Compiler Techniques to Reduce Synchronization Overhead of GPU Redundant Multithreading

**Speaker:** Manish Gupta

Outline

• Redundant Multithreading (RMT)
• Motivation
• GPU RMT
• GPU RMT Overhead
• GPU RMT Overhead Reduction
• Summary
Redundant Multithreading (RMT)
RMT Overheads

- Input Replication
- Memory
- Logic
- Shared memory
- Output Comparison
- Redundant Computations
- Synchronization Overhead
Motivation: Why GPU Reliability?

- GPUs running HPC workloads need higher reliability
- Embedded GPUs in safety-critical autonomous vehicle

- 1 error in $1.5 \times 10^6$ miles => 16000 – 23000 FIT
- Automotive Safety Integrity Level D (ASIL) require 10 FIT
- This means detecting 99.99% of all faults
Outline

• Redundant Multithreading (RMT)

• Motivation

• GPU RMT
  • GPU Execution Model
  • Intra-LDS GPU RMT
  • Inter GPU RMT

• GPU RMT Overhead
• GPU RMT Overhead Reduction
• Summary
GPU Execution Model

- Work-items within the same workgroup can synchronize through faster LDS memory
- Work-items within different workgroups synchronize through slower global memory
GPU RMT (Intra-LDS) Replicating within a workgroup

Intra-LDS RMT
Original-Redundant WI pair executing in lockstep on the same SIMD.

Original Execution
1 WG, 1 WF, 16 WIs

RMT Execution
1WG, 1 WF, 32 WIs
GPU RMT (Inter) Replicating across workgroup

Inter RMT
Original-Redundant WI pair executing independently on different SIMDs.

Original Execution
1 WG, 1 WF, 16 WIs

RMT Execution
2 WG, 2 WF, 16 Wis each WF
GPU RMT Overheads

Overhead breakdown (x100%)

- Synchronization overhead
- Redundant execution
- Baseline execution

Intra-LDS:
- 55%

Inter:
- 1726%
• Synchronization dominates the performance overhead penalty
• We propose two new compiler transformations to reduce synchronization overhead
Outline

• Redundant Multithreading (RMT)
• Motivation
• GPU RMT
• GPU RMT Overhead
• GPU RMT Overhead Reduction
  • Intra-Permute
  • Inter-Fingerprinting
• Summary
Intra-Permute

Inter-Fingerprinting

Reg_{original} = Permute (Reg_{redundant})

• Intra-Permute uses faster register-level communication instead of LDS memory

• Inter-Fingerprinting combines multiple synchronization events into one event by hashing
## Intra-Permute

### Intra-Permute vs. Intra-LDS

- Reduced branch divergence by half
- Exclusion of `memfence`
- Fewer total instructions
- Efficient register-level communication

<table>
<thead>
<tr>
<th>(a) Original code</th>
<th>(b) Intra-LDS compiler transformation</th>
<th>(c) Intra-Permuate compiler transformation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Work-item</strong></td>
<td><strong>Work-item A (WIA)</strong></td>
<td><strong>Work-item B (WIB)</strong></td>
</tr>
<tr>
<td>R1=R2+R3; //Compute Val</td>
<td>R1=R2+R3;</td>
<td>R1=R2+R3;</td>
</tr>
<tr>
<td>R4=R4+4; //Addr</td>
<td></td>
<td>R4=R4+4;</td>
</tr>
<tr>
<td>STORE R1,</td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Evaluation Framework

• GPU kernels in C++AMP
• Embedded and Proxy Apps
• Open-source LLVM-based compiler
• RMT kernels executed on AMD FX-8800P GPU
Intra-Permute Performance

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Dynamic Local Store</th>
<th>Dynamic Global Store</th>
</tr>
</thead>
<tbody>
<tr>
<td>matmul (MM)</td>
<td>512</td>
<td>1</td>
</tr>
<tr>
<td>simpleMatmul (SM)</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>FFT</td>
<td>32</td>
<td>16</td>
</tr>
<tr>
<td>SPMV</td>
<td>1 - 5</td>
<td>1</td>
</tr>
<tr>
<td>XSBench (XSB)</td>
<td>0</td>
<td>7</td>
</tr>
<tr>
<td>CoMD</td>
<td>0</td>
<td>338</td>
</tr>
<tr>
<td>miniAMR (mAR)</td>
<td>10 - 30</td>
<td>8</td>
</tr>
<tr>
<td>snap-c (SnPC)</td>
<td>0 - 4</td>
<td>4</td>
</tr>
</tbody>
</table>
Outline

• Redundant Multithreading (RMT)
• Motivation
• GPU RMT
• GPU RMT Overhead
• GPU RMT Overhead Reduction
  • Intra-Permute
  • Inter-Fingerprinting
• Summary
Inter-Fingerprinting (Inter-FP)

Intra-FP vs. Inter
- Reduces communication overhead
- Reduces locking overhead
- Adds a small overhead of hashing
- XOR (See poster) and CRC32 hashing

hash = genHash(hash, V, A)

Global Memory

Communication
Locking
Hashing

Is Lock rel?

hash = genHash(hash, V, A)

Lock acq
Inter-FP Performance (CRC32)

See poster for XOR results
See poster for XOR results
See poster for XOR results
Inter-FP synchronization overhead is decomposed into the overhead of locking (L) and communication (C).
Summary

• We implement GPU RMT as low-cost compiler transformations which can be enable/disable based on the application at compile time

• Software-based RMT helps reduce hardware complexity

• We reduce the synchronization overhead of GPU RMT with Intra-Permute and Inter-Fingerprinting by 43% and 73.5%, respectively
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.

ATTRIBUTIONS

© 2016 Advanced Micro Devices, Inc. All rights reserved. AMD, the AMD Arrow logo and combinations thereof are trademarks of Advanced Micro Devices, Inc.

Other names are for informational purposes only and may be trademarks of their respective owners.