Reducing Divergence in GPGPU Programs with Loop Merging

Tianyi David Han
Tarek S. Abdelrahman

March 16, 2013
Summary

• Branch divergence has become an increasingly prominent issue

• We propose Loop Merging, aiming to reduce loop-induced divergence

• Promising results on synthetic and application benchmarks
Branch-induced Divergence

- A GPU consists of SIMD execution engines

```c
if ((x = A[tid]) > 0.0F)
    y = logf(x);
else
    y = x/2.0F;
```

Low ALU utilization!
Loop-induced Divergence

• A GPU consists of SIMD execution engines

```
for (i = 0; i < C[tid]; ++i)
   // work_i
```

![Diagram showing loop execution and ALU utilization](image)

Instr. Issue

warp / Wavefront threads

Low ALU utilization!
Loop Merging (LM)

Targets two nested loops with varying inner loop trip-count

Original Execution Schedule

while (outer_cond) {
    // inner_prologue
    while (inner_cond) {
        // inner_loop_body
    }
    // inner_epilogue
}
Loop Merging (LM)

Each thread starts the next outer iteration immediately after completing the current one.
Loop Merging (LM)

Time

T1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2
T2 1 1 1 2 2 2 2 2 2 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
T3 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 3 3 3 3 3 3 3
T1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
T2 1 1 1 2 2 2 2 2 2 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
T3 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 3 3 3 3 3 3 3

Time Saved
while (outer_cond) {
    // inner_prologue
    while (inner_cond) {
        // inner_loop_body
    }
    // inner_epilogue
}

if (outer_cond) {
    // inner_prologue
    // (outer iter. #1)
    do {
        if (inner_cond) {
            // inner_loop_body
        } else {
            // inner_epilogue
            if (outer_cond) {
                // inner_prologue
                // (next outer iter.)
            }
        }
    } while (outer_cond);
}
Performance Factors

1. Variance of inner loop trip-counts

2. Variance of merged loop trip-counts

3. Divergent execution of transition code

4. Degradation of existing memory coalescing

5. Instruction overhead
Experimental Evaluation

- LM automation in an LLVM pass
  - Integrated with NVPTX back-end

- Platform
  - CUDA 5.0 on Ubuntu 10.04
  - Geforce GTX 480
  - Intel Core i7-960 with 6GB main memory

- Benchmarks
  - One synthetic benchmark (0.4x – 1.6x)
  - Five application benchmarks (0.98x – 4.3x)
Application Benchmarks

- **MCX**: MC simulation of photon movements in 3D media
- **GPU-MCML**: MC simulation of photon movements in 2D media
  - Use shared memory
- **MC-GPU**: MC simulation of X-ray propagation in 3D media
  - Use a different trajectory modeling than MCX and GPU-MCML
- **MUMmerGPU**: DNA sequence alignment
  - Parallel traversal of a suffix tree
- **MO**: molecular amplitudes on a 3-D grid
  - No loop-induced divergence
## Application Benchmarks

### Loop characteristics

<table>
<thead>
<tr>
<th></th>
<th>MCX</th>
<th>GPU-MCML</th>
<th>MC-GPU</th>
<th>MUMmerGPU</th>
<th>MO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inner loop body size</td>
<td>800</td>
<td>300</td>
<td>2700</td>
<td>120</td>
<td>160</td>
</tr>
<tr>
<td>“transition” code size</td>
<td>30</td>
<td>20</td>
<td>860</td>
<td>40</td>
<td>20</td>
</tr>
<tr>
<td>Avg. outer loop trip-count</td>
<td>558</td>
<td>74</td>
<td>100</td>
<td>400</td>
<td>60</td>
</tr>
<tr>
<td>Avg. inner loop trip-count</td>
<td>119</td>
<td>613</td>
<td>22</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>Std.dev. Of inner loop trip-count</td>
<td>269</td>
<td>975</td>
<td>11</td>
<td>1.2</td>
<td>0</td>
</tr>
<tr>
<td>Global memory load efficiency</td>
<td>0.8%</td>
<td>93%</td>
<td>3.8%</td>
<td>2.3%</td>
<td>100%</td>
</tr>
</tbody>
</table>
# Application Benchmark Performance

<table>
<thead>
<tr>
<th></th>
<th>MCX</th>
<th>GPU-MCML</th>
<th>MC-GPU</th>
<th>MUMmerGPU</th>
<th>MO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inner loop body size</td>
<td>800</td>
<td>300</td>
<td>2700</td>
<td>120</td>
<td>160</td>
</tr>
<tr>
<td>“transition” code size</td>
<td>30</td>
<td>20</td>
<td>860</td>
<td>40</td>
<td>20</td>
</tr>
<tr>
<td>Avg. outer loop trip-count</td>
<td>558</td>
<td>74</td>
<td>100</td>
<td>400</td>
<td>60</td>
</tr>
<tr>
<td>Avg. inner loop trip-count</td>
<td>119</td>
<td>613</td>
<td>22</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>Std.dev. Of inner loop trip-count</td>
<td>269</td>
<td>975</td>
<td>11</td>
<td>1.2</td>
<td>0</td>
</tr>
<tr>
<td>Global memory load efficiency</td>
<td>0.8%</td>
<td>93%</td>
<td>3.8%</td>
<td>2.3%</td>
<td>100%</td>
</tr>
</tbody>
</table>

![Bar chart showing kernel speedup for different applications and benchmark types]

- **MCX**
- **GPU-MCML**
- **MC-GPU**
- **MUMmerGPU**
- **MO**
Loop Merging vs. Loop Coalescing

• More generic than Loop Coalescing
  – Allow arbitrary natural loops with early exits
  – Loops do not have to be perfectly nested

• Restriction of LM
  – The loop nest cannot contain explicit or implicit barrier synchronization
Conclusions & Future work

• Loop merging – a GPU optimization for reducing loop-induced divergence

• Promising results from preliminary evaluation
  – Up to 1.6x on synthetic, 4.3x on applications
  – Benefit of LM sensitive to loop characteristics and program inputs

• Future work
  – Automatically select loop nests targeted by LM
Thank you!
Loop Merging (LM)
Synthetic Benchmark (SYN-LM)

• Five parameters
  – Random inner loop trip-count
  – Configurable sizes of inner loop body and epilogue

```c
for (i = 0; i < OUTER_TC; ++i) {
  n_inner_iters = get_rand(INNER_TC_AVG, INNER_TC_VAR);
  for (j = 0; j < n_inner_iters; ++j) {
    // inner loop body (INNER_SIZE instructions)
  }
  // epilogue (EP_SIZE instructions)
}
```
Experiments on SYN-LM

- Performance benefit as a function of $\text{INNER\_TC\_VAR}$
  - For different values of $\text{OUTER\_TC}$
Experiments on SYN-LM

- Performance benefit as a function of **INNER_SIZE**
  - For different values of **INNER_TC_VAR**

![Graph showing performance benefit as a function of INNER_SIZE for different values of INNER_TC_VAR. The graph plots Kernel speedup (over unoptimized) against INNER_SIZE for various INNER_TC_VAR values.](image-url)