Supplementary Materials to Adaptive and Transparent Cache Bypassing for GPUs

Ang Li[*,†], Gert-Jan van den Braak[*], Akash Kumar[‡], and Henk Corporaal[*]

[*]Eindhoven University of Technology, Eindhoven, The Netherlands

[†]National University of Singapore, Singapore

[‡]Technische Universität Dresden, Dresden, Germany

{ang.li, g.j.w.v.d.braak}@tue.nl, akash.kumar@tu-dresden.de, h.corporaal@tue.nl

ABSTRACT

This document is the supplementary supporting file to the corresponding SC-15 conference paper titled *Adaptive and Transparent Cache Bypassing for GPUs*. In this document, we first show the experiment figures for the four extra GPU platforms that cannot fit into the original paper due to page limitation. We then show the simulation results for the hardware approach that attempts to reduce bypass overhead. Finally, we analyze the performance patterns of the applications with respect to different bypassing threshold, which may explain why certain applications can benefit significantly from cache bypassing than others.

CCS Concepts

•Computer systems organization → Multiple instruction, multiple data; •Software and its engineering → Source code generation;

Keywords

Cache bypassing; GPUs; Thread throttling

1. EXPERIMENTS

In this section, we show the experiment figures for the four additional GPU platforms (Platform-4 to 7) which cannot be included in the conference paper. The platform information is listed in Table 1. The application information is listed in Table 2. The results for 16KB L1, 48KB L1 and L2 cache bypassing on Fermi GPU with CC-2.1 are illustrated in Figures 1, 2 and 3. The results for 16KB, 32KB, 48KB L1 and L2 cache bypassing on Kepler GPU with CC-3.0 are shown in Figures 4, 5, 6 and 7. The results for 16KB, 32KB, 48KB L1, read-only cache and L2 cache bypassing on Kepler GPU with CC-3.5 are illustrated in Figures 8, 9, 10,

SC '15, November 15-20, 2015, Austin, TX, USA

DOI: http://dx.doi.org/10.1145/2807591.2807606

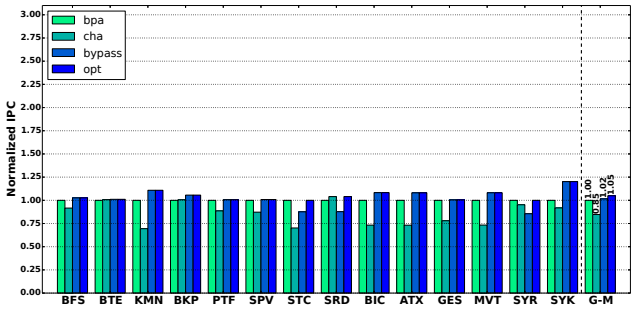

Figure 1: 16KB L1 bypassing on Fermi CC-2.1.

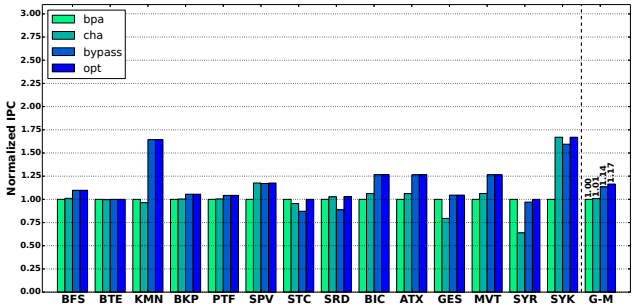

Figure 2: 48KB L1 bypassing on Fermi CC-2.1.

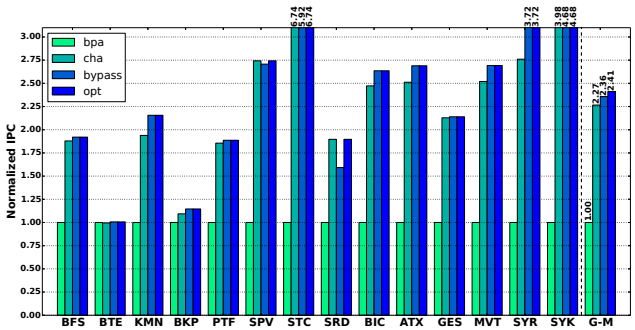

Figure 3: L2 bypassing on Fermi CC-2.1.

Table 1: Experiment Platforms

Plat.	GPU	Arch-code	CC.	Cores	GPU Freq	Mem Band	Dri/Rtm	CPU	gcc
1	GTX570	Fermi-110	**2.0**	15 SMx32	1464 MHz	152 GB/s	6.5/4.0	Intel Q8300	4.4.7
2	Tesla K80	Kepler-210	**3.7**	13 SMXx192	824 MHz	240 GB/s	7.0/7.0	Intel E5-2690	4.4.7
3	GTX750Ti	Maxwell-107	**5.0**	5 SMMx128	1137 MHz	86 GB/s	6.5/6.5	Intel i7-4770	4.4.7
4	GTX460	Fermi-104	**2.1**	7 SMx32	1400 MHz	88 GB/s	6.5/6.5	Intel i7-920	4.6.3
5	GTX690	Kepler-104	**3.0**	8 SMx192	1020 MHz	192 GB/s	7.0/6.5	Intel i7-5930K	4.8.4
6	Tesla K40	Kepler-110	**3.5**	15 SMXx192	876 MHz	288 GB/s	6.0/6.0	Intel E5-2620	4.4.7
7	GTX980	Maxwell-204	**5.2**	16 SMMx128	1216 MHz	224 GB/s	6.5/6.5	Intel i3-4160	4.8.2

Table 2: Benchmark Characteristics

Application	Description	abbr.	Warps	Input dataset	Source
bfs	Breadth First Search	BFS	16	graph1MW_6.txt	Rodinia[1]
backprop	Back Propagation	BKP	8	65536	Rodinia[1]
b+tree	B+ Tree Operation	BTE	8	mil.txt-command.txt	Rodinia[1]
kmeans	K-means Clustering	KMN	8	kdd_cup	Rodinia[1]
stencil	3-D Stencil	STE	4	128x128x32.bin-128-128-32-100	Parboil[2]
particlefilter	Particle Filter	PTF	16	128x128x10, np:1000	Rodinia[1]
spmv	Sparse Matrix-Vector Multiplication	SPV	6	Dubcova3.mtx - vector.bin	Parboil[2]
streamcluster	Stream Cluster	STC	16	10-20-256-65536-65536-1000	Rodinia[1]
srad	Speckle Reducing Anisotropic Diffusion	SRD	16	100-0.5-502-458	Rodinia[1]
bicg	BiCGStab Linear Solver	BIC	8	default	Polybench[3]
atax	Matrix Transpose Vector Multiply	ATX	8	default	Polybench[3]
gesummv	Scalar Vector Matrix Multiply	GES	8	default	Polybench[3]
mvt	Matrix Vector Product Transpose	MVT	8	default	Polybench[3]
syrk	Symmetric Rank-K Operations	SYR	8	default	Polybench[3]
syr2k	Symmetric Rank-2K Operations	SYK	8	default	Polybench[3]
similarityscore	Similarity Measure between Documents	SSC	16	256-128	Mars[4]

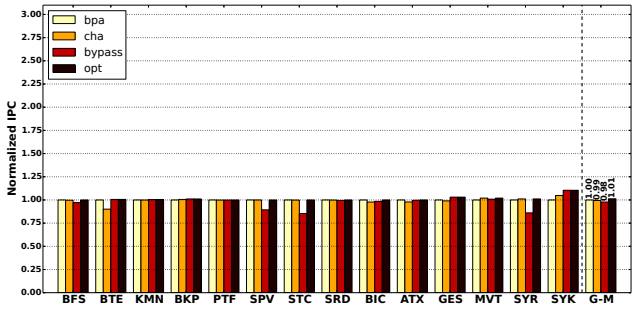

Figure 4: 16KB L1 bypassing on Kepler CC-3.0.

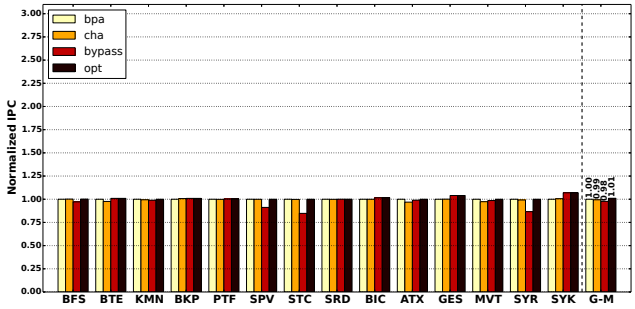

Figure 5: 32KB L1 bypassing on Kepler CC-3.0.

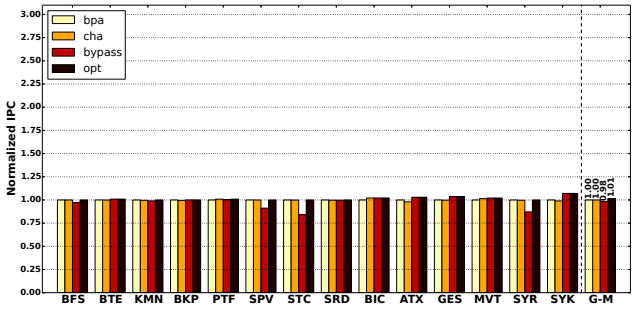

Figure 6: 48KB L1 bypassing on Kepler CC-3.0.

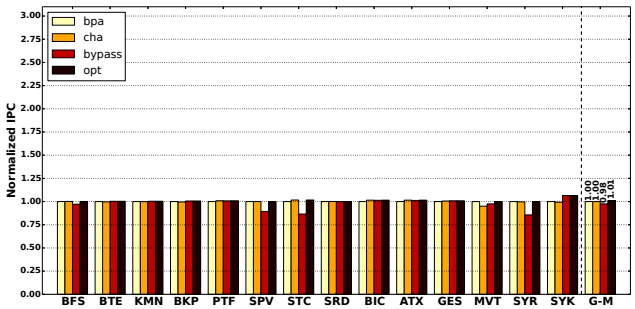

Figure 7: L2 bypassing on Kepler CC-3.0.

11 and 12. Finally, the results for read-only cache and L2 cache bypassing on Maxwell GPU with CC-5.2 are shown in Figures 13 and 14.

2. HARDWARE DESIGN

In this section, we discuss the possibility to reduce bypassing overhead via hardware approach. The idea is to implement the judging process of bypassing (shown in Listing 1 of the conference paper) in the cache controller instead

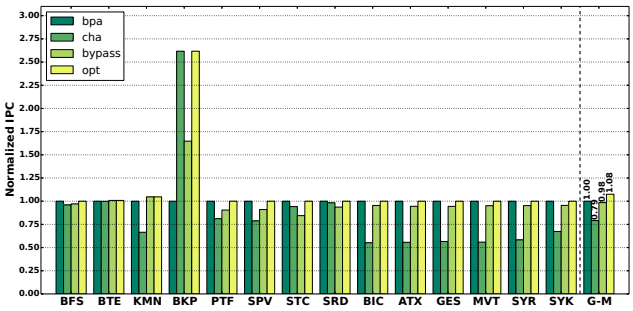

Figure 8: 16KB L1 bypassing on Kepler CC-3.5.

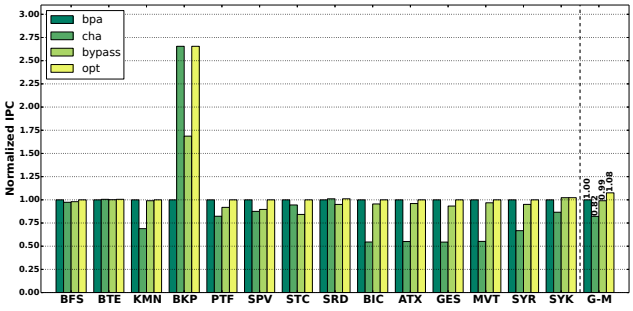

Figure 9: 32KB L1 bypassing on Kepler CC-3.5.

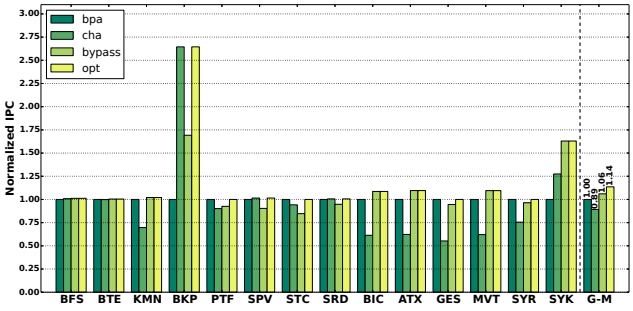

Figure 10: 48KB L1 bypassing on Kepler CC-3.5.

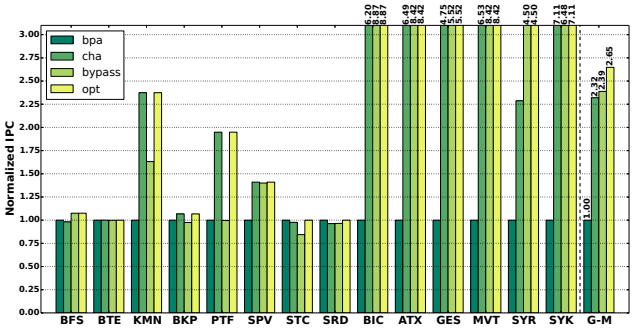

Figure 11: Read-only cache bypassing on Kepler CC-3.5.

of in the program. We use a 6-bit register[1] to conserve the

[1]As discussed in the conference paper, the maximum number of warps is 32.

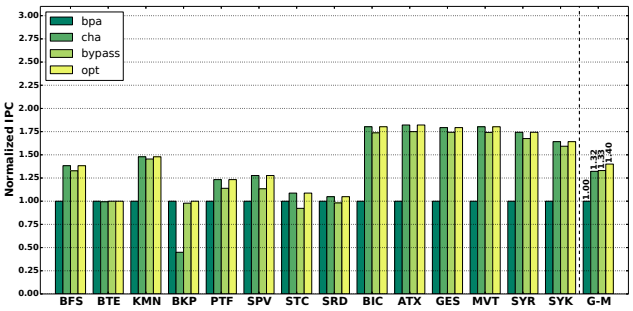

Figure 12: L2 bypassing on Kepler CC-3.5.

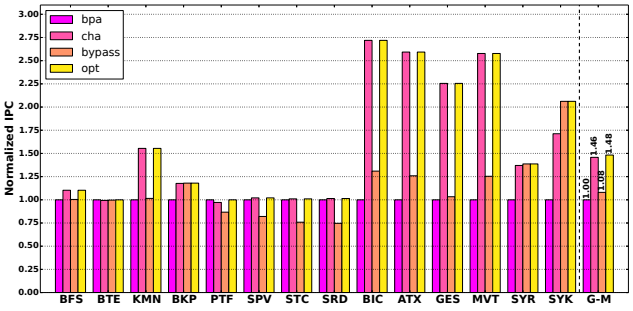

Figure 13: Read-only cache bypassing on Maxwell CC-5.2.

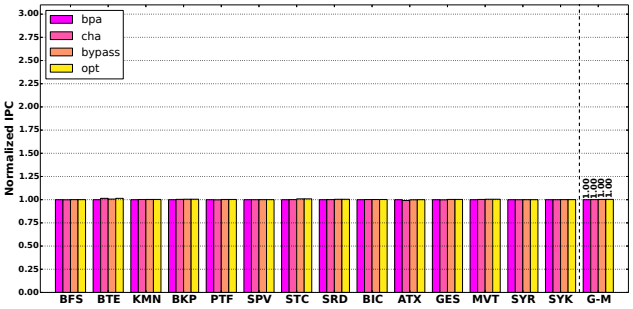

Figure 14: L2 bypassing on Maxwell CC-5.2

bypassing threshold. The register is configured when the kernel launches. Then for a memory request, upon it arrives at the cache, its warp index is compared with the threshold register, if the value is less, it is appended to the cache waiting queue, otherwise, it is forwarded to the request queue of the lower-level memory devices. For example, if bypassing L1, the request is forwarded to the *MRQ* [5] and is later injected into the interconnection network (Figure 1 of the conference paper).

Migrating the bypassing functionality into the hardware eliminates the 1-bit predicate register cost per thread as well as the corresponding assessment upon each time's memory access, which improves performance and reduces power. We implemented such design in GPGPU-Sim Version 3.3.2 [6] with the power module GPUWattch [7].

The simulation configuration is shown in Table 3. We compare the performance and power for `cha`, `bpa`, the `software` and `hardware` implementations with the optimal threshold value profiled. The results are shown in Figures 15 and

Table 3: GPGPU-Sim Configurations

Architecture	Fermi (GTX480), 15 SMx32, 700MHz
L1 cache	16KB, 32 sets, 128 B/line, LRU, 32 MSHRs
L2 cache	768KB, 6 channels, 64 sets, 128 B/line, LRU, 32 MSHRs
DRAM	6 MCs, FR-FCFS

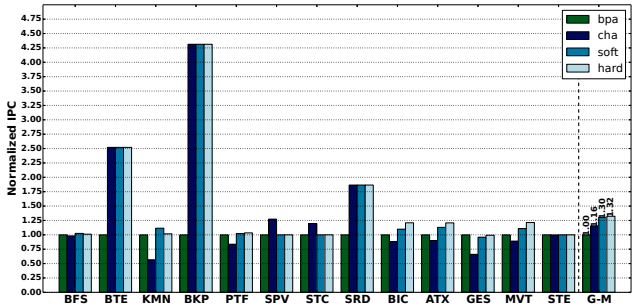

Figure 15: Simulation Results for Normalized IPC.

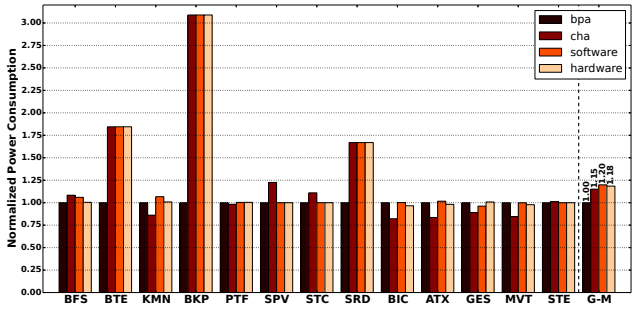

Figure 16: Simulation Results for Power.

16 for performance and power. Note, we do not include the applications of *syrk* and *syr2k* because simulation of them takes days and still cannot finish.

Although it looks the hardware implementation has a lower IPC than the software implementation (1.30x vs 1.19x), the major contribution is the *similarityscore* (see Table 2). Without *SSC*, the hardware implementation is slightly better (hardware: 1.32x vs. software: 1.30x). The reason for *SSC* showing an inconsistent behavior is that the Mars Library [4] implements Map-Reduce framework which contains numerous small kernels that are executed repeatedly. These kernels have inter-kernel locality. However, while the software implementation can configure bypassing threshold for a specific kernel, currently the hardware implementation cannot. It uses the same bypassing threshold for all kernels thus harming the inter-kernel locality. This is why the performance for `hardware` *SSC* is inferior. Further, compare Figure 15 with the real hardware testing results in Figure 12 of the conference paper, there are evident mismatch, e.g. `bpa` is better than `cha` in real hardware, but is inferior in the simulation, `cha` of *SPV* and *STC* exhibit the best in simulation but are the worst in real hardware testing, etc. This is because GPGPU-Sim does not accurately mimic the complete behaviors of the real hardware. For example, based on our previous work [8], Fermi uses an XOR-based hashing for the L1 cache, but such module is not realized in GPGPU-Sim.

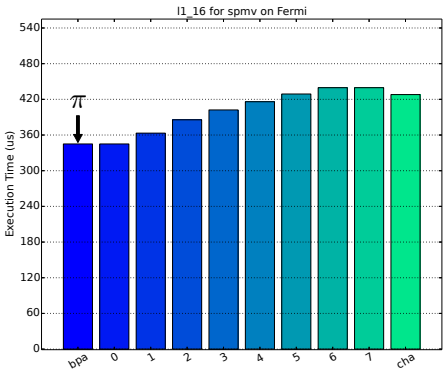

Figure 17: Bypass-favorite: `SPV` on 16 KB L1.

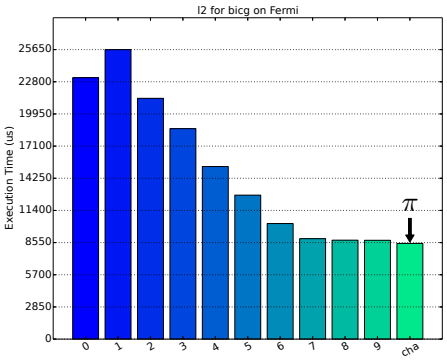

Figure 18: Cache-favorite: `BIC` on 16 KB L1.

As can be seen from Figure 16, the hardware implementation can reduce the power consumption by 4% with respect to bpa. Without *SSC*, the figure is hardware:1.20x vs. software:1.18x, which is 2%. Note, although the improvement for the hardware implementation is not prominent, it is the simulation result for the Fermi architecture, on which the overhead introduced is already quite small (less than 4%, see the conference paper). We expect more profit from Kepler and Maxwell, although only Fermi architecture is supported by the simulator.

3. APPLICATION BYPASS PATTERNS

In this section, we show the typical figures for each of the application categories based on the performance trend according to the variation of the bypassing threshold. In the conference paper, we characterize all the tested applications in Table 2 into five categories: *bypass-favorite*, *cache-favorite*, *cache-congested*, *cache-insensitive* and *irregular*. Here we show the figures for Fermi with CC-2.0 (i.e. Platform-1) as the examples.

- **Bypass-favorite**: As shown in Figure 17, the performance of bypass-favorite applications continuously degrades with a higher bypass threshold. `bpa` is the best choice. Applications such as *atax, gesummv, mvt, particlefilter* for 16KB L1 in Kepler CC-3.5 and CC-3.7 belong to this category.

- **Cache-favorite**: As shown in Figure 18, for cache-favorite applications, the performance keeps increas-

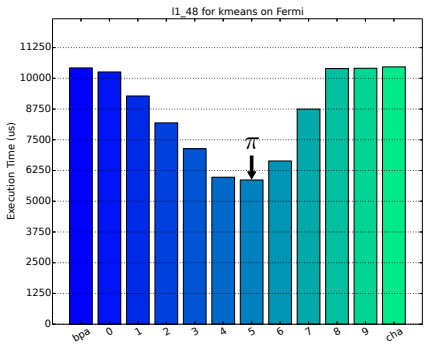

Figure 19: Cache-congested: `KMN` on 48 KB L1.

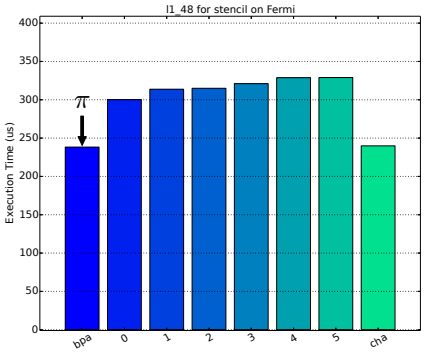

Figure 20: Cache-insensitive: `STE` on 48 KB L1.

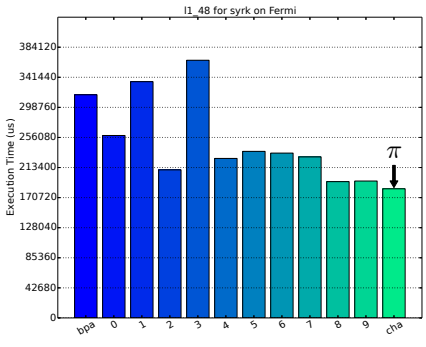

Figure 21: Irregular: `SYR` on 48 KB L1.

ing with higher threshold. `cha` is the optimal choice. Most applications on L2 of Fermi and Kepler fall in this category (Maxwell does not essentially supports L2 bypassing, as discussed in Section 5.1 of the conference paper).

- **Cache-congested**: As shown in Figure 19, for cache-congested applications, the curves are convex which looks like a bowl. Therefore, the optimal value falls in the middle. Applications such as *bfs, kmeans, bicg, mvt*, etc fall in this category and demonstrate the best bypassing performance.
- **Cache-insensitive**: As shown in Figure 20, the performance of cache-insensitive applications keeps almost steady with respect to bypassing threshold. For these applications (such as *stencil* and *streamcluster*) both `bpa` and `cha` show much better performance than adding the bypass framework. Meanwhile, `bpa` and `cha` are quite similar. Cache-insensitive applications show the worst performance for cache bypassing as it only introduces overhead. This scenario can be obtained in all figures of the conference paper and this document with the application *stencil*.
- **Irregular**: As shown in Figure 21, irregular applications show a messy shape that no clear trends are shown. *syrk* and *syr2k* are in this category.

3.1 Conclusions

In this document, we supplement the corresponding SC-2015 conference paper with three topics: the bypassing experiment figures for Platform 4 to 7, the simulation results for hardware based bypassing design and the five different performance patterns revealed by the tested applications. These contents strongly support the original conference paper.

4. REFERENCES

[1] Shuai Che, Michael Boyer, Jiayuan Meng, David Tarjan, Jeremy W Sheaffer, Sang-Ha Lee, and Kevin Skadron. Rodinia: A benchmark suite for heterogeneous computing. In *IISWC*. IEEE, 2009.

[2] John A Stratton, Christopher Rodrigues, I-Jui Sung, Nady Obeid, Li-Wen Chang, Nasser Anssari, Geng Daniel Liu, and W-M Hwu. Parboil: A revised benchmark suite for scientific and commercial throughput computing. *Center for Reliable and High-Performance Computing*, 2012.

[3] Scott Grauer-Gray, Lifan Xu, Robert Searles, Sudhee Ayalasomayajula, and John Cavazos. Auto-tuning a high-level language targeted to GPU codes. In *Innovative Parallel Computing (InPar)*. IEEE, 2012.

[4] Bingsheng He, Wenbin Fang, Qiong Luo, Naga K Govindaraju, and Tuyong Wang. Mars: a MapReduce framework on graphics processors. In *PACT*. ACM, 2008.

[5] Jaekyu Lee, Nagesh B Lakshminarayana, Hyesoon Kim, and Richard Vuduc. Many-thread aware prefetching mechanisms for GPGPU applications. In *MICRO*. IEEE, 2010.

[6] Ali Bakhoda, George L Yuan, Wilson WL Fung, Henry Wong, and Tor M Aamodt. Analyzing CUDA workloads using a detailed GPU simulator. In *ISPASS*. IEEE, 2009.

[7] Jingwen Leng, Tayler Hetherington, Ahmed ElTantawy, Syed Gilani, Nam Sung Kim, Tor M Aamodt, and Vijay Janapa Reddi. GPUWattch: enabling energy optimizations in GPGPUs. *ACM SIGARCH Computer Architecture News*, 41(3), 2013.

[8] Cedric Nugteren, Gert-Jan van den Braak, Henk Corporaal, and Henri Bal. A detailed GPU cache model based on reuse distance theory. In *HPCA*. IEEE, 2014.