Streamlining GPU Applications On the Fly

Thread Divergence Elimination through Runtime Thread-Data Remapping

Eddy Z. Zhang, Yunlian Jiang, Ziyu Guo, Xipeng Shen
Department of Computer Science, College of William and Mary
GPU Divergence

• GPU Features
  – Streaming multiprocessors
    • SIMD
    • Single instruction issue per SM
  – Warp / Half Warp
    • SIMD execution unit

• Divergence
  – Threads in a warp take different execution paths
Example of GPU Divergence

Instructions: A, B, C
Control Flow

Thread Serialization Behavior
Impact of GPU Divergence

• Degrading GPU Throughput
  – E.g., up to $\frac{15}{16}$ degradation on Tesla 1060

• Impairing GPU Usage
  – Esp. when having non-trivial condition statements
Related Work

• Stream packing and unpacking
  • [Popa: U. Waterloo C.S. master thesis’04]
  • Simulate hardware packing on CPU

• Dynamic warp formation & scheduling
  • [Fung+: MICRO’07]
  • Hardware solution

• Control structure splitting
  • [Carrillo+: CF’09]
  • Reducing register pressure but removing no divergences
Basic Idea of Our Solution
Swapping Jobs of Threads through Thread-Data Remapping

```c
if ( A[tid] ) { // (green)
    C[tid] += 1;
} else { // (red)
    C[tid] -= 1;
}
```

A[ ]

threads

warp 1  warp 2  warp 3

Thread-Data Remapping
Challenges

• How to determine a desirable mapping?
  – Complexities
    • irregular accesses, complex indexing expressions, side effects on memory reference patterns, ...

• How to realize the new mapping?
  – Data movement or redirect threads’ data references
    • limitations, effectiveness, and safety.

• How to do it on the fly?
  – Large overhead v.s. Need for runtime remapping
    • dependence on runtime data values, minimizing and hiding overhead
Outline

• Thread-data Remapping
  – Concept & mechanisms

• Transformation on the Fly
  – CPU-GPU pipelining & LAM

• Evaluation

• Conclusion
GPU Divergence Causes

• Control Flows in Code
  – E.g., if, do, for, while, switch

• Input Data Dependence
  – Input data-set --> execution path
  – Thread-data mapping --> amount of thread divergence
Define Divergence

• Control Flow Path Vector for One Thread

Def: \( Pvector[tid] = < b_1, b_2, b_3, \ldots, b_n > \)

**Condition Statements**

\[
\begin{align*}
\text{if ( } A[tid] \text{ % 2 ) } & \{ \ldots \}; \\
\text{if ( } A[tid] \text{ < 10 ) } & \{ \ldots \};
\end{align*}
\]

**Path Vector Example**

<table>
<thead>
<tr>
<th>tid</th>
<th>A[tid]</th>
<th>Pvector</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2</td>
<td>&lt;0,1&gt;</td>
</tr>
<tr>
<td>1</td>
<td>11</td>
<td>&lt;1,0&gt;</td>
</tr>
<tr>
<td>2</td>
<td>14</td>
<td>&lt;0,0&gt;</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Define Divergence

• Control Flow Path Vector for One Thread

Def: $P_{vector}[tid] = < b_1, b_2, b_3, \ldots, b_n >$

Path Vector Example

<table>
<thead>
<tr>
<th>tid</th>
<th>A[tid]</th>
<th>Pvector</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2</td>
<td>&lt;0,1&gt;</td>
</tr>
<tr>
<td>1</td>
<td>11</td>
<td>&lt;1,0&gt;</td>
</tr>
<tr>
<td>2</td>
<td>14</td>
<td>&lt;0,0&gt;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>...</td>
</tr>
</tbody>
</table>

Condition Statements

if ( $A[tid] \% 2$ ) {...};
if ( $A[tid] < 10$ ) {...};
Regroup Threads

- To Satisfy Convergence Condition:
  - Sort $P_{vector}[0], P_{vector}[1], P_{vector}[2], \ldots$ for all threads
  - E.g., after sorting, the grouping of threads

<table>
<thead>
<tr>
<th>Path Vector:</th>
<th>$&lt;0,0&gt;$</th>
<th>$&lt;0,1&gt;$</th>
<th>$&lt;1,1&gt;$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Thread Index:</td>
<td>0 12 8 11 9 4 6 7</td>
<td>2 5 10 3 1 13 14 15</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Warp</td>
<td>Warp</td>
<td>Warp</td>
</tr>
</tbody>
</table>

Eddy@eddy@cs.wm.edu
Example of GPU Thread Divergence

Data Index: $j$

Path Vector:

Thread Index: $i$

Mapping:

$\text{Thread}[i] \rightarrow \text{Data}[j]$

WARP 1

WARP 2

i == j
Remapping by Reference Redirection

Data Index: $j$

Path Vector:

Mapping:
Thread[i] --> Data[j]

Thread Index: $i$

j == IND[i]
Remapping by Data Transformation

Data Index: $j$
Path Vector:

Thread Index: $i$

WARP 1
WARP 2
Remapping by Data Transformation

Data Index: \( j \)

Path Vector: [1, 2, 3, 4, 5, 6, 7, 8]

Thread Index: \( i \)

Mapping: \( \text{Thread}[i] \rightarrow \text{Data}[j] \)

\( i == j \)
Outline

• Thread-data Remapping
  – Concept & mechanisms

• Transformation on the Fly
  – CPU-GPU pipelining & LAM

• Evaluation

• Conclusion
Overview of CPU-GPU Synergy

- Path Vectors
- Compute Best Mapping
- Feedback Control

CPU

Collect Branch Info

Send Remapping Info

GPU

- Independent
- Protected
- Pipelined

• Realize Desired Thread-data Map.
CPU-GPU Pipeline Scheme

• Without Pipelining Scheme

\[
\frac{T_{\text{div}}}{(T_{\text{no-div}} + T_{\text{remap}})} : \geq 1 \text{ or } < 1
\]

• With Pipelining Scheme

\[
\frac{T_{\text{div}}}{(T_{\text{no-div}} + T_{\text{remap}})} : \geq 1
\]

No Slow-down!
CPU-GPU Pipeline Example I

**GPU Code**

Timeline

- Remapping Thread

- GPU Kernel Func.
- CPU Remap Func.

......

......
CPU-GPU Pipeline Example II

- Controllable Threading
- Adaptive to Avail. Resource
Applicable Scenarios

• Loops
  – Multiple invocation of same kernel function

• Input Data Partition for A Kernel Function
  – Create multiple iterations

• Across Different Kernels
  – With idle CPU processing system resources
LAM: Reduce Data Movements

- For Data Layout Transformation - LAM Scheme
  - 3 Steps: Label, Assign & Move (LAM)
  - Label -- Classify path vectors into multiple classes
    - Based on similarity
  - Assign -- Assign warps to different classes
    - Based on occupation ratio
  - Move -- Determine the destination
Outline

• Thread-data Remapping
  – Concept & mechanisms

• Transformation on the fly
  – CPU-GPU pipelining & LAM

• Evaluation

• Conclusion
Experiment Settings

• Host Machine
  – Dual-socket quad-core Intel Xeon E5540

• GPU Device
  – NVIDIA Tesla 1060

• Runtime Library
  – Reference redirection & data transformation
  – Pipeline threads management
## Benchmarks

<table>
<thead>
<tr>
<th>Program</th>
<th>Comments</th>
<th>Potential Div. Source</th>
<th>Percent of Div. Warps</th>
</tr>
</thead>
<tbody>
<tr>
<td>3D-LBM</td>
<td>LBM based PDE solver</td>
<td>if statement</td>
<td>50-100%</td>
</tr>
<tr>
<td>GAFORT</td>
<td>genetic algorithm</td>
<td>if statement &amp; loop</td>
<td>100%</td>
</tr>
<tr>
<td>Marching Cubes</td>
<td>graphics algorithm</td>
<td>if statement</td>
<td>100%</td>
</tr>
<tr>
<td>Reduction</td>
<td>parallel sum</td>
<td>if statement</td>
<td>100%</td>
</tr>
<tr>
<td>Blackscholes</td>
<td>option pricing</td>
<td>if statement</td>
<td>0%</td>
</tr>
</tbody>
</table>
## Benchmarks

<table>
<thead>
<tr>
<th>Program</th>
<th>Comments</th>
<th>Potential Div. Source</th>
<th>Percent of Div. Warps</th>
</tr>
</thead>
<tbody>
<tr>
<td>3D-LBM</td>
<td>LBM based PDE solver</td>
<td>if statement</td>
<td>50-100%</td>
</tr>
<tr>
<td>GAFORT</td>
<td>genetic algorithm</td>
<td>if statement &amp; loop</td>
<td>100%</td>
</tr>
<tr>
<td>Marching Cubes</td>
<td>graphics algorithm</td>
<td>if statement</td>
<td>100%</td>
</tr>
<tr>
<td>Reduction</td>
<td>parallel sum</td>
<td>if statement</td>
<td>100%</td>
</tr>
<tr>
<td>Blackscholes</td>
<td>option pricing</td>
<td>if statement</td>
<td>0%</td>
</tr>
</tbody>
</table>
Evaluation: Data Transformation

- **GAFORT**
  - Mutation probabilities
  - Regular mem. access
  - `select_cld` kernel
  - Remap scheme
    - Data layout transform.
  - Efficiency control
    - LAM & Pipeline

![Performance Comparison Graph]

<table>
<thead>
<tr>
<th>Perform.</th>
<th>Before</th>
<th>After</th>
<th>44% Reduced</th>
</tr>
</thead>
<tbody>
<tr>
<td>Div. Ratio</td>
<td>100%</td>
<td>56%</td>
<td>1.31 Speedup</td>
</tr>
<tr>
<td>Time</td>
<td>67225</td>
<td>51325</td>
<td></td>
</tr>
</tbody>
</table>
Evaluation: Reference Redirect.

- MarchingCubes
  - Div: number of vertices that intersect isosurface
  - Random memory access
  - generateTriangles2 kernel
  - Remap scheme
    - Reference redirection
  - Efficiency control
    - Pipeline

### Time (micro-sec)

<table>
<thead>
<tr>
<th>BlockSize</th>
<th>32</th>
<th>64</th>
<th>256</th>
</tr>
</thead>
<tbody>
<tr>
<td>Org.Time</td>
<td>17414</td>
<td>16707</td>
<td>16673</td>
</tr>
<tr>
<td>Opt.Time</td>
<td>12666</td>
<td>12371</td>
<td>12425</td>
</tr>
</tbody>
</table>

### Performance

<table>
<thead>
<tr>
<th>BlockSize</th>
<th>32</th>
<th>64</th>
<th>256</th>
</tr>
</thead>
<tbody>
<tr>
<td>Div.Reduct</td>
<td>99%</td>
<td>99%</td>
<td>99%</td>
</tr>
<tr>
<td>SpeedUp</td>
<td>1.37</td>
<td>1.35</td>
<td>1.34</td>
</tr>
</tbody>
</table>
Evaluation: All Benchmarks

![Bar chart showing speedup for different benchmarks and configurations]
Conclusion

• An Efficient Software Solution for GPU Div.
  – Mechanism
    • On-the-fly thread-data remapping
  – Overhead control
    • CPU-GPU pipelining: whole-system synergy
    • LAM scheme: balance between benefit & overhead
  – Effectiveness
    • Up to 1.4x speedup
    • Efficiency protection: no slowdown
Acknowledgement

• Ye Zhao
• Xiaoming Li
• NVIDIA
  – Donation of GPU Device
• NSF Funds & IBM CAS Fellowship
• Anonymous Reviewers
Questions?