Challenge Benchmarks That Must be Conquered to Sustain the GPU Revolution

Emily Blem Matthew Sinclair Karthikeyan Sankaralingam
University of Wisconsin - Madison
{blem, sinclair, karu}cs.wisc.edu

***Abstract*—The shift from GPUs to GPGPUs has brought with it many changes to the GPU architecture (e.g. more caches, more concurrent kernels, better synchronization). As GPUs press further into the general-purpose domain, architects must continue to address the performance of challenging workloads. This paper presents a set of challenge benchmarks and their key performance limitations to help direct future GPU architecture research. Our study shows GPUs must develop multiple innovative architectural techniques to efficiently execute these applications to continue making inroads into general purpose computing.**

I. Introduction

GPU architectures have become increasingly general purpose as manufacturers successfully expand into the high throughput computing market. The introduction of CUDA and OpenCL for programmability coupled with architecture changes including more processors, increased caching, and improved pipelines have all allowed the GPU to support more diverse applications. However, current GPUs struggle when executing programs where SIMD parallelism is not abundant. In order for GPUs to become a truly general purpose architecture, programmers and architects alike must address the performance bottlenecks of challenging workloads.

This paper presents a set of challenging benchmarks for GPGPU architectural research. Current GPGPU benchmark suites contain a broad set of workloads, most of which perform extremely well on GPUs. Current research, such as Fung and Aamodt's thread block compaction work [6], include their own sets of challenging benchmarks for their specific problem domain. However, an important question remains: in general, what are the truly difficult GPU workloads, and what makes them challenging? Answering this question will help to direct future GPU architecture research.

GPGPU performance is primarily limited by program parallelism, control flow issues, and stalls due to memory accesses. Without sufficient parallelism (threads), GPGPU workloads cannot exploit the massively parallel architecture. Control flow issues can further limit performance, even if the workload does follow the GPU single-instruction, multiple-data paradigm. Finally, the performance impact of long memory latencies and limited memory bandwidth is significant if not hidden by concurrent threads. Therefore, a truly challenging GPGPU benchmark suite must include workloads that trigger multiple combinations of these (and maybe other) performance bottlenecks, while still being a representative set of real applications that users care about.

In this work, we first survey multiple GPU applications, and present such a challenge benchmark suites. We present a detailed characterization of their execution on GPUs, and identify key architectural bottlenecks or application properties that make these benchmarks challenging. Based on this characterization data, we then use an analytic model to predict the performance impact of mitigating each of these bottlenecks. Many of these challenge benchmarks require multiple bottlenecks to be alleviated before they achieve good performance. We find that not only has the low hanging architectural fruit been picked, but that there is no silver bullet waiting to help architects reach the next level of performance on these workloads.

Our study shows that if multiple benchmark-specific architectural techniques are applied, these benchmarks can be conquered. Our broad conclusion is that GPUs will be forced to turn to specialization to energy-efficiently improve performance, much sooner than generally anticipated. CPUs also must do the same, and hence we see opportunities for synergy.

The limitations of our study include both the benchmarks used and our methodology. We use available algorithms and CUDA code that may not be fully optimized, but that represents the current standard in research. We also acknowledge that some of the applications are heavily skewed in their memory access compared to their computation. Building a viable general-purpose machine to perform well on them may be an unrealistic goal. Finally, our study relies on a mix of data from real hardware, simulation, and analytic modeling; we discuss errors introduced by this approach in Section III.

Section II describes our selection process for the

Appears EAMA Workshop 2011, Held in conjunction with ISCA

TABLE I
BENCHMARK EFFECTIVE IPC (CHALLENGE BMKS SHADED)

	Benchmark	Abbrev	Input Size	Eff IPC
GPGPUsim	BlackScholes	BLK	400M	202
	AES Cryptography	AES	256KB	184
	StoreGPU	STO	192KB	184
	Ray Tracing	RAY	256x256 image	159
	Coulumbic Potential	CP	200 atoms, 256x256	147
	Libor Monte Carlo	LIB	15 options, 4K paths	129
	3D Laplace Solver	LPS	100x100x100	104
	Fast Walsh Transform	FWT	8M elements	—
	gpuDG	DG	N=6, 2 steps	—
	Weather Prediction	WP	10 timesteps	43.2
	Neural Network	NNW	28 digits	12.4
	N-Queens Solver	NQU	10 queens	8.5
	Mummer	MUM	200 queries/30K entries	3.8
	Breadth First Search	BFSG	64K nodes	3.7
Rodinia	Cellular Automata	CELL	1024x32, 8×	228
	Kmeans	KM	494K objects	193
	Hotspot	HOT	512x512x2	191
	Leukocyte	LKT	10 frames	180
	PathFinder	DYN	8192x8192x32	169
	SRAD 2	SRAD2	402x458, 10×	158
	Gaussian	GAU	dim = 512	139
	LU Decomposition	LUD	dim = 256	135
	ParticleFilter(fp64)	PFT	10000 particles, 10 frames	116
	Streamcluster	SC	65K points	90.5
	SRAD 1	SRAD	402x458, 10×	86.9
	Backprop	BPP	64K elements	82.5
	HW Tracking	HWT	10 frames	81.2
	Heartwall	HW	5 frames	81.2
	Comp Fluid Dyn	CFD	97K data points	74.9
	Breadth First Search	BFS	1M nodes	44.5
	Nearest Neighbor	NNB	42K records, 4 files	7.4
	Needleman-Wunsch	NW	4K elements	4.0
	Myocyte	MYO	100 ms, 100×	1.6
PARSEC	Fluidanimate	FLD	100 frames, 4K cells	0.2
	Swaptions	SWP	64 swaptions, 20K sims	3.8
Other	S3D(fp64) [5]	S3D	4K points	—
	Mummer++ [7]	MMP	200 queries/33K entries	0.3

challenge benchmarks. In Section III, we characterize the challenge benchmarks, and analyze their bottlenecks. In Section IV, we discuss design choices suggested by the bottleneck model and their impacts. Finally, we conclude in Section V with a discussion of the implications of our findings for the future of GPGPU architecture research.

II. CHALLENGE BENCHMARKS

We begin our search for challenge benchmarks with a survey of the benchmarks listed in Table I. We obtained these benchmarks from the GPGPU-Sim suite [2], Rodinia suite [4], PARSEC suite [3], and other suites. We limited our search to CUDA benchmarks as CUDA is supported by both GPU hardware and the GPGPU-Sim simulator [2] and is one of the most widely used GPU programming languages. GPGPU-Sim and Rodinia are commonly used CUDA benchmark suites. PARSEC is a heavily used multicore benchmark suite from which we ported fluidanimate and swaptions. Rather than making wholesale algorithmic changes, our implementations modify existing algorithms for GPU execution. The remaining two benchmarks are cited in previous work as challenging workloads [6].

In Table I, we list these benchmarks, input sizes used, and our observed *effective IPC*. The effective IPC is the IPC using only useful instructions per cycle (e.g., ignoring masked instructions due to warp divergence) and is found using GPGPU-Sim with a Tesla C1060-like configuration (see Section III-A). The peak IPC for this system is 240. Our study focuses only on kernels executed on the GPU – CPU work is ignored. Further, except for the two benchmarks indicated on the table, all others use only 32-bit floating point computation.

We classify any benchmark with overall or per kernel effective IPC less than 40% of the peak (96) as a *challenging benchmark*; these benchmarks are shaded in the table. As mentioned earlier some applications are inherently skewed in their computation to memory access ratio and for these improving the utilization of compute engines may be inherently hard in a general-purpose setting. A third of the GPGPU-Sim benchmarks and just over half of the Rodinia benchmarks are classified as challenging[1].

III. GPU BOTTLENECKS

A. Overview

We now analyze the performance bottlenecks for these challenge benchmarks and determine the performance impact of each bottleneck. We use the GPGPU-Sim simulator [2] for determining the bottlenecks and a GPU performance model for determining the performance impact from removing the bottleneck. We classify the bottlenecks into three categories: parallelism, control flow, and memory limitations. We present a validation of both the simulator and the performance model by comparing their projections to measurements from real hardware.

We first describe our methodology and tools used. Our study focuses on an Nvidia Tesla C1060 GPU, although we expect the conclusions to hold for similar GPU architectures. The C1060 has 30 simultaneous multicores (SMs) with eight streaming processors (SPs) each for a peak compute capability of 240 instructions per cycle (IPC), and a peak memory bandwidth of 102 GB/s.

The GPGPU-Sim simulator (version 2.1.1b) models the general-purpose functionality of GPUs, including SMs, SPs, registers, memory, memory controllers, interconnect, and local, shader, texture, and constant memory.

[1]Performance information for DG, FWT and $S3D$ are missing due to GPGPU-Sim runtime or compilation errors; we include $S3D$ in the challenge benchmark suite based on hardware profiling results.

TABLE II

DETAILED CHALLENGE BENCHMARK ANALYSIS (KERNELS IN NUMERIC-ALPHA ORDER, EXCEPT NNW, WHICH IS IN LAYER ORDER).

			Available Parallelism			**Control Flow**		**Memory**			
Kernel	Eff IPC	Kernel Time	Blocks	Th. per Block	Total Threads	Avg Th. per Warp	Serial	Accesses Coalesced	DRAM BW (GB/s)	Stalled for Mem	Anticipated Bottlenecks
BFS_1	4.87	92%	1954	512	1000448	10	25%	56%	70	76%	WP, ST
BFS_2	104.28	8%	1954	512	1000448	27	4%	97%	34	33%	LAT
BFS_G	3.69	100%	256	256	65536	10	25%	50%	69	66%	WP, ST
BPP_1	12.07	66%	4096	256	1048576	11	0%	88%	23	76%	WP,BW
BPP_2	132.94	34%	4096	256	1048576	12	0%	93%	41	11%	—
CFD_1	72.02	89%	506	192	97152	31	0%	76%	93	64%	BW
CFD_2	173.56	2%	506	192	97152	32	0%	94%	80	9%	BW
CFD_3	100.79	0%	506	192	97152	32	0%	94%	65	5%	—
CFD_4	82.09	9%	506	192	97152	32	0%	94%	91	62%	BW
FLD_1	0.81	0%	19	256	4864	14	11%	7%	47	96%	LAT, WP, BP, ST
FLD_2	0.16	40%	32	256	8192	3	39%	4%	14	—	LAT, BP, WP, ST
FLD_3	1.49	0%	32	256	8192	8	3%	13%	67	88%	LAT, WP, BP
FLD_4	0.12	58%	32	356	8192	3	51%	3%	13	40%	LAT, WP, ST, BP
FLD_5	1.22	0%	19	256	4864	13	12%	7%	43	94%	LAT, WP, BP
FLD_6	2.49	0%	19	256	4864	19	7%	94%	25	79%	WP, BP
HW	81.17	100%	51	512	26112	25	1%	91&	86	26%	BW, BP
HWT	81.22	100%	51	512	26112	25	1%	91%	86	26%	BW, BP
MUM	3.75	100%	196	256	50176	8	37%	77%	52	58%	WP, ST
MMP	0.28	100%	1	256	256	8	26%	30%	4	44%	BP, WP, ST
MYO	1.60	100%	4	32	128	25	0%	0%	14	91%	BP, LAT
NNW_1	42.59	3%	168	169	28392	27	0%	90%	64	65%	LAT
NNW_2	11.96	19%	1400	25	35000	25	0%	83%	83	91%	BW
NNW_3	0.12	78%	2800	1	2800	1	100%	0%	80	47%	TP, WP, ST, BW
NNW_4	0.11	1%	280	1	280	1	100%	0%	68	44%	TP, WP, ST
NNB	7.40	100%	938	16	15008	16	0%	22%	98	86%	LAT, WP, BW
NQU	8.53	100%	256	96	24576	26	6%	90%	0	43%	ST
NW_1	4.14	49%	1 to 127	16	16 to 2032	11	5%	83%	6	82%	WP, BP, TP
NW_2	3.91	51%	1 to 127	16	16 to 2032	11	5%	83%	5	82%	WP, BP, TP
SC	90.52	100%	128	512	65536	30	5%	93&	61	46%	BW
$SRAD_1$	205.02	0%	450	512	230400	31	0%	94%	41	2%	—
$SRAD_2$	207.52	0%	450	512	230400	32	0%	94%	57	4%	—
$SRAD_3$	53.01	41%	450	512	2304000	18	17%	93%	7	3%	WP, ST
$SRAD_4$	116.52	32%	1 or 450	512	512 or 2304000	32	0%	42%	67	76%	BP
$SRAD_5$	91.69	21%	450	512	2304000	32	0%	93%	78	57%	BW
$SRAD_6$	98.47	6%	450	512	2304000	32	0%	94%	89	54%	BW
SWP	3.78	100%	1	512	512	21	1%	94%	15	18%	BP, WP
WP	43.22	100%	72	64	4608	25	3%	83%	55	65%	TP

KEY			
Available Parallelism	⇒	TP: Threads per Block,	BP: Blocks per Kernel
Control Flow	⇒	WP: Parallelism within Warp,	ST: Serial Execution
Memory Accesses	⇒	BW: Memory Bandwidth,	LAT: Memory Latency

We used the performance model proposed by Hong and Kim [8]. It models GPU performance by considering the computational and memory resources in the hardware and the following application characteristics: memory accesses, synchronization, block and grid structure.

B. Characterization

In Table II, we present the detailed workload characterization for each kernel in the challenging benchmarks using data from the GPGPU-Sim simulator. The first three columns give general information about the kernel: name, effective IPC, and percent of GPU time spent in that kernel. Note that some kernels have IPCs greater than 96; we include all kernels from a challenging benchmark even if the particular kernel performs well. The next eight columns are divided into three sets: Available Parallelism, Control Flow, and Memory. Section III-C discusses these columns in detail, and is organized similarly. The last column gives our diagnosed bottlenecks based on intuition from the upcoming data analysis.

C. Data Analysis

The following likely GPU bottlenecks are included in Table II, with abbreviations listed in the table key:

1) Available Parallelism: GPUs achieve high performance by running many concurrent threads on their massively parallel architecture, but the total number of threads can be limited by the number of blocks in the kernel (BP) or the number of threads per block (TP). Block and thread level parallelism is limited by the fraction of the algorithm that has been parallelized and the problem size. In our table, we consider a kernel parallelism limited if there are fewer than ten thousand total threads (each SM is less than half full), and observe that 12 of the 38 kernels are limited by available parallelism.

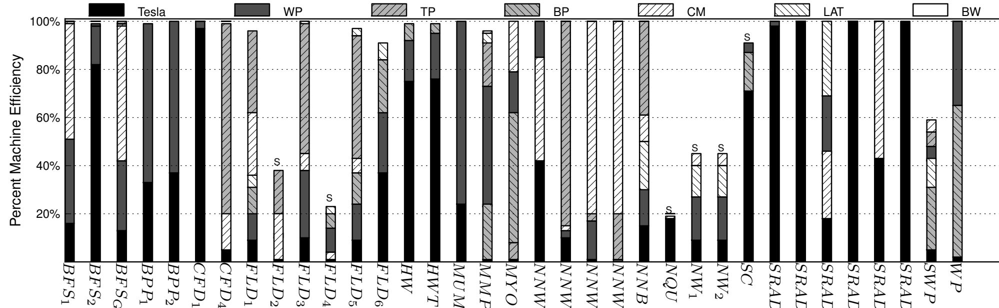


Fig. 1. Modeled bottleneck removal: impact of reducing warp divergence (WP), increasing threads per block (TP), increasing blocks per kernel (BP), coalescing all memory accesses (CM), reducing memory latency (LAT), and increasing memory bandwidth (BW). (S: sync overheads)

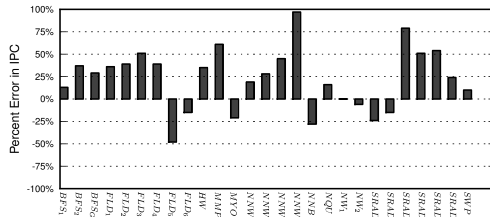


Fig. 2. Simulator IPC validation against Tesla C1060

2) Control Flow: The single-instruction, multiple-thread (SIMT) architecture of GPUs makes control flow divergence a limiting factor for performance. We quantify the impact of thread divergence by measuring the average number of active threads in a warp over all warp issues. We further measure the average number of warp issues with only a single thread, which indicates serial execution, synchronization, atomic operations, or extreme thread divergence. Nineteen of the kernels have fewer than 25 active threads per warp (WP) and twelve of them have more than 10% of their issue cycles with only a single active warp (ST).

3) Memory Accesses: Limited caching and heavy cache contention make GPUs dependent on many accesses to main memory, and the long latencies may not always be hidden by the heavy multi-threading if parallelism is limited or there are many memory accesses. We observe that ten of the kernels use more than 70% of the total 102 GB/s of memory bandwidth (BW). It is important to note that DRAM performance can slow with more than 70% utilization due to queuing effects and memory access bursts. Further, we suspect that nine benchmarks with few coalesced memory accesses and many stalls for memory accesses are slowed by the long latency of memory accesses (LAT). We note here that applications that are limited by memory access latency could overcome this using massive parallelism. In our classification, this becomes a bottleneck only when such large levels of parallelism are unavailable in the application to hide the memory access latency.

D. Simulator Validation

Previous work has validated the GPGPUSim simulator against an 8 SM system [2]; we validate against the Tesla C1060 hardware. For hardware results, we use `computeprof` [1] to get the total number of GPU cycles required to execute each kernel. Since GPGPUSim uses PTX instructions, we need to translate this hardware IPC. The synthetic hardware IPC is the number of PTX instructions counted by GPGPU-Sim divided by the number of cycles executed on the hardware; this approach normalizes internal GPU instructions to PTX instructions. In Figure 2, we show the percent error in the calculated IPC for the majority of the challenge benchmark kernels. Errors are less than 50% for all kernels that contribute greater than 5% of the GPU execution time, which is sufficient for our purposes. Since, our simulator is generally over-estimating performance, in reality the performance of these challenge benchmarks will be worse on real hardware.

E. Overall Bottleneck Impacts

Now that we have shown that the challenge benchmarks span a set of performance bottlenecks, we use the Hong and Kim model to show that mitigating the impact of these constraints will indeed improve performance. For model inputs, we use the kernel characteristics listed in Table II and the Tesla C1060 architecture. We reduce or eliminate the impact of each bottleneck by assuming either algorithmic or hardware improvements. Iteratively, we reduce or eliminate the bottleneck with the biggest performance impact until all six bottlenecks discussed have been removed. We repeat this process for each kernel; each kernel may have bottlenecks removed in a unique order. The results of this process are in Figure 1, where the y-axis gives the percent machine efficiency (percent of peak IPC, 240).

We make relatively to extremely optimistic assumptions about potential algorithmic or hardware changes to predict maximum potential improvements. To increase the level of parallelism, we increase either the number

of blocks or number of threads per block, keeping the total amount of work done constant (improving BP and TP, respectively). For control flow issues, we assume that the algorithm and/or hardware can make it look like all threads are active in every warp (WP). We assume that synchronization points cannot be removed from kernels, and do not model the removal of this performance limiting factor. For memory accesses, we try three scenarios: memory bandwidth increases by five times (BW), memory latency halved (LAT), and all memory accesses coalesced (CM). Note that coalescing memory accesses decreases the effective memory access latency seen by the kernel.

Figure 1 shows that 32 of the 38 kernels reach practically 100% machine efficiency. The six kernels that do not reach near peak machine efficiency are limited by synchronization (labeled S in Figure 1). Kernels require up to five bottleneck removals to achieve maximum performance. While the first bottleneck relieved may not have the largest impact, it must be mitigated before later bottlenecks have an impact (e.g., increasing the number of blocks may only have a significant impact once memory is fast enough to keep the threads supplied). Across the kernels, the first bottleneck removed varied. Interestingly, increasing memory bandwidth does not appear to improve performance for kernels. However, as discussed in the model validation below, the model does not include any queuing effects for memory accesses. Given that many kernels use close to the bandwidth limit, we observe these performance degradations in both the GPGPU-Sim simulations and our hardware data. This analysis shows that these kernels must obtain a geometric mean speedup of 19× to reach peak machine efficiency.

F. Model Validation

For the analytic model, we are more interested in the model's ability to predict trends than its raw performance predictions. Therefore, Figure 3 shows the percent error in the model's predicted speedup from a Quadro FX580 with 4 SMs to a Tesla C1060 with 30 SMs. Typical errors vary widely, from -70% to 2×. Two benchmarks, FLD_4 and NQU, have even higher errors: $505\times$ and $30\times$. The model only includes synchronization delays due to `__syncthreads()` within a block; kernels that use synchronization outside of a block, as these two kernels do, will have their performance overpredicted. The model is particularly optimistic when memory traffic is significant; it does not model any queuing effects at the memory controller or bursty traffic. This validation shows that the model includes key bottlenecks, and we expect the trends suggested by the model to hold. Like the simulator, the model predominantly over-predicts,

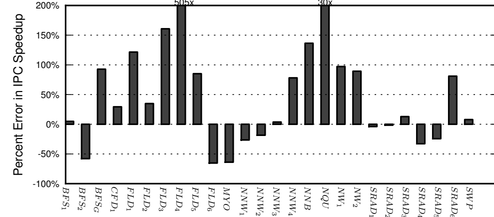


Fig. 3. Model speedup validation against Tesla C1060/QuadroFX580

and hence in reality, alleviating any one of the bottlenecks is likely to have less performance improvement on real hardware. This suggests, the projections from our study under-estimate the severity of these challenge workloads.

G. Limitations

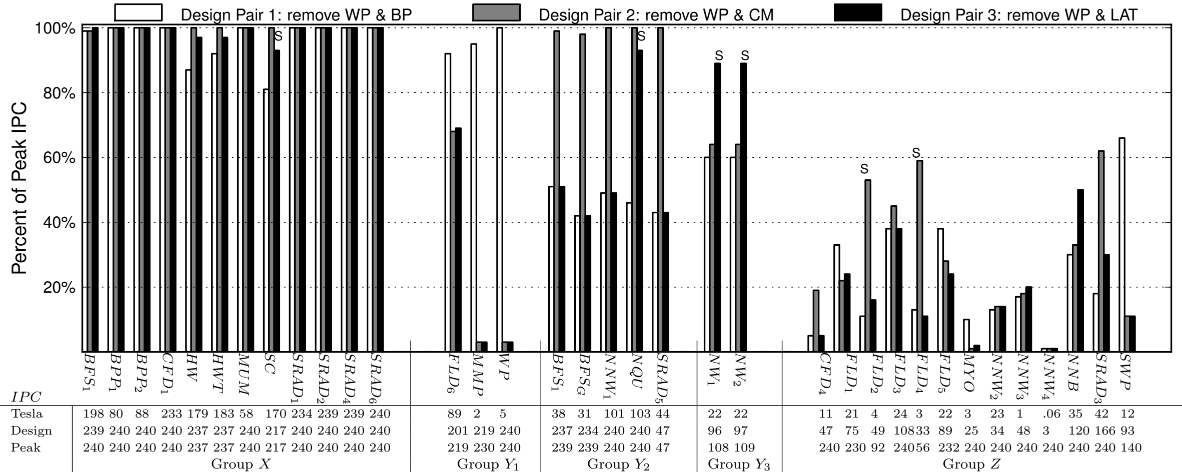
Our technique's limitations include our use of freely available CUDA benchmark implementations, potentially unoptimized algorithms, simulators, and analytic models. For many statistics, `computeprof` only has counters on a single SM. Kernels with limited parallelism or non-steady state behavior are not well profiled with counts from a single SM, and so using a simulator allows us to collect a richer and more representative set of data. As shown in the previous section, using the simulator and model also introduces errors in our predictions. From a workload perspective, we acknowledge that the benchmarks could potentially be rewritten to be less challenging in the future, especially if algorithms are designed specifically to exploit the GPUs architectural features.

IV. DESIGN

We now apply the analysis from the previous section to explore the types of design improvements that must be made to continue improving GPGPU performance. Given that many kernels are limited by warp divergence, memory latency, synchronization, and available parallelism, just adding additional cores or picking a single bottleneck to mitigate is not sufficient to improve performance across all benchmarks. Instead, we search for a pair of bottlenecks that architects can focus their design work on and expect that this *design pair* will improve performance across many challenge benchmarks. We focus on just two design features at a time to keep the design effort practical and reduce added complexity.

To identify the design pairs most likely to improve performance over many kernels, we find each kernel's performance after applying all possible design pairs. We require that each of our proposed design pairs improve performance to near maximal for at least two kernels.

The approach produces three design pairs: (1) removing warp divergence and spreading work across an optimal number of blocks, (2) removing warp divergence



IPC	BFS_1	BPP_1	BPP_2	CFD_1	HW	HWT	MUM	SC	$SRAD_1$	$SRAD_2$	$SRAD_4$	$SRAD_6$
Tesla	198	80	88	233	179	183	58	170	234	239	239	240
Design	239	240	240	240	237	237	240	217	240	240	240	240
Peak	240	240	240	240	237	237	240	217	240	240	240	240
	Group X											

IPC	FLD_6	MMP	WP
Tesla	89	2	5
Design	201	219	240
Peak	219	230	240
	Group Y_1		

IPC	BFS_1	$BFSG$	NNW_1	NQU	$SRAD_5$	NW_1	NW_2
Tesla	38	31	101	103	44	22	22
Design	237	234	240	240	47	96	97
Peak	239	239	240	240	47	108	109
	Group Y_2					Group Y_3	

IPC	CFD_4	FLD_1	FLD_2	FLD_3	FLD_4	FLD_5	MYO	NNW_2	NNW_3	NNW_4	NNB	$SRAD_3$	SWP
Tesla	11	21	4	24	3	22	3	23	1	.06	35	42	12
Design	47	75	49	108	33	89	25	34	48	3	120	166	93
Peak	240	230	92	240	56	232	240	240	240	240	240	240	140
	Group Z												

Fig. 4. Modeled speedups after adding design pairs. Table includes IPC for Tesla, adding best design pair, and peak IPC. (S: sync overheads)

and coalescing all memory accesses, and (3) removing warp divergence and halving the latency of memory accesses. Across these three design pairs, nearly two thirds of the kernels reach near optimal performance.

In Figure 4, we plot the percent of optimal IPC achieved by each kernel after implementing each of these design pairs. Below the figure, we include a three line table with the following rows: (1) model predicted IPC, (2) best IPC after adding one design pair, (3) peak IPC from Section III. The kernels can be binned into three groups: those with near peak IPC after any new design pair is introduced (Group X), those with near peak IPC after particular design pairs are implemented (Group Y), and those where implementing only one design pair is not sufficient to achieve near peak IPC (Group Z). Group Y is further subdivided by the corresponding design pair.

Note that while Group X contains twelve kernels, nine had nearly peak IPC before any design improvements were applied. Kernels in Group Y require implementation of specific design pairs. Group Z has no single design pair that obtains near peak performance. In the best case, SWP reaches 66% of peak IPC, and on average kernels in Group Z reach 26% of peak IPC. *Thus, there is no silver performance bullet for either a single kernel or across all kernels in our challenge benchmarks suite.*

Some current changes to GPU hardware seem tailored toward making GPUs more general-purpose and improving GPGPU performance. The Fermi hardware includes additional L1 caching and an L2 cache to reduce memory latency [9]. Atomic operations are also significantly faster, at least partially due to the L2 cache. Fermi also increases the number of SPs per SM, effectively doubling the peak IPC. These changes will not, however, address performance issues for kernels with limited parallelism or significant control flow overheads. We performed the same profiling study on a Tesla C2050 (a Fermi GPU) and found that challenge benchmarks were only sped up by 1.5×. Current work in thread block compaction [6] addresses the warp divergence issue that we observed, but on average, our model suggests that only eliminating thread divergence speeds up challenge kernels by just 1.8×. These are significant speedups, but our model suggests that kernels from challenge benchmarks must obtain a geometric mean speedup of 19× to reach peak machine efficiency.

V. Conclusions

This paper characterizes a set of GPGPU challenge benchmarks to find their performance bottlenecks and predict the possible performance improvements after mitigating those bottlenecks. We found that the bottlenecks for challenge benchmarks are distributed across memory, control flow, and parallelism limitations, and on average leave a 19× performance gap from the peak achievable performance. This need for higher performance exists for at least half of the benchmarks in common suites, but there is no single architectural feature to focus on for that improvement. A small number of kernels are heavily skewed in their computation to memory usage and hence building a general-purpose machine that uses the compute engines well for these applications may be unrealistic.

One of the contributions of our work is to identify these challenge benchmarks. We have shown that there is need for significant innovation to increase GPU performance on them. We also expect that these results apply to other many-core technologies and vector extensions like Intel's AVX.

While no single technique helps improve performance for many benchmarks across the board, benchmark-

specific techniques show promising results. There appears to be no low hanging fruit or a silver bullet in sight for architects to enable high performance for GPUs on these benchmarks and thus diversify the application space for GPUs even further. We believe, the main implication of our study is that GPUs will be forced to turn to specialization to energy-efficiently improve performance, much sooner than generally anticipated.

Acknowledgments

We thank the anonymous reviewers and the Vertical group for comments and the Wisconsin Condor project and UW CSL for their assistance. Many thanks to Mark Hill for comments on improving the paper. Support for this research was provided by NSF under the following grants: CCF-0845751, CCF-0917238, and CNS-0917213. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of NSF or other institutions.

References

[1] Nvidia compute visual profiler version 3.1. http://developer.download.nvidia.com/compute/cuda/3_1/toolkit/docs/VisualProfiler/computeprof.html.

[2] A. Bakhoda, G.L. Yuan, W.W.L. Fung, H. Wong, and T.M. Aamodt. Analyzing cuda workloads using a detailed gpu simulator. In *ISPASS 2009.*, pages 163 –174, April 2009.

[3] Christian Bienia. *Benchmarking Modern Multiprocessors*. PhD thesis, Princeton University, January 2011.

[4] Shuai Che, M. Boyer, Jiayuan Meng, D. Tarjan, J.W. Sheaffer, Sang-Ha Lee, and K. Skadron. Rodinia: A benchmark suite for heterogeneous computing. In *IISWC 2009.*, pages 44 –54, 2009.

[5] Anthony Danalis, Gabriel Marin, Collin McCurdy, Jeremy S. Meredith, Philip C. Roth, Kyle Spafford, Vinod Tipparaju, and Jeffrey S. Vetter. The scalable heterogeneous computing (shoc) benchmark suite. GPGPU '10, pages 63–74, 2010.

[6] Wilson W.L. Fung and Tor M. Aadmodt. Thread block compaction for efficient simt control flow. In *HPCA-17*, 2011.

[7] Abdullah Gharaibeh and Matei Ripeanu. Size matters: Space/time tradeoffs to improve gpgpu applications performance. SC '10, pages 1–12, 2010.

[8] Sunpyo Hong and Hyesoon Kim. An analytical model for a gpu architecture with memory-level and thread-level parallelism awareness. ISCA '09, pages 152–163, 2009.

[9] NVIDIA. Nvidias next generation cuda compute architecture: Fermi. http://www.nvidia.com/content/PDF/fermi_white_papers/NVIDIA_Fermi_Compute_Architecture_Whitepaper.pdf, 2009.