Scheduling Page Table Walks for Irregular GPU Applications

Seunghee Shin\textsuperscript{1}, Guilherme Cox\textsuperscript{2}, Mark Oskin\textsuperscript{3}, Gabriel H. Loh\textsuperscript{3}
Yan Solihin\textsuperscript{1}, Abhishek Bhattacharjee\textsuperscript{2}, Arkaprava Basu\textsuperscript{4}

\textsuperscript{1} North Carolina State University, \textsuperscript{2} Rutgers University
\textsuperscript{3} AMD Research, \textsuperscript{4} Indian Institute of Science
GPU APPLICATIONS

- Finance
- Deep Learning
- Oil & Gas Exploration
- Climate Modeling
- Big Data Analytics

Companies and Technologies:
- AMD
- BAIR
- CNTK
- Chainer
- Microsoft
- nag
- OneView
- Global Valuation
- SciFinance
- TU Delft
- NCAR
- COSMO
- blazegraph
- MAPD
- ANACONDA
- POLYMATICA
- Halliburton
- AISIGHT
- CGG
- Echelon
GPU ARCHITECTURE

- A Compute Unit (CU) is the basic computational block (8-64 CUs in a GPU)
- Each CU has multiple Single-Instruction-Multiple-Data (SIMD) units
- A SIMD unit has multiple lanes of execution (32 or 64)
- Each lane executes one workitem (thread)
Integration of CPUs and GPUs with shared memory

- Provides unified virtual address space
- Reduces CPU/GPU communication latency
- Removes the CPU/GPU programmability barrier
- Requires GPUs to walk the X86 page tables
VIRTUAL ADDRESS TRANSLATION WITH IOMMU

Number of memory accesses for translations per wavefront
- IOMMU has multiple page table walkers (e.g., 8–16)
- Number of memory accesses per SIMD instruction varies
- One SIMD instruction can create up to 256 memory accesses
- 27-61% of instructions require 1 to 16 memory accesses
- 33-70% of instructions require 49 or more
- High variance in address translation work (memory accesses) across SIMD instructions
**OVERHEADS BY IRREGULAR APPLICATIONS**

- Irregular application memory accesses have low spatial locality
- Requests queue up in the IOMMU buffer to be serviced
- Divergent accesses can slowdown irregular applications by up to 4x*

*IOMMU bottleneck

---

*Observations and Opportunities in Architecting Shared Virtual Memory for Heterogeneous Systems, ISPASS16, Jan Vesely, Arkaprava Basu, Mark Oskin, Gabriel H. Loh, Abhishek Bhattacharjee*
WHY IS IOMMU SCHEDULING IMPORTANT?

- Scheduling can be beneficial/detrimental
- Performance difference is more than 2.1x
SIMT-AWARE SCHEDULING #1: SHORTEST-JOB-FIRST (SJF)

SJF minimizes the average waiting time of wavefronts
- Approximate SJF by measuring the number of DRAM accesses per instruction (score)
- Requests from a SIMD instruction with the lowest score are serviced first
- Wavefronts with less traffic will complete faster
SIMT-AWARE SCHEDULING #2: BATCHING

Why is batching needed?

- A SIMD instruction can progress only after all addresses are translated
- The last page walk determines when a SIMD instruction can proceed
- Interleaving requests from multiple SIMD instructions slows everyone down
SIMT-AWARE SCHEDULING IMPLEMENTATION

SIMT-aware scheduling algorithm (SJF + Batching)

Step 1. Batching: schedule a request from the instruction of the most-recently scheduled walk

Step 2. SJF: schedule a request from an instruction that requires the fewest memory accesses (lowest score)
### System Configuration

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>GPU</strong></td>
<td>2GHz, 8 CUs, 4 SIMD per CU&lt;br&gt;16 SIMD width, 64 threads per wavefront</td>
</tr>
<tr>
<td><strong>L1D Cache</strong></td>
<td>32KB, 16-way, 64B block</td>
</tr>
<tr>
<td><strong>L2 Cache</strong></td>
<td>4MB, 16-way, 64B block</td>
</tr>
<tr>
<td><strong>L1 TLB</strong></td>
<td>32 entries, private, Fully-associative</td>
</tr>
<tr>
<td><strong>L2 TLB</strong></td>
<td>512 entries, shared, 16-way set associative</td>
</tr>
<tr>
<td><strong>IOMMU</strong></td>
<td>256 buffer entries, 8 page table walkers&lt;br&gt;32/256 entries for IOMMU L1/L2 TLB,&lt;br&gt;FCFS scheduling of page walks</td>
</tr>
<tr>
<td><strong>DRAM</strong></td>
<td>DDR3-1600 (800MHz), 2 channel&lt;br&gt;16 banks per rank, 2 ranks per channel</td>
</tr>
</tbody>
</table>

- GEM5 simulator modeling HSA (CPU + integrated GPU)
- Implemented detailed address translation model including request coalescer, TLB hierarchy, page table walk caches, and IOMMU
- SIMT-aware scheduling performs 30% better on average (up to 41%)
- No performance impact on regular applications
EVALUATION (2) – NORMALIZED CU STALLS

- CU stall cycles with SIMT-aware scheduling, normalized to FCFS
- CU stalls when no instruction is ready to execute
- SIMT-aware scheduling reduces stall cycles by 23% on average (up to 29%)
Observations
- Irregular GPU applications can bottleneck on address translations
- Different SIMD instructions may incur vastly different numbers of memory accesses

SIMT-aware scheduler
- Prioritize page walks from instructions that require less work to service
- Batch page walks from the same SIMD instruction

Results
- Page walk scheduling has major performance implications
- SIMT-aware scheduler shows 30% performance improvement for irregular GPU applications
DISCLAIMER & ATTRIBUTION

The information presented in this document is for informational purposes only and may contain technical inaccuracies, omissions and typographical errors.

The information contained herein is subject to change and may be rendered inaccurate for many reasons, including but not limited to product and roadmap changes, component and motherboard version changes, new model and/or product releases, product differences between differing manufacturers, software changes, BIOS flashes, firmware upgrades, or the like. AMD assumes no obligation to update or otherwise correct or revise this information. However, AMD reserves the right to revise this information and to make changes from time to time to the content hereof without obligation of AMD to notify any person of such revisions or changes.

AMD MAKES NO REPRESENTATIONS OR WARRANTIES WITH RESPECT TO THE CONTENTS HEREOF AND ASSUMES NO RESPONSIBILITY FOR ANY INACCURACIES, ERRORS OR OMISSIONS THAT MAY APPEAR IN THIS INFORMATION.

AMD SPECIFICALLY DISCLAIMS ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE. IN NO EVENT WILL AMD BE LIABLE TO ANY PERSON FOR ANY DIRECT, INDIRECT, SPECIAL OR OTHER CONSEQUENTIAL DAMAGES ARISING FROM THE USE OF ANY INFORMATION CONTAINED HEREIN, EVEN IF AMD IS EXPRESSLY ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

ATTRIBUTION

© 2018 Advanced Micro Devices, Inc. All rights reserved. AMD, the AMD Arrow logo, Radeon, and combinations thereof are trademarks of Advanced Micro Devices, Inc. in the United States and/or other jurisdictions. Other names are for informational purposes only and may be trademarks of their respective owners.