Exploring GPGPU Workloads: Characterization Methodology, Analysis and Microarchitecture Evaluation Implications

Nilanjan Goswami, Ramkumar Shankar, Madhura Joshi, and Tao Li
Intelligent Design of Efficient Architecture Lab (IDEAL)
University of Florida, Gainesville, Florida, USA
{nil, sramkumar, mjoshi}@ufl.edu, taoli@ece.ufl.edu

***Abstract*— The GPUs are emerging as a general-purpose high-performance computing device. Growing GPGPU research has made numerous GPGPU workloads available. However, a systematic approach to characterize these benchmarks and analyze their implication on GPU microarchitecture design evaluation is still lacking. In this research, we propose a set of microarchitecture agnostic GPGPU workload characteristics to represent them in a microarchitecture independent space. Correlated dimensionality reduction process and clustering analysis are used to understand these workloads. In addition, we propose a set of evaluation metrics to accurately evaluate the GPGPU design space. With growing number of GPGPU workloads, this approach of analysis provides meaningful, accurate and thorough simulation for a proposed GPU architecture design choice. Architects also benefit by choosing a set of workloads to stress their intended functional block of the GPU microarchitecture. We present a diversity analysis of GPU benchmark suites such as *Nvidia CUDA SDK*, *Parboil* and *Rodinia*. Our results show that with a large number of diverse kernels, workloads such as S*imilarity Score*, *Parallel Reduction*, and *Scan of Large Arrays* show diverse characteristics in different workload spaces. We have also explored diversity in different workload subspaces (e.g. memory coalescing and branch divergence). *Similarity Score*, *Scan of Large Arrays*, *MUMmerGPU*, *Hybrid Sort,* and *Nearest Neighbor* workloads exhibit relatively large variation in branch divergence characteristics compared to others. Memory coalescing behavior is diverse in *Scan of Large Arrays*, *K-Means*, *Similarity Score* and *Parallel Reduction*.**

I. INTRODUCTION

With the increasing numbers of cores per CPU chip, the performance of microprocessors has increased tremendously over the past few years. However, data-level parallelism is still not well exploited by general-purpose chip multiprocessors for a given chip area and power budget. With hundreds of in-order cores per chip, GPU provides performance throughput on data parallel and computation intensive applications. Therefore, a heterogeneous microarchitecture, consisting of chip multiprocessors and GPUs seems to be good choice for data parallel algorithms. Nvidia CUDA™ [19], AMD stream™ [23] and OpenCL [49] programming abstractions have provided data parallel application development thrust by reducing significant amount of development effort.

Emerging CPU-GPU heterogeneous multi-core processing has motivated the computer architecture research community to study various microarchitectural designs, optimizations, and analysis for GPU, such as, an efficient GPU on-chip interconnect arbitration scheme [1], more efficient GPU SIMD branch execution mechanisms [2], technique to diverge on a memory miss to better tolerate memory latencies in SIMD cores [4] etc. However, it is largely unknown whether the currently available GPGPU workloads are capable of evaluating the whole design space. An ideal evaluation mechanism must be *accurate*, *thorough* and *realistic*. *Accuracy* is provided by the applicability of the deduced conclusions that are minimally affected by the chosen benchmarks. A *thorough* evaluation mechanism covers a large amount of diverse benchmarks, where each workload stresses different aspects of the design. *Realistic* evaluation guarantees that for lesser number of simulations *thoroughness* can be achieved. *Realistic* evaluation narrows down the workload simulation space (time as well), while keeping the simulation fidelity and actual conclusion within an acceptable threshold.

In order to achieve the above goals, we propose a set of GPU microarchitecture agnostic GPGPU workload characteristics to accurately capture workload behavior and use a wide range of metrics to evaluate the effectiveness of our characterization. We employ principal component analysis [7] and clustering analysis methods [8], which have been shown [9, 10, 11, 12] to be effective in analyzing benchmark suites such as SPEC CPU2000 [13], SPEC CPU2006 [51], MediaBench [14], MiBench [15], SPLASH-2 [16], STAMP [17] and PARSEC [18] benchmarks.

This paper makes the following contributions:

- We propose a set of GPGPU workload characterization metrics. Using 38×6 design points, we show that these metrics are independent of the underlying GPU microarchitecture. These metrics will allow GPGPU researchers to evaluate the performance of emerging GPU microarchitectures regardless of their microarchitectural improvements. Though, we characterize the GPGPU workloads using Nvidia GPU microarchitecture [19], the conclusions drawn here are mostly applicable to other GPU microarchitectures such as AMD ATI [23].
- Using the proposed GPGPU workload metrics, we study the similarities between existing GPGPU kernels and observe that they often stress the same bottlenecks. We show that removing redundancy can significantly save simulation time.
- We provide workload categorization based on various workload subspaces like, divergence characteristics, kernel characteristics, memory coalescing etc. We categorize different workload characteristics according to their importance. We also show that available workload space is most diverse in terms of branch divergence characteristics and least diverse in terms of thread-batch level coalescing behavior. Relative diversity among *Nvidia CUDA SDK* [28], *Parboil* [26] and *Rodinia* [25] benchmark suites is also explored.

The rest of this paper is organized as follow. Section II provides the background of GPU microarchitecture along with CUDA [19] programming model. Section III describes the proposed GPU kernel characterization metrics as well as the statistical methods used for data analysis. Section IV describes our experimental methodologies including simulation framework, benchmarks and evaluation metrics. Section V presents our experimental results. Section VI highlights the related research. Section VII concludes the paper.

II. An Overview of GPU Microarchitecture and Programming Model

Fig.1. shows the general-purpose GPU microarchitecture where unified shader cores work as in-order multi-core streaming processor [19]. In each streaming multiprocessor (SM), there are several in-order cores, each of which is known as single processor (SP). The number of SPs per SM varies with the generations of GPU. Each SM consists of an instruction cache, a read-only constant cache, a shared memory, registers, a multi-threaded instruction fetch and issue unit (MT unit), several load/store and special functional units [21, 3]. Two or more SMs are grouped together to share a single texture processor unit and L1 texture cache among the SMs [21]. An on-chip interconnection network connects different SMs to L2 cache and memory controllers. Note that, in addition to CPU main memory, the GPU device has its own external memory (off-chip), which is connected to the on-chip memory controllers. The host CPU and GPU device communicate with each other through the PCI express bus. Some high-end GPUs also have a L2 cache, configurable shared memory, L1 data cache and error-checking-correction unit (ECC) for memory.

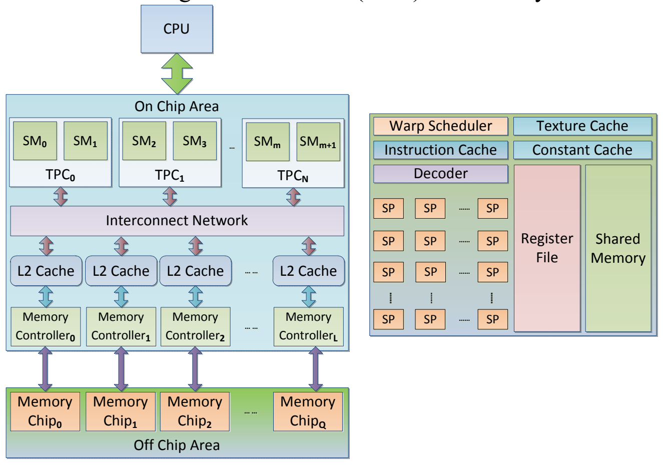

Fig. 1. GPU Microarchitecture (left), Streaming Multiprocessor (right)

General-purpose programming abstractions, such as Nvidia CUDA and AMD Stream, are often used to facilitate GPGPU application development. In this paper, we have used Nvidia CUDA programming model but some of the basic constructs will hold for most programming models. Using CUDA, thousands of parallel and concurrent lightweight threads can be launched by the host CPU and those are grouped together in an execution entity called *blocks*. Different *blocks* are grouped into an entity called a *grid*. During execution, a particular *block* is assigned to a given SM. *Blocks* running on different SMs cannot communicate with each other. Based on the available resources in a SM, one or more *blocks* can be assigned to the same SM. All the registers of the SM are allocated to different threads of different *blocks*. This leads to faster context switching among the executing threads.

Usually 32 threads from a *block* are grouped into a warp that executes the same instruction. However, this number can vary depending upon the GPU generation. Threads in GPU execute based on the single-instruction-multiple-thread (SIMT) [20] model. The execution of branch instructions may cause some threads to jump, while others fall through in a warp. This phenomenon is called warp divergence. Threads in a diverged warp execute in serial fashion. Diverged warps possess inactive thread lanes. This reduces the thread activity of the kernel below 100%. Kernels with less divergent warps work efficiently for both independent scalar threads as well as data parallel coordinated threads in SIMT mode. Load/store requests issued by different threads in a warp get coalesced in load/store unit into a single memory request according to their access pattern. Memory coalescing improves the performance by hiding the individual smaller memory accesses in a single large memory access. This phenomenon is termed as *intra-warp memory coalescing*. Threads in a *block* execute in synchronized fashion.

III. Understanding GPGPU Kernels: Characteristics and Methodology

A. GPGPU Kernel Characteristics

Emerging GPGPU benchmarks are not well understood; because these workloads are significantly different from conventional parallel applications that utilize control-dominated heavy weight threads. GPGPU benchmarks are inherently data parallel with light weight kernels.

Until now, the effects of different workload characterization parameters have not been explored to understand the fundamentals of GPGPU benchmarks. In this paper, we propose and characterize the GPGPU workload characteristics according to microarchitectural viewpoint. Table I lists the features that specify the GPGPU specific workload characteristics. Table II lists the generic workload behaviors. Index in Table I refers to the dimensions of the data in the input matrix (129×38).

To capture overall workload stress on the underlying microarchitecture, we have used *dynamic instruction count*, which is the total number of instructions executed on a GPU inclusive of all streaming multiprocessors. The *dynamic instruction count per kernel* provides per-kernel instruction count of the workload. *Average instruction count per thread* captures average stress applied to each streaming processors during kernel execution. To express the degree of parallelism we proposed *thread count per kernel* and *total thread count* metrics. Above mentioned metrics represent the *kernel stress subspace*.

Instruction mix is captured using *floating-point instruction count*, *integer instruction count*, *special instruction count,* and *memory instruction count* metrics. These parameters provide an insight of the usage of different functional blocks in the GPU. Inherent capability of hiding memory access latency is encapsulated within *arithmetic intensity*. Memory type based classification is captured using *memory instruction count*, *shared memory instruction count*, *texture memory instruction count*, *constant memory instruction count*, *local memory instruction count,* and *global memory instruction count*. *Local memory instruction count* and *global memory instruction count* encapsulate the information of off-chip DRAM access frequency that radically improves or degrades the performance of the workload. *Branch instruction count* provides the control flow behavior. Due to warp divergence, the frequency of branch instructions plays a significant role in characterizing the benchmarks. *Barrier instruction count* and *atomic instruction count* show synchronization and mutual exclusion behavior of the workload, respectively. Branch and divergence behaviors are characterized by *percentage of divergent branches*, *number of divergent warps*, *percentage of divergent branches in serialized section,* and *length of the serialized section*. Branches due to loop constructs in a workload or thread-level separate execution paths do not contribute to warp divergence. Number of threads in the taken path or fall-through path defines the length of the serialized section. *Number of divergent warps* is defined as total number of divergent warps during the execution of the benchmark. It expresses the amount of SIMD lane inactivity in terms of idle SPs in SM. *Length of serialized section* is the number of instructions executed between a divergence point and corresponding convergence point.

TABLE I
GPGPU WORKLOAD CHARACTERISTICS

Index	Characteristics	Synopsys
1	Special instruction count (I_{sop})	Total number of special functional unit instructions executed.
2	Parameter memory instruction count (I_{Par})	Total number of parameter memory instructions.
3	Shared memory instruction count (I_{shd})	Total number of shared memory instructions.
4	Texture memory instruction count (I_{tex})	Total number of texture memory instructions.
5	Constant memory instruction count (I_{const})	Total number of constant memory instructions.
6	Local memory instruction count (I_{loc})	Total number of local memory instructions.
7	Global memory instruction count (I_{glo})	Total number of global memory instructions.
8 – 17	Percentage of divergent branches (D_{bra})	Ratio of divergent branches to total branches.
18 – 27	Number of divergent warps (D_{wrp})	Total number of divergent warps in a benchmark.
28 – 37	Percentage of divergent branches in serialized section (B_{ser})	Ratio of divergent branches to total branches inside a serialized section.
38 – 47	Length of serialized section (I_{serial})	Instructions executed between a divergence and a convergence point.
48 – 57	Thread count per kernel (T_{kernel})	Count of total threads spawned per kernel.
58	Total thread count (T_{total})	Count of total threads spawned in a workload.
59 – 63	Registers used per thread (Reg)	Total number of registers used per thread.
64 – 68	Shared memory used per thread (Sh_{mem})	Total amount of shared memory used per thread.
69 – 73	Constant memory used per thread (c_{mem})	Total amount of constant memory used per thread.
74 – 78	Local memory used per thread(l_{mem})	Total amount of local memory used per thread.
79	Bytes transferred from host to device (H2D)	Bytes transferred from host to device using *cudaMemcpy*() API.
80	Bytes transferred from device to host(D2H)	Bytes transferred from device to host using *cudaMemcpy*() API.
81	Kernel count (K_n)	Number of kernel calls in the workload.
82 – 91	Row locality (R)	Average number of accesses to the same row in DRAM.
92 – 101	Merge miss (M_m) [6]	Cache misses/uncached accesses can be merged with ongoing request.

TABLE II
GENERIC WORKLOAD CHARACTERISTICS

Index	Characteristics	Synopsys
102	Dynamic instruction count (I_{total})	Dynamic instruction count for all the kernels in a workload.
103 – 112	Dynamic instruction count per kernel (I_{kernel})	Per kernel split up of the dynamic instructions executed in a workload.
113 – 122	Average instruction count per thread (I_{avg})	Average number of dynamic instructions executed by a thread.
123	Floating point instruction count (I_{fop})	Total number of floating point instructions executed in a given workload.
124	Integer instruction count (I_{op})	Total number of integer instructions executed in a given workload.
125	Memory instruction count (I_{mop})	Total number of memory instructions.
126	Branch instruction count (I_b)	Total number of branch instructions.
127	Barrier instruction count (I_{bar})	Total number of barrier synchronization instructions.
128	Atomic instruction count(I_{ato})	Total number of atomic instructions.
129	Arithmetic Intensity (A_i)	Arithmetic and logical operations per memory operation across all kernels.

Thread level resource utilization information is retrieved using metrics such as *registers per thread*, *shared memory used per thread*, *constant memory used per thread* and *local memory used per thread*. For example, in a SM registers are allocated to a thread from a pool of registers; minimum granularity of thread allocation is a *block* of threads. Too many threads per *block* may exhaust the register pool. The same holds true for shared memory. Thread-level local arrays are allocated into off-chip memory arrays; designated by *local memory used per thread*. CPU-GPU communication information is gathered from *bytes transferred from host to device* and *bytes transferred from device to host*. The former provides a notion of off-chip memory usage in GPU. The scenario when the data in the GPU is computed by itself without using any input data from CPU is distinct from the scenario when GPU processes the data received from CPU and returns it back to CPU. *Kernel count* tells the count of different parallel instruction sequences present in the workload. These metrics provide *kernel characteristics subspace*.

In order to capture intra-warp and inter-warp memory access coalescing, we use *row locality* [1] and *merge miss* [6] as described in Table I. Merge miss reveals number of cache misses or uncached accesses that can be merged into another in-flight memory request [6]. These accesses can be coalesced into a single memory access and memory bandwidth can be improved. DRAM row access locality in the data access pattern is captured before it is shuffled by the memory controller. We assume GDDR3 memory in this study. With different generations of GDDR memory these numbers may slightly differ, but it still remains microarchitecture agnostic. These two metrics comprise the *coalescing characteristics subspace*.

Some of the previously mentioned parameters are correlated with each other. For example, if a workload exhibits large integer instruction count, it will also show large dynamic instruction count. If *percentage of divergent branches in a serialized section* is high then it is probable that the *length of serialized section* will also be high. This nature of high data correlation justifies the use of PCA analysis.

B. Statistical Methods

A brief description of *principal component analysis* and *hierarchical clustering analysis* is provided here.

B.1 Principal Component Analysis

Principal component analysis (PCA) is a multivariate data analysis technique to remove the correlations among the different input dimensions and to significantly reduce the data dimensions. In PCA analysis, a matrix is formed using different columns representing different input dimensions and different rows representing different observations. PCA transforms these observations into principal component (PC) domain, where PCs are linear combination of input dimensions. Mathematically, a dataset of n correlated variables has k^{th} observation of the form, $D_k \equiv (d_{0k}, d_{1k}, d_{2k}, d_{3k}, \ldots, d_{(n-1)k})$. The k^{th} observation in PC domain is represented as $P_k \equiv (p_{0k}, p_{1k}, p_{2k}, p_{3k}, \ldots, p_{(n-1)k})$ where $p_{0k} = c_0 d_{0k} + c_1 d_{1k} + c_2 d_{2k} + c_3 d_{3k} + \ldots + c_{(n-1)} d_{(n-1)k}$. Terms like $c_0, c_{1,\ldots,} c_{n-1}$ are referred to as *factor loading* and those are selected in PCA process to maximize the variance for a particular PC. Variance of i^{th} PC (σ^2_i) has the property of $\sigma^2_{i-1} < \sigma^2_i < \sigma^2_{i+1}$. PCs are arranged in the decreasing order of eigen values. Eigen values describe the amount of information present in a principal component. *Kaiser criteria* [46] suggests considering all the PCs that have eigen value greater than 1. In general, certain numbers of first principal components are chosen to obtain 90% of the total variance as compared to original data variance. Usually the count is much smaller than the input dimensions. Hence, dimensionality reduction is achieved without losing much information.

B.2 Hierarchical Clustering Analysis

Hierarchical clustering analysis is a type of statistical data classification technique based on some kind of perceived similarity defined in the dataset. Clustering techniques [8] can be broadly categorized into *hierarchical* and *partitional* clustering. In hierarchical clustering, clusters can be chosen according to the linkage distance without defining the number of clusters *apriori*. In contrast, *partitional* clustering requires that the number of clusters be defined as an input to the process. Hierarchical clustering can be done in *agglomerative* (bottom-up) or *divisive* (top-down) manner. In the bottom-up approach, all of the points are defined as different clusters at the beginning. The most similar clusters are found and grouped together. The previous steps are repeated until all the data points are clustered. In *single linkage clustering*, similarity checking can be done by considering minimum distance among two different clusters. Alternatively, in *complete linkage clustering,* maximum distance among the two different clusters is chosen. *Average linkage clustering* refers to similarity checking based on mean distance between different clusters. The whole process of clustering produces a tree-like structure (dendrogram), where one axis represents different data points and the other represents linkage distance. The whole dataset can be categorized by selecting a particular *linkage distance*. Higher linkage distance expresses dissimilarity between the data points.

IV. Experimental Methodology

A. GPGPU-Sim and Simulator Configurations

In this study, we used GPGPU-Sim [6], a cycle accurate PTX-ISA simulator. GPGPU-Sim encapsulates a PTX ISA [5] functional simulator (CUDA-Sim) unit and a parallel SIMD execution performance simulator (GPU-Sim) unit. The simulator simulates shader cores, interconnection network, memory controllers, texture caches, constant caches, L1 caches, and off-chip DRAM. In the baseline configuration, different streaming multiprocessors (SM) are connected to the memory controllers through the mesh or crossbar interconnection network. On-chip memory controllers are connected to the off-chip memory chips. A thread scheduler distributes blocks among the shader cores in breadth-first manner; the least used SM is assigned the new block to achieve load balancing among cores [6]. The SIMD execution pipeline in the shader core has following stages: fetch, decode, pre-execute, execute, pre-memory (optional), memory access (texture, constant and other) and write-back. GPGPU-Sim provides L1 global and local memory caches. In the simulator compilation flow, *cudafe* [22] separates GPU code, which is further compiled by *nvopencc* [22] to produce PTX assembly. *Ptxas* assembler generates thread level shared memory, register and constant memory usage that is used by GPGPU-Sim during thread allocation in individual SM. GPGPU-Sim uses the actual PTX assembly instructions generated for functional simulation. Moreover, GPGPU-Sim implements a custom CUDA runtime library to divert the CUDA runtime API calls to GPGPU-Sim. Host C code is linked with the GPGPU-Sim custom runtime.

TABLE III
GPGPU-Sim Shader Configuration

	Conf 1	Conf 2	Base	Conf 3	Conf 4	Conf 5
Shader cores	8	8	28	64	110	110
Warp size	32	32	32	32	32	32
SIMD Width	32	32	32	32	32	32
DRAM controllers	8	8	8	8	11	11
DRAM queue size	32	32	32	32	32	128
Blocks / SM	8	2	8	8	8	16
Bus width	8bytes/cycle					
Const. / Texture cache	8KB / 64B (2-way, 64B line, LRU)					
Shared memory	16KB	8KB	16KB	16KB	16KB	32KB
Reg. count	8192	8192	16384	16384	16384	32768
Threads per SM	1024	512	1024	1024	1024	2048

TABLE IV
GPGPU-Sim Interconnect Configuration

	Conf 1	Conf 2	Base	Conf 3	Conf 4	Conf 5
Topology	Mesh	Mesh	Mesh	Crossbar	Mesh	Mesh
Routing	Dim. order	Dim. order	Dim. order	Dim. order	Dim. order	Dim. order
Virtual channels	2	2	2	1	4	4
VC buffers	4	4	4	8	16	16
Flit size	16 B	8 B	8 B	32 B	32 B	64 B

We have used the GPGPU-Sim configuration as specified in Tables III and IV. To show that the GPU configuration has no impact on the workload data, we have used 6 different configurations. Conf-2, baseline configuration, and Conf-5 are significantly different from each other, whereas other configurations are changed in a subtle manner to see the effect of few selected components. Additionally, we have heavily instrumented the simulator to extract the GPU characteristics such as floating-point instruction count, arithmetic intensity, percentage of divergent branches, percentage of divergent warp, percentage of divergent branches in serialized section, length of serialized section, etc.

TABLE V
GPGPU WORKLOAD SYNOPSIS

Abbr.	Workload	Instr. Count	Arith. Intensity	Branch Div? / Merge Miss? / Shared Mem? / Barriers?
LV	Levenshtein Edit-distance calculation	32K	221.5	Y/Y/Y/N
BFS	Breadth First Search [29]	589K	99.7	Y/Y/Y/N
BP	Back Propagation [25]	1048K	167.6	Y/Y/Y/Y
BS	BlackScholes Option Pricing [28]	61K	1000	Y/Y/Y/N
CP	Columbic Potential [26]	8K	471.8	Y/Y/Y/N
CS	Separable Convolution [28]	10K	3.1	Y/Y/Y/Y
FWT	Fast Walsh Transform [28]	32K	289.1	Y/Y/Y/Y
GS	Gaussian Elimination [25]	921K	5.7	Y/Y/Y/Y
HS	Hot Spot [31]	432K	101.4	Y/Y/Y/Y
LIB	LIBOR [32]	8K	353.6	Y/Y/Y/N
LPS	3D Laplace Solver [30]	12K	118.9	Y/Y/Y/Y
MM	Matrix Multiplication [28]	10K	42.4	Y/Y/Y/Y
MT	Matrix Transpose [28]	65K	195.6	Y/Y/Y/Y
NN	Neural Network [34]	66K	71.5	Y/Y/Y/N
NQU	N-Queen Solver [35]	24K	209.3	Y/Y/Y/Y
NW	Needleman Wunsch [25]	64	80.1	Y/Y/Y/Y
PF	Path Finder [25]	1K	378.1	Y/Y/Y/Y
PNS	Petri Net Simulation [26]	2.5K	411.5	Y/Y/Y/Y
PR	Parallel Reduction [28]	830K	1.0	Y/Y/Y/Y
RAY	Ray Trace [36]	65K	335.0	Y/Y/Y/Y
SAD	Sum of Absolute Difference [26]	11K	67.7	Y/Y/Y/Y
SP	Scalar Product [28]	32K	284.9	Y/Y/Y/Y
SRAD	Speckle Reducing Anisotropic Diffusion [26]	460K	14.0	Y/Y/Y/Y
STO	Store GPU [37]	49K	361.9	Y/Y/Y/N
CL	Cell [25]	64	765.2	Y/Y/Y/Y
HY	Hybrid Sort [38]	541K	32.1	Y/Y/Y/Y
KM	K-Means [39]	1497K	20.2	Y/Y/Y/N
MUM	MUMmerGPU [40]	50K	468.5	Y/Y/Y/N
NE	Nearest Neighbor [25]	60K	196.8	Y/Y/Y/N
BN	Binomial Options [28]	131K	39.4	Y/Y/Y/Y
MRIF	Magnetic Resonance Imaging FHD [44]	1050K	86.7	Y/Y/Y/N
MRIQ	Magnetic Resonance Imaging Q [26]	526K	107.6	Y/Y/Y/N
DG	Galerkin time-domain solver [41]	1035K	11.4	Y/Y/Y/Y
SLA	Scan of Large Arrays [28]	1310K	1.9	Y/Y/Y/Y
SS	Similarity Score [25]	51K	1.1	Y/Y/Y/Y
AES	AES encryption [42]	65K	70.25	N/Y/Y/Y
WP	Weather Prediction [43]	4.6K	459.8	Y/Y/Y/N
64H	64 bin histogram [28]	2878K	22.8	Y/Y/Y/Y

B. GPGPU Workloads

To perform GPU benchmark characterization analysis, we have collected a large set of available GPU workloads from *Nvidia CUDA SDK* [28], *Rodinia* Benchmark [25], *Parboil* Benchmark [26] and some third party applications. Due to simulation issues (e.g. simulator deadlock), we have excluded some benchmarks from our analysis. The problem size of the workloads is scaled to avoid long simulation time or unrealistically small workload stress. Table V lists the basic workload characteristics.

TABLE VI
GPGPU WORKLOAD EVALUATION METRICS

Evaluation metric	Synopsys
Activity factor	Average percentage of threads active at a given time.
SIMD parallelism	Speedup with infinite number of SP per SM.
DRAM efficiency	% of time spent sending the data across the pins of DRAM when other commands are pending /serviced.

C. Workload Evaluation Metrics

To evaluate the accuracy and the effectiveness of the proposed workload characteristics, we use the set of metrics as listed in Table VI. *Activity factor* [24] is defined as the average number of active threads at a given time during the execution phase. Several branch divergence related characteristics of the benchmark change this parameter. For example, the absence of branch divergence produces an *activity factor* of 100%. *SIMD parallelism* [24] captures the scalability of a workload. Higher value for *SIMD parallelism* indicates that the workload performance will improve on a GPU that have higher SIMD width. *DRAM efficiency* [6] describes how frequently memory accesses are requested during kernel computation. It also captures the amount of time spent to perform DRAM memory transfer during overall kernel execution. If a benchmark has large number of shared memory accesses or the benchmark has ALU operations properly balanced in between memory operations, then the metric will show higher value.

D. Experiment Stages

The experimental procedure consists of following steps:

(1) The heavily instrumented GPGPU-Sim simulator retrieves the statistics of GPU characteristics listed in Tables I, II and VI for all the configurations described in Tables III and IV. Six different configurations are simulated to demonstrate the microarchitectural independence of the GPGPU workload characteristics.

(2) We have performed vectorization of some characteristics to produce 10-bin or 5-bin histograms from the GPGPU-Sim output. This process provides an input matrix for principal component analysis of size 129×38 (dimensions×workloads). Few dimensions of the matrix with zero *standard deviation* are kept out of the analysis to produce a 93×38 normalized data matrix.

(3) PCA is performed using STATISTICA [45]. Several PCAs are performed on the whole workload set according to different workload characteristics subspaces and all the characteristics.

(4) Based on required percentage of total variance, *p* principal components are chosen.

(5) STATISTICA is used to perform hierarchical cluster analysis. Both single linkage and complete linkage based dendrograms are generated using *m* principal components, where $m < p$ (*p*: total number of PC retained).

V. RESULTS AND ANALYSIS

A. Microarchitecture Impact on GPGPU Workload Characteristics

We verify that GPGPU workload characteristics are independent of the underlying microarchitecture. If the microarchitecture has little or no impact on the set of benchmark characteristics, then benchmarks executed on different microarchitectures will be placed close to each other in uncorrelated principal component domain. Figures 2 and 3 show the dendrogram of all the benchmarks and the distribution of the variance obtained in different PCs. According to

Kaiser Criterion [46] we need to consider all the principal components till PC-20. Yet we have decided to consider first 7 principal components that retain 70% of the total variance and first 16 principal components that retain 90% of the total variance. In Figure 3 we see that executions of same benchmark on different microarchitectures are having small linkage distance, which demonstrates strong clustering. This suggests that irrespective of the varied microarchitecture configurations (e.g. shader cores, register file size, shared memory size, interconnection network configuration etc.), our proposed metrics are capable of capturing GPGPU workload characteristics. Note that, in Figure 3 some benchmarks' execution on a particular microarchitecture is missing due to lack of microarchitectural resources of the GPU or simulation issues. For example, Conf-2 (Tables III and IV) is incapable of executing benchmark-problem size pair of STO, NQU, HY and WP.

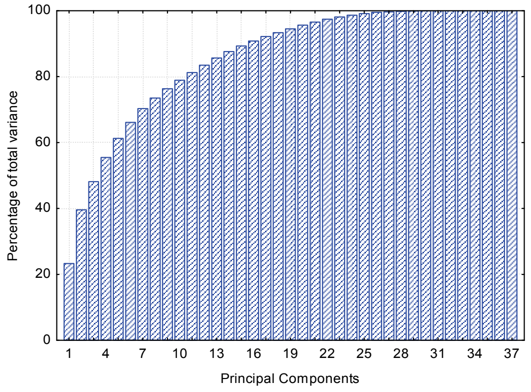

Fig. 2. Cumulative Distribution of Variance by PC

B. GPGPU Workload Classification

The program-input pairs defined as workload in Table V show significantly distinct behavior in PCA domain. The hierarchically clustered principal components are listed in Figures 4, 5, 6, and 7. To avoid the hierarchical clustering artifacts, we have done both single linkage and complete linkage clustering dendrograms [27]. We have observed that for the set of 4 and 6 workloads the single linkage and complete linkage clustering shows identical results irrespective of 70% variance and 90% variance cases. Benchmarks in set of 4 are SS, SLA, PR and NE. The set of 6 includes SS, SLA, PR, BN, KM and NE. For set size of 8 and 12 we have found single linkage and complete linkage clustering deviates by 1 and 2 workloads respectively, for the 90% variance case. Table VII depicts results; bold type face highlights the differences in single and complete linkage clustering. To represent a cluster we choose the closest benchmark to the centroid of that cluster. The results for 70% of total variance are presented in Figures 6 and 7. Dendrogram also highlights that *Rodinia*, *Parboil* and *Nvidia CUDA SDK* benchmark suites demonstrate the diversity in decreasing order according to the available workloads. Table VII shows the time savings obtained for different sets in the form of speedup. Out of all the benchmarks SS and 64H contribute 26% and 29% of the total simulation time respectively. Speedup numbers are in fact higher without SS. SS demonstrates very distinct characteristics from the rest; such as high instruction count, low arithmetic intensity, very high kernel count, diverse inter-warp coalescing behavior, branch divergence diversity, different types of kernels and diverse instruction mix. As expected, overall speedup decreases as we increase the set size. Due to the inclusion of 64H, the set of 12 in complete linkage clustering does not show high speedup. Architects can choose the set size to achieve the trade-off between available simulation time and amount of accuracy desired.

C. GPU Microarchitecture Evaluation

Figure 8 shows the performance comparison of different evaluation metrics as GPU microarchitecture varies. Table VIII shows the average error observed. Due to lack of computing resources and simulation issues with some benchmarks (e.g. SS, STO, HY, NQU), configurations such as Conf-2, Conf-4 and Conf-5 are not considered. This should not affect the results as the benchmark characteristics are microarchitecture agnostic. Maximum average error for activity factor is less than 17%. It reveals that the subsets are capable of representing branch divergence characteristics present in the whole set. Divergence based clustering shows branch divergence behavior is diverse. Therefore, a small set of 4 is relatively less accurate in capturing divergence behavior. As we increase the set size to 6 and 8, we reach the average error as close as 1%. Average error in SIMD parallelism suggests that we are capturing the kernel level parallelism through activity factor and dynamic instruction count with maximum error less than 18%. Increase of set size also decreases the average error for SIMD parallelism. DRAM efficiency shows that coalescing characteristics, memory request and memory transfer behaviors are captured closely with maximum error less than 6%. We also observe that the increase in subset size decreases the average error.

D. GPGPU Workload Fundamentals Analysis

In this subsection we analyze the workloads from the point of view of input characteristics. Architects are often interested in identifying the most influential characteristics to tune the performance of the workload on the microarchitecture. Different PCs are arranged in decreasing order of variances. Hence, by observing their factor loadings we can infer the impact of the input characteristics. We choose to analyze top 4 principal components, which account for 53% of the total variance.

Figure 9 shows the factor loadings for principal component 1 and 2, which account for 39% of total variance. Figure 10 shows the scatter plot. SS, SLA, PR, GS, SRAD, 64H, HS, and KM show relatively low arithmetic intensity than others. SS, SLA, PR, GS, SRAD, 64H, HS, and KM have large number of threads spawned, small number of instructions in a kernel, moderate merge miss count, and good row locality. The exceptions are CS and HY, which possess large kernel count. Moreover, workloads excluding these are relatively less diverse in the previously mentioned behavioral domain. Close examination of the simulation statistics verify that benchmarks except SS, SLA, PR, GS, SRAD, 64H, and KM have good arithmetic intensity, large number of kernels and kernel level diversity. There are no benchmarks with very high arithmetic intensity. SS exhibits high integer instruction count, host to device data transfer, low branch divergence, and per thread shared memory usage. In contrast, SLA demonstrates low merge miss, good row locality and branch divergence. Simulation statistics verifies observed branch divergence, merge miss, and row locality.

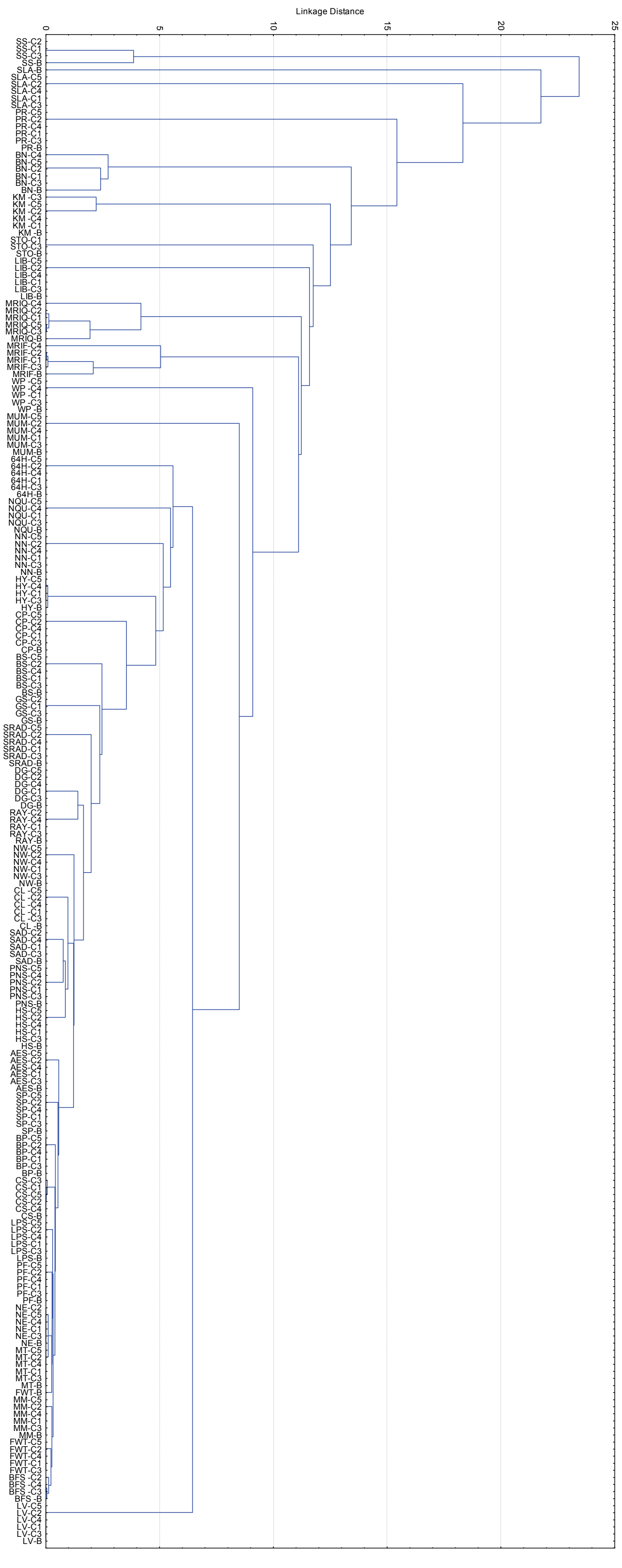

Fig. 3. Dendrogram of all Workloads

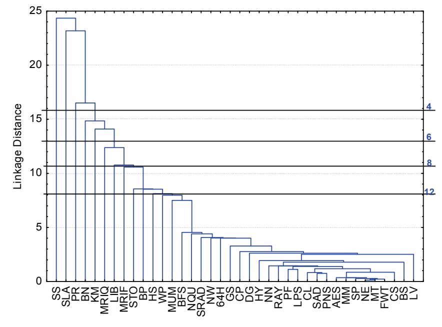

Fig. 4. Dendrogram of Workloads (SL, 90% Var)

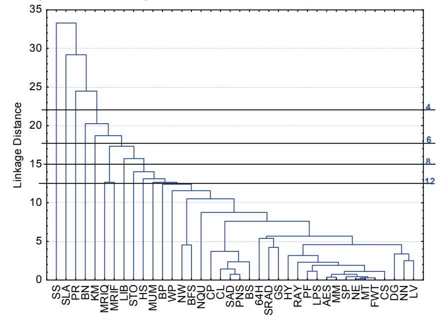

Fig. 5. Dendrogram of Workloads (CL, 90% Var)

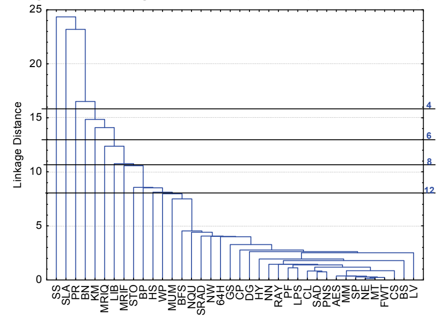

Fig. 6. Dendrogram of Workloads (SL, 70% Var)

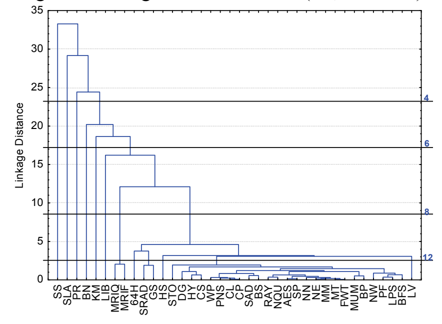

Fig. 7. Dendrogram of Workloads (CL, 70% Var)

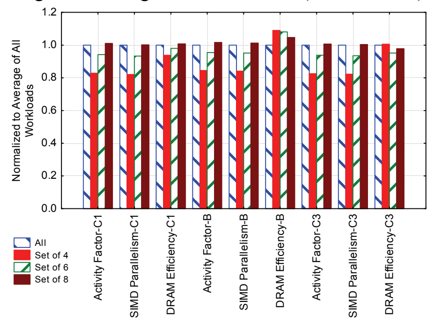

Fig. 8. Performance comparison (normalized to total workloads) of different evaluation metrics across different GPU Microarchitectures (C1: Config. 1, B: Baseline, C3: Config. 3)

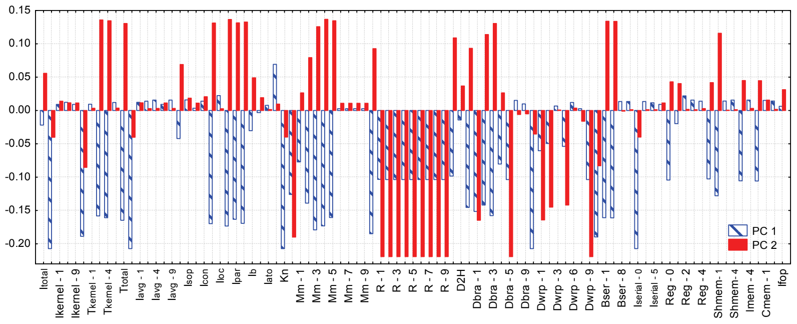

Fig. 9. Factor Loading of PC-1 and PC-2

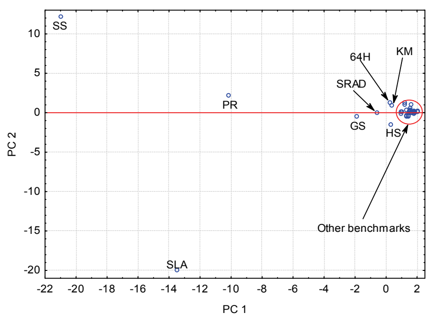

Fig. 10. PC-1 vs. PC-2 Scatter Plot

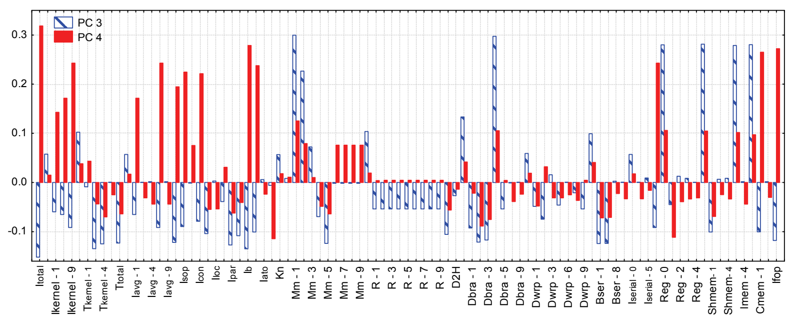

Fig. 11. Factor Loading of PC-3 and PC-4

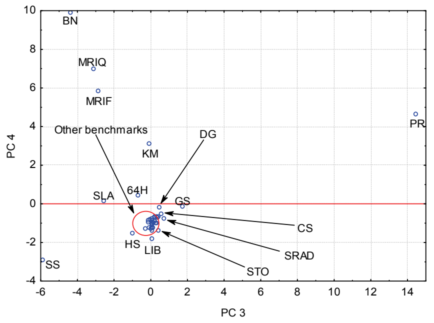

Fig. 12. PC-3 vs. PC-4 Scatter Plot

TABLE VII
GPGPU WORKLOAD SUBSET FOR 90% VARIANCE

	Single linkage (SL)	SL speedup	Complete linkage (CL)	CL speedup
Set 4	SS, SLA, PR, NE	3.60	SS, SLA, PR, NE	3.60
Set 6	SS, SLA, PR, KM, BN, NE	2.38	SS, SLA, PR, KM, BN, NE	2.38
Set 8	SS, SLA, PR, KM, BN, LIB, MRIQ, **NE**	1.80	SS, SLA, PR, KM, BN, LIB, MRIQ, **MRIF**	1.53
Set 12	SS, SLA, PR, KM, BN, LIB, MRIQ, MRIF, STO, HS, **BP**, **NE**	1.51	SS, SLA, PR, KM, BN, LIB, MRIQ, MRIF, STO, HS, **64H**, **MUM**	1.05

TABLE VIII
AVERAGE % OF ERROR AS MICROARCHITECTURE VARIES

Evaluation metric	Set 4	Set 6	Set 8
Activity Factor	16.7	5.5	1.1
SIMD parallelism	17.2	6.0	0.5
DRAM efficiency	5.2	5.0	2.5
Average	13.1	5.5	1.4

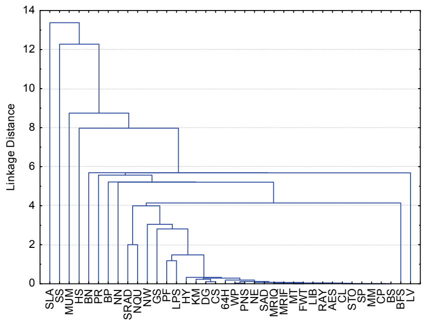

Fig. 13. Dendrogram based on Divergence Characteristics

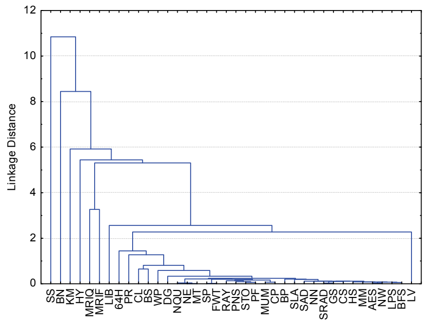

Fig. 14. Dendrogram based on Instruction Mix

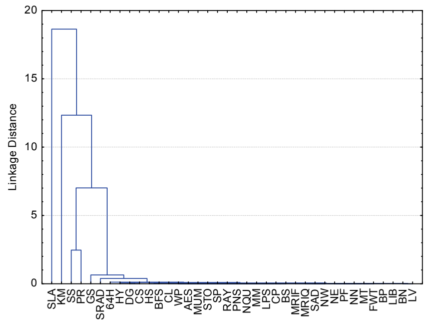

Fig. 15. Dendrogram based on Merge Miss and Row Locality

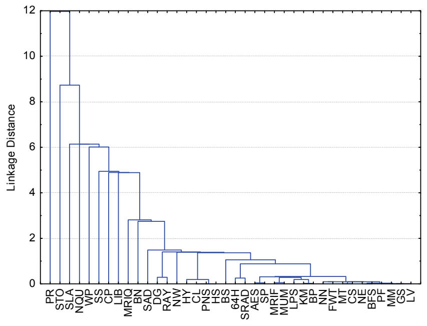

Fig. 16. Dendrogram based on Kernel Characteristics

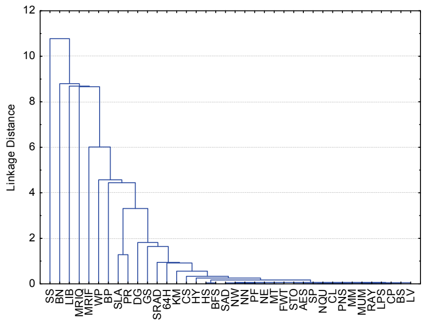

Fig. 17. Dendrogram based on Kernel Stress

Figures 11 and 12 show the PC-3 vs. PC-4 (14% of total variance) factor loadings and scatter plot. PC-3 shows that PR has large number of kernels, low merge miss and moderate branch divergence. BN, SLA, MRIF and MRIQ possess moderate branch divergence and heavyweight threads. BN, MRIQ, MRIF, KM, and PR program-input pairs have long serialized section due to branch divergence (large I_{serial}), high barrier and branch instruction count, heavy weight threads and more computations.

E. *Characteristics Based Classification*

In Section 3.1 we have mentioned several subspaces, such as instruction mix subspace, coalescing characteristics subspace, divergence characteristics subspace, to categorize the workload characteristics. In Figures 13, 14, 15, 16 and 17, we present dendrograms generated from the principal components (90% of total variance) obtained from PCA analysis of the workloads based on those different subspaces. Architects interested in improving branch divergence hardware might want to simulate SS, SLA, MUM, HS, BN and NE as they exhibit relatively large amount of branch divergence diversity. On the other hand, SS, BN, KM, MRIF, MRIQ, and, HY have distinct behavior in terms of instruction mix. These are valuable benchmarks for evaluating the effectiveness of design with ISA optimization. Interestingly, workloads excluding SLA, KM, SS and PR show similar types of inter-warp and intra-warp memory coalescing behavior. Hence, SLA, KM, SS and PR provide diversity to evaluate microarchitecture level memory access optimizations. Though it is possible that the workloads with similar inter and intra warp coalescing have large variations in cache behavior when the size of the cache is increasing beyond that used to measure intra-warp coalescing. Figure 16 shows the static kernel characteristics of different benchmarks. PR, STO, SLA, NQU, WP, SS, CP and LIB show distinct kernel characteristics. SS, BN, LIB, MRIF, MRIQ, WP, BP and NE demonstrate diverse behavior in terms of instructions executed by each thread, total thread count, total instruction count etc.

F. *Discussions and Limitations*

This research explores PTX translated GPGPU workload diversity in terms of different microarchitecture independent program characteristics. The GPU design used in our research closely resembles Nvidia GPUs. However, growing GPU industry has several other microarchitecture designs (AMD-ATI [23], Intel Larrabee [50]), programming models (ATI Stream [23], OpenCL[49]) and ISAs (ATI intermediate language, x86, Nvidia Native Device ISA, ATI Native Device ISA). The microarchitecture independent characteristics proposed in this paper are by and large applicable to ATI GPUs though they use different virtual ISA (ATI IL). For example, SFU instructions may be specific to Nvidia, but AMD-ATI also processes these instructions in a transcendental unit inside the thread processor. Branch divergence and memory coalescing behavior of the kernels are a characteristic of the workload and do not affect the results produced in this paper for ATI GPUs. However, for Intel Larrabee architecture, these characteristics will be different because of dissimilarity in the programming model. Also, Larrabee microarchitecture is different from traditional GPUs. Hence, for Larrabee the suitability of using the proposed metrics to represent workload characteristics needs further exploration.

PTX ISA changes are minor over the generations; therefore it has insignificant effect on the results. In addition, PTX instruction mapping to different generations of native Nvidia ISAs and programming model change has little impact on the GPGPU kernel characteristics because we characterize the kernels in terms of the PTX virtual ISA.

VI. Related Work

Saavedra et al. [47] demonstrated how workload performance on a new microarchitecture can be predicted using the study of microarchitecture independent and dependent workload characteristics. However, their work did not consider the correlated nature of different characteristics. In [11, 48], Eeckhout et al. demonstrated a technique to reduce the simulation time while keeping the benchmark diversity information intact by performing PCA and clustering analysis on correlated microarchitecture independent workload characteristics. [12] showed that the 26 CPU2000 workloads only stress four different bottlenecks by collecting data from 340 different machines. [9] reported the redundancy in SPEC2006 benchmarks using the same technique. Results showed that 6 CINT2006 and 8 CFP2006 benchmarks can be representative of the whole workload set. In previously mentioned works, the following characteristics of the program were widely adopted: instruction mix, branch prediction, instruction level parallelism, cache miss rates, sequential flow breaks etc. The using of microarchitecture and transactional architecture independent characteristics, such as transaction percentage, transaction size, read/write set size ratio and conflict density, for clustering transactional workloads was performed in [10]. Our approach differs from all the above as we characterize the data parallel GPGPU workloads using several newly proposed microarchitecture independent characteristics.

So far, there have been very few studies [24, 25] on GPGPU workload classification. [24] characterized 50 different PTX kernels which include *NVIDIA CUDA SDK* kernels and *Parboil* benchmarks suit. This study does not consider correlation among various workload characterization metrics. In contrast, we use a standard statistical methodology with a wide range of workload characterization metrics. Moreover, [24] have not considered diverse benchmark suites like *Rodinia* and *Mars* [33]. In [25], the *MICA framework* [11] was used to characterize single-core CPU version of the GPGPU kernels, which fails to capture branch divergence, row access locality, and memory coalescing, the characteristics of numerous parallel threads running on massively parallel GPU microarchitecture. Moreover, instruction level parallelism has less effect in GPU performance due to the simplicity of the processor core architecture. More than data stream size, data access pattern plays important role in boosting the application performance on GPU. We capture these over-looked features in our program characteristics to look into the GPGPU workload behavior in detail.

VII. Conclusions

The emerging GPGPU workload space has not been explored methodically to understand the fundamentals of the workloads. To understand the GPU workloads, we have proposed a set of microarchitecture independent GPGPU workload characteristics that are capable of capturing five important behaviors: kernel stress, kernel characteristics, divergence characteristics, instruction mix, and coalescing characteristics. We have also demonstrated that the proposed evaluation metrics accurately represents the input characteristics. This work provides GPU architect a clear understanding of the available GPGPU workload space to design better GPU microarchitecture. Our endeavor also demonstrates that workload space is not properly balanced with available benchmarks; while SS, SLA, PR workloads show significantly different behavior due to their large number of diverse kernels, the rest of the workloads provide similar characteristics. We also observe that benchmark suites like *Nvidia CUDA SDK*, *Parboil*, and *Rodinia* has different behavior in terms workload space diversity. Our research shows that branch divergence diversity is best captured by SS, SLA, MUM, HS, BN, and, NE. SLA, KM, SS, and, PR show relatively large memory coalescing behavior diversity than the rest. SS, BN, KM, MRIF, and HY have distinct instruction mix. Kernel characteristic is diverse in PR, STO, NQU, SLA, WP, SS, CP, and LIB. We also show that among the chosen benchmarks divergence characteristics are most diverse and coalescing characteristic are least diverse. Therefore, we need benchmarks with more diverse memory coalescing behavior. In addition, we show that simulation speedup of 3.5× can be achieved by removing the redundant benchmarks.

ACKNOWLEDGEMENTS

This work is supported in part by NSF grants CNS-0834288, CCF-0845721 (CAREER), SRC grant 2008-HJ-1798, and by three IBM Faculty Awards. The authors acknowledge the UF HPC Center for providing computational resources. We also acknowledge anonymous reviewers for their valuable suggestions.

REFERENCES

[1] G. Yuan, A. Bakhoda, and T. Aamodt, Complexity Effective Memory Access Scheduling for Many-Core Accelerator Architectures, MICRO, 2009.
[2] W. Fung, I. Sham, G. Yuan, and T. Aamodt, Dynamic Warp Formation and Scheduling for Efficient GPU Control Flow, MICRO, 2007.
[3] T. Halfhill, Looking Beyond Graphics, Microprocessor Report, 2009.
[4] D. Tarjan, J. Meng, and K. Skadron, Increasing Memory Miss Tolerance for SIMD Cores, SC, 2009.
[5] NVIDIA Compute PTX: Parallel Thread Execution ISA, Version 1.4, 2009.
[6] A. Bakhoda, G. Yuan, W. Fung, H. Wong, and T. Aamodt, Analyzing CUDA Workloads using a Detailed GPU Simulator, ISPASS, 2009.
[7] G. Dunteman, Principal Components Analysis, SAGE Publications, 1989.
[8] C. Romesburg, Cluster Analysis for Researchers, Lifetime Learning Publications, 1984.
[9] A. Joshi, A. Phansalkar, L. Eeckhout, and L. John, Measuring Benchmark Similarity using Inherent Program Characteristics, IEEE Transactions on Computers, Vol.55, No.6, 2006.
[10] C. Hughe, J. Poe, A. Qouneh, and T. Li, On the (Dis)similarity of Transactional Memory Workloads, IISWC, 2009.
[11] L. Eeckhout, H. Vandierendonck, and K. De Bosschere, Designing Computer Architecture Research Workloads, Computer, Vol. 36, No. 2, 2003.
[12] H. Vandierendonck and K. Bosschere, Many Benchmarks Stress the Same Bottlenecks, Workshop on Computer Architecture Evaluation Using Commercial Workloads, 2004.
[13] J. Henning, SPEC CPU2000: Measuring CPU Performance in the New Millennium, IEEE Computer, pp. 28-35, July 2000.
[14] C. Lee, M. Potkonjak, and W. Smith, MediaBench: A Tool for Evaluating and Synthesizing Multimedia and Communication Systems, MICRO, 1997.
[15] M. Guthaus, J. Ringenberg, D. Ernst, T. Austin, T. Mudge, and R. Brown, MiBench: A Free, Commercially Representative Embedded Benchmark Suite, Workshop on Workload Characterization, 2001.
[16] S. Woo, M. Ohara, E. Torrie, J. Singh, and A. Gupta, The SPLASH-2 Programs: Characterization and Methodological Considerations, ISCA, 1995.
[17] C. Minh, J. Chung, C. Kozyrakis, and K. Olukotun, STAMP: Stanford Transactional Memory Applications for Multi-Processing, IISWC, 2008.
[18] C. Bienia, S. Kumar, J. Singh, and K. Li, The PARSEC Benchmark Suite: Characterization and Architectural Implications, Princeton University Technical Report, 2008.
[19] NVIDIA CUDA™ Programming Guide Version 2.3.1, Nvidia Corporation, 2009.
[20] J. Nickolls, I. Buck, M. Garland, and K. Skadron, Scalable Parallel Programming with CUDA, Queue 6, 2 (Mar. 2008), 40-53.
[21] E. Lindholm, J. Nickolls, S. Oberman, and J. Montrym, NVIDIA Tesla: A Unified Graphics and Computing Architecture, Micro, vol.28, no.2, 2008.
[22] The CUDA Compiler Driver NVCC, Nvidia Corporation, 2008.
[23] Technical Overview, ATI Stream Computing, AMD Inc, 2009.
[24] A. Kerr, G. Diamos, and S. Yalamanchilli, A Characterization and Analysis of PTX Kernels, IISWC 2009.
[25] S. Che, M. Boyer, J. Meng, D. Tarjan, J. Sheaffer, S. Lee, and K. Skadron. Rodinia: A Benchmark Suite for Heterogeneous Computing, IISWC, 2009.
[26] Parboil Benchmark suite. URL: http://impact.crhc.illinois.edu/ parboil.php.
[27] M. Hughes, Exploration and Play Re-visited: A Hierarchical Analysis, International Journal of Behavioral Development, 1979.
[28] http://www.nvidia.com/object/cuda_sdks.html
[29] P. Harish and P. J. Narayanan, Accelerating Large Graph Algorithms on the GPU Using CUDA, HiPC, 2007.
[30] M. Giles, Jacobi Iteration for a Laplace Discretisation on a 3D Structured Grid, http://people.maths.ox.ac.uk/˜ gilesm/hpc/NVIDIA/laplace3d.pdf, April, 2008.
[31] W. Huang, S. Ghosh, S. Velusamy, K. Sankaranarayanan, and K. Skadron, M. R. Stan, Hotspot: A Compact Thermal Modeling Methodology for Early-Stage VLSI Design, IEEE Transactions on VLSI Systems 14 (5) (2006).
[32] M. Giles and S. Xiaoke, Notes on using the NVIDIA 8800 GTX graphics card. http://people.maths.ox.ac.uk/˜gilesm/hpc/.
[33] B. He, W. Fang, Q. Luo, N. Govindaraju, and T. Wang, Mars: a MapReduce Framework on Graphics Processors, PACT, 2008.
[34] Billconan and Kavinguy, A Neural Network on GPU. http://www.codeproject.com/KB/graphics/GPUNN.aspx.
[35] Pcchen. N-Queens Solver, http://forums.nvidia.com/index.php?showtopic=76893, 2008.
[36] Maxime. Ray tracing. http://www.nvidia.com/cuda.
[37] S. Al-Kiswany, A. Gharaibeh, E. Santos-Neto, G. Yuan, and M. Ripeanu, StoreGPU: Exploiting Graphics Processing Units to accelerate Distributed Storage Systems, HPDC, 2008.
[38] E.Sintorn and U. Assarsson, Fast Parallel GPU-sorting using a Hybrid Algorithm, Journal of Parallel and Distributed Computing, Volume 68, Issue 10, October 2008.
[39] R. Narayanan, B. Ozisikyilmaz, J. Zambreno, J. Pisharath, G. Memik, and A. Choudhary, MineBench: A Benchmark Suite for Data Mining Workloads, IISWC, 2006.
[40] M. Schatz, C. Trapnell, A. Delcher, and A. Varshney, High-throughput Sequence Alignment using Graphics Processing Units, BMC Bioinformatics, 8(1): 474, 2007.
[41] T. Warburton, Mini Discontinuous Galerkin Solvers. http://www.caam.rice.edu/~timwar/RMMC/MIDG.html, 2008.
[42] S. Manavski, CUDA compatible GPU as an Efficient Hardware Accelerator for AES Cryptography, ICSPC, 2007.
[43] J. Michalakes and M. Vachharajani, GPU Acceleration of Numerical Weather Prediction, IPDPS, 2008.
[44] S. Stone, J. Haldar, S. Tsao, W. Hwu, Z. Liang, and B. Sutton, Accelerating Advanced MRI Reconstructions on GPUs, Computing Frontiers, 2008.
[45] StatSoft, Inc. STATISTICA, http://www.statsoft.com/.
[46] K. Yeomans and P.Golder, The Guttman-Kaiser Criterion as a Predictor of the Number of Common Factors", Journal of the Royal Statistical Society. Series D (The Statistician), Vol. 31, No. 3, 1982.
[47] R. H. Saavedra and A. J. Smith, Analysis of Benchmark Characteristics and Benchmark Performance Prediction, ACM Trans. Computer Systems, 1998.
[48] K. Hoste and L. Eeckhout. Microarchitecture-independent Workload Characterization. IEEE Micro, 27(3):63–72, 2007.
[49] http://www.khronos.org/opencl/
[50] L. Seiler, D. Carmean, E. Sprangle, T.Forsyth, M. Abrash, P. Dubey, S. Junkins, A. Lake, J. Sugerman, R. Cavin, R. Espasa, E. Grochowski, T. Juan, and P. Hanrahan, Larrabee: A Many-core X86 Architecture for Visual Computing, ACM Trans. Graph. 27-3, 2008.
[51] A. Phansalkar, A. Joshi, and L. John, Analysis of Redundancy and Application Balance in the SPEC CPU2006 Benchmark Suite, ISCA, 2007.