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, 11.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

SC ’15, November 15-20, 2015, Austin, TX, USA
© 2015 ACM. ISBN 978-1-4503-3723-6/15/11…$15.00
DOI: http://dx.doi.org/10.1145/2807591.2807606
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 of using soft 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 of using soft hardware approach.

Table 1: Experiment Platforms

<table>
<thead>
<tr>
<th>Plat.</th>
<th>GPU</th>
<th>Arch-code</th>
<th>CC.</th>
<th>Cores</th>
<th>GPU Freq</th>
<th>Mem Band</th>
<th>Dri/Rtm</th>
<th>CPU</th>
<th>gcc</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>GTX570</td>
<td>Fermi-110</td>
<td>2.0</td>
<td>15 SMx32</td>
<td>1464 MHz</td>
<td>192 GB/s</td>
<td>6.5/4.0</td>
<td>Intel Q8300</td>
<td>4.4.7</td>
</tr>
<tr>
<td>2</td>
<td>Tesla K80</td>
<td>Kepler-210</td>
<td>3.7</td>
<td>13 SMx192</td>
<td>824 MHz</td>
<td>240 GB/s</td>
<td>7.0/7.0</td>
<td>Intel E5-2690</td>
<td>4.4.2</td>
</tr>
<tr>
<td>3</td>
<td>GTX690</td>
<td>Maxwell-106</td>
<td>5.0</td>
<td>5 SMx128</td>
<td>1137 MHz</td>
<td>86 GB/s</td>
<td>6.5/6.5</td>
<td>Intel i7-4790K</td>
<td>4.4.7</td>
</tr>
<tr>
<td>4</td>
<td>GTX690</td>
<td>Fermi-104</td>
<td>2.1</td>
<td>7 SMx32</td>
<td>1400 MHz</td>
<td>88 GB/s</td>
<td>6.5/6.5</td>
<td>Intel i7-920</td>
<td>4.6.3</td>
</tr>
<tr>
<td>5</td>
<td>GTX690</td>
<td>Kepler-104</td>
<td>3.0</td>
<td>8 SMx192</td>
<td>1020 MHz</td>
<td>192 GB/s</td>
<td>7.0/6.5</td>
<td>Intel i7-5930K</td>
<td>4.8.4</td>
</tr>
<tr>
<td>6</td>
<td>Tesla K40</td>
<td>Kepler-110</td>
<td>3.5</td>
<td>15 SMx192</td>
<td>876 MHz</td>
<td>288 GB/s</td>
<td>6.0/6.0</td>
<td>Intel E5-2620</td>
<td>4.4.7</td>
</tr>
<tr>
<td>7</td>
<td>GTX980</td>
<td>Maxwell-204</td>
<td>5.2</td>
<td>16 SMx128</td>
<td>1216 MHz</td>
<td>224 GB/s</td>
<td>6.5/6.5</td>
<td>Intel 3-4100</td>
<td>4.8.2</td>
</tr>
</tbody>
</table>

Table 2: Benchmark Characteristics

<table>
<thead>
<tr>
<th>Application</th>
<th>Description</th>
<th>abbr.</th>
<th>Warps</th>
<th>Input dataset</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>bfs</td>
<td>Breadth First Search</td>
<td>BFS</td>
<td>16</td>
<td>graph1MW.bin</td>
<td>Rodinia1</td>
</tr>
<tr>
<td>backprop</td>
<td>Back Propagation</td>
<td>BKP</td>
<td>8</td>
<td>65536</td>
<td>Rodinia1</td>
</tr>
<tr>
<td>b+tree</td>
<td>B+ Tree Operation</td>
<td>BTE</td>
<td>8</td>
<td>ml.txt-command.txt</td>
<td>Rodinia1</td>
</tr>
<tr>
<td>kneans</td>
<td>K-means Clustering</td>
<td>KMN</td>
<td>8</td>
<td>kdd.json</td>
<td>Rodinia1</td>
</tr>
<tr>
<td>stencil</td>
<td>3-D Stencil</td>
<td>STE</td>
<td>4</td>
<td>128x128x32.bin-128-128-32-100</td>
<td>Parboil2</td>
</tr>
<tr>
<td>particlefilter</td>
<td>Particle Filter</td>
<td>PTF</td>
<td>16</td>
<td>128x128x10, np:1000</td>
<td>Rodinia1</td>
</tr>
<tr>
<td>spmv</td>
<td>Sparse Matrix-Vector Multiplication</td>
<td>SPV</td>
<td>6</td>
<td>Dubcova3.mtx - vector.bin</td>
<td>Parboil2</td>
</tr>
<tr>
<td>streamcluster</td>
<td>Stream Cluster</td>
<td>ST</td>
<td>16</td>
<td>10-20-256-65536-65536-1000</td>
<td>Rodinia1</td>
</tr>
<tr>
<td>svd</td>
<td>Speckle Reducing Anisotropic Diffusion</td>
<td>SVD</td>
<td>16</td>
<td>100-1-3-602-458</td>
<td>Rodinia1</td>
</tr>
<tr>
<td>lcg</td>
<td>BICGSTab Linear Solver</td>
<td>LCG</td>
<td>8</td>
<td>default</td>
<td>Polybench3</td>
</tr>
<tr>
<td>atax</td>
<td>Matrix Transpose Vector Multiply</td>
<td>ATX</td>
<td>8</td>
<td>default</td>
<td>Polybench3</td>
</tr>
<tr>
<td>gesummv</td>
<td>Scalar Vector Matrix Multiply</td>
<td>GES</td>
<td>8</td>
<td>default</td>
<td>Polybench3</td>
</tr>
<tr>
<td>met</td>
<td>Matrix Vector Product Transpose</td>
<td>MET</td>
<td>8</td>
<td>default</td>
<td>Polybench3</td>
</tr>
<tr>
<td>syrk</td>
<td>Symmetric Rank-K Operations</td>
<td>SYR</td>
<td>8</td>
<td>default</td>
<td>Polybench3</td>
</tr>
<tr>
<td>syrk2k</td>
<td>Symmetric Rank-2K Operations</td>
<td>SYK</td>
<td>8</td>
<td>default</td>
<td>Polybench3</td>
</tr>
<tr>
<td>similarityscore</td>
<td>Similarity Measure between Documents</td>
<td>SSC</td>
<td>16</td>
<td>256-128</td>
<td>Mars4</td>
</tr>
</tbody>
</table>

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 of using soft hardware approach.
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.

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 GPUWatch [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
1.00
1.15
1.20
1.18
bpa
cha
soft
hard

Figure 16: Simulation Results for Power.

1.00
1.15
1.20
1.18
bpa
cha
soft
hard

Figure 15: Simulation Results for Normalized IPC.

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, met, 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-
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, mut, 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


