence alignment using graphics processing units. *BMC Bioinformatics*, 8(1):474, 2007.

[29] The Standard Performance Evaluation Corporation (SPEC). Web resource. http://www.spec.org.

[30] Parboil Benchmark suite. Web resource. http://impact.crhc.illinois.edu/parboil.php.

[31] L. G. Szafaryn, K. Skadron, and J. J. Saucerman. Experiences accelerating MATLAB systems biology applications. In *Proceedings of the Workshop on Biomedicine in Computing: Systems, Architectures, and Circuits*, June 2009.

[32] MATLAB Statistics Toolbox. Web resource. http://www.mathworks.com.

[33] E. Ukkonen. On-line construction of suffix trees, 1995.

[34] N. B. Williams, C. Fensch, and S. Moore. A communication characterization of SPLASH-2 and PARSEC. In *Proceedings of the IEEE International Symposium on Workload Characterization*, Oct 2009.

[35] 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.

[36] J. J. Yi, D. J. Lilja, and D. M. Hawkins. A statistically rigorous approach for improving simulation methodology. In *Proceedings of the Ninth International Symposium on High Performance Computer Architecture*, Feb 2002.