One Stone Two Birds: Synchronization Relaxation and Redundancy Removal in GPU-CPU Translation

Ziyu Guo
Qualcomm CDMA

Bo Wu
Xipeng Shen
The College of William and Mary
Heterogeneous Systems and Programmability

- OpenCL
- mcuda [stratton+, LCPC’08]
- OpenMP to CUDA [Lee+, PPoPP’09]
- Ocelot [Diamos+, PACT’10]
- GPU-CPU translation correctness [Guo+, PACT’11]
Heterogeneous Systems and Programmability

- OpenCL
- mcuda [stratton+, LCPC’08]
- OpenMP to CUDA [Lee+, PPoPP’09]
- Ocelot [Diamos+, PACT’10]
- GPU-CPU translation correctness [Guo+, PACT’11]
Heterogeneous Systems and Programmability

- OpenCL
- mcuda [stratton+, LCPC’08]
- OpenMP to CUDA [Lee+, PPoPP’09]
- Ocelot [Diamos+, PACT’10]
- GPU-CPU translation correctness [Guo+, PACT’11]
Heterogeneous Systems and Programmability

- OpenCL
- mcuda [stratton+, LCPC’08]
- OpenMP to CUDA [Lee+, PPoPP’09]
- Ocelot [Diamos+, PACT’10]
- GPU-CPU translation correctness [Guo+, PACT’11]
GPU-CPU Translation

• Translate SPMD code to CPU architecture

• Motivations
  ❖ Easier to aggregate than to parallelize
  ❖ A large cuda code base
  ❖ Simplify coding for heterogeneous systems
  ❖ Enable seamless collaboration of different devices

• Challenges
  ❖ Correctness
  ❖ Efficiency
GPU-CPU Translation

• Translate SPMD code to CPU architecture

• Motivations
  ❖ Easier to aggregate than to parallelize
  ❖ A large cuda code base
  ❖ Simplify coding for heterogeneous systems
  ❖ Enable seamless collaboration of different devices

• Challenges
  ❖ Correctness
  ❖ Efficiency
GPU Execution Model

Thread Block

Stream

Multiprocessor
GPU Execution Model

- Thread Block
- Stream Multiprocessor
GPU Execution Model

Thread Block

Stream Multiprocessor
GPU Execution Model

- Thread Block
- Stream Multiprocessor
- a GPU thread
GPU Execution Model

Thread Block

Stream Multiprocessor

a GPU thread
GPU Execution Model

- Thread Block
- Stream Multiprocessor
- a GPU thread
Poor Ability to Handle Control Divergence

if (...) do something;
else do something else;
Synchronizations in SPMD code

// s[ ]: (volatile) input array
for (i=blockSize/2; i>32; i>>=1) {
    if (tid < i)  s[tid]+=s[tid+i];
    __syncthreads();
}
if (tid < 32) {
    s[tid] += s[tid+32];
    s[tid] += s[tid+16];
    s[tid] += s[tid+8];
    s[tid] += s[tid+4];
    s[tid] += s[tid+2];
    s[tid] += s[tid+1];
}
Synchronizations in SPMD code

```c
// s[ ]: (volatile) input array
for (i=blockSize/2; i>32; i>>=1){
    if (tid < i)  s[tid]+=s[tid+i];
    __syncthreads();  // explicit synchronization
}
if (tid < 32){
    s[tid] += s[tid+32];
    s[tid] += s[tid+16];
    s[tid] += s[tid+8];
    s[tid] += s[tid+4];
    s[tid] += s[tid+2];
    s[tid] += s[tid+1];
}
```
Synchronizations in SPMD code

```c
// s[ ]: (volatile) input array
for (i=blockSize/2; i>32; i>>=1){
    if (tid < i)    s[tid]+=s[tid+i];
    __syncthreads(); // explicit synchronization
}
if (tid < 32){
    s[tid] += s[tid+32];
    s[tid] += s[tid+16];
    s[tid] += s[tid+8];
    s[tid] += s[tid+4];
    s[tid] += s[tid+2];
    s[tid] += s[tid+1];
}
```

(explicit synchronization)

(implicit synchronization
(used to guarantee correctness)
Two Sources of Inefficiency

s[tid] += s[tid+8];
s[tid] += s[tid+4];
s[tid] += s[tid+2];
s[tid] += s[tid+1];
Two Sources of Inefficiency

s[tid] += s[tid+8];

s[tid] += s[tid+4];

s[tid] += s[tid+2];

s[tid] += s[tid+1];
Two Sources of Inefficiency

s[tid] += s[tid+8];
s[tid] += s[tid+4];
s[tid] += s[tid+2];
s[tid] += s[tid+1];
Two Sources of Inefficiency

![Diagram showing the inefficient addition operations]

- `s[tid] += s[tid+8];` with values: 2 4 2 4 2 4 2 4
- `s[tid] += s[tid+4];` with values: 4 8 4 8
- `s[tid] += s[tid+2];` with values: 8 16
- `s[tid] += s[tid+1];`
Two Sources of Inefficiency

\[ s[tid] += s[tid+8]; \]
\[ s[tid] += s[tid+4]; \]
\[ s[tid] += s[tid+2]; \]
\[ s[tid] += s[tid+1]; \]
Two Sources of Inefficiency

• Overly strong synchronizations

• Computation redundancy
  ❖ Less than half of the computation is useful
  ❖ Hidden by the massive parallelism in GPU hardware
  ❖ Redundant computations are in critical path on CPU
Two Sources of Inefficiency

- Overly strong synchronizations
- Computation redundancy
  - Less than half of the computation is useful
  - Hidden by the massive parallelism in GPU hardware
  - Redundant computations are in critical path on CPU
GPU-CPU Translation: State of the Art

- No special treatment to the sources of inefficiency
  - Overly strong synchronizations -> too many loops
  - Redundant Computations -> useless work on critical path
• Key Insight on the Inefficiencies

  ❖ Caused by the uniform treatment to non-uniform threads.

• Key Idea in Our Solution

  ❖ Expose the non-uniformity concisely

  ❖ Enable fine-grained synch. and redundancy removal
One Stone Two Birds

- Synch. Relaxation
  - 3 scheduling algor.
  - 2 code shape opt.
- Thread-Level Partial Redundancy Elimination

Thread-Level Dependence Graph
Thread Level Dependence Graph (TLDG)

- Capturing dependences among instruction instances
- A node: one instance of an instruction
- An edge:
  - Control dependence within a thread
  - Data dependence within or across threads
- Loops are fully unrolled (feasible for most GPU kernels)
  - A loop unfeasible for unrolling is put as a single node
TLDG: an example

//node 0
tempBuf[tid] = sdata[tid+4];
//node 1
sdata[tid] += tempBuf[tid];
//node 2
tempBuf[tid] = sdata[tid+2];
//node 3
sdata[tid] += tempBuf[tid];
//node 4
tempBuf[tid] = sdata[tid+1];
//node 5
sdata[tid] += tempBuf[tid];
TLDG: an example

//node 0
tempBuf[tid] = sdata[tid+4];

//node 1
sdata[tid] += tempBuf[tid];

//node 2
tempBuf[tid] = sdata[tid+2];

//node 3
sdata[tid] += tempBuf[tid];

//node 4
tempBuf[tid] = sdata[tid+1];

//node 5
sdata[tid] += tempBuf[tid];
TLDG: an example

//node 0
tempBuf[tid] = sdata[tid+4];
//node 1
sdata[tid] += tempBuf[tid];
//node 2
tempBuf[tid] = sdata[tid+2];
//node 3
sdata[tid] += tempBuf[tid];
//node 4
tempBuf[tid] = sdata[tid+1];
//node 5
sdata[tid] += tempBuf[tid];
TLDG: an example

//node 0
tempBuf[tid] = sdata[tid+4];
//node 1
sdata[tid] += tempBuf[tid];
//node 2
tempBuf[tid] = sdata[tid+2];
//node 3
sdata[tid] += tempBuf[tid];
//node 4
tempBuf[tid] = sdata[tid+1];
//node 5
sdata[tid] += tempBuf[tid];

Intra-thread edges are not shown for better readability
Properties of TLDG

- Captures instance-level dependences
  - Rather than traditional instruction-level dependences
- Bounded graph size: #threads per block
- Lays the foundation for fine-grained analysis and optimizations
  - E.g., thread-level dependences and redundancies
A Unified Solution Based on TLDG

• Works on GPU code represented by TLDG

• Thread-Level Partial Redundancy Removal
  ❖ Converted to dead-code elimination through TLDG

• Instance-level instruction scheduling
  ❖ Breath-first, depth-first, length-first

• Code shape optimization
  ❖ Resolving ID-dep conditional statements
  ❖ Code compression through graph pattern matching
A Unified Solution Based on TLDG

- Works on GPU code represented by TLDG
- Thread-Level Partial Redundancy Removal
  - Converted to dead-code elimination through TLDG
- Instance-level instruction scheduling
  - Breath-first, depth-first, length-first
- Code shape optimization
  - Resolving ID-dep conditional statements
  - Code compression through graph pattern matching
Specialty of TLDG-Based Scheduling

- Place an order to preserve the dependences among instruction instances (barriers eliminated)
- Automatically relax overly strong synchronizations

```
2 4 2 4 2 4 2 4
```

- Different from traditional instruction scheduling
  - Instruction of the largest weight scheduled first?
Specialty of TLDG-Based Scheduling

- Place an order to preserve the dependences among instruction instances (barriers eliminated)
- Automatically relax overly strong synchronizations

- Different from traditional instruction scheduling
  - Instruction of the largest weight scheduled first?
Specialty of TLDG-Based Scheduling

- Place an order to preserve the dependences among instruction instances (barriers eliminated)

- Automatically relax overly strong synchronizations

- Different from traditional instruction scheduling
  - Instruction of the largest weight scheduled first?
Breadth-First Scheduling

• Example

```
T0     T1     T2     T3
```

• Find all nodes with no incoming edges to form a group
Breadth-First Scheduling

• Example

• Find all nodes with no incoming edges to form a group
Breadth-First Scheduling

• Example

```
T0     T1     T2     T3
```

• Find all nodes with no incoming edge to form a group

• Schedule the group and remove the nodes and outgoing edges
Breadth-First Scheduling

- Example

- Find all nodes with no incoming edge to form a group

- Schedule the group and remove the nodes and outgoing edges
Breadth-First Scheduling

• Example

• Repeat the process until all nodes are scheduled
Breadth-First Scheduling

• Example

• Repeat the process until all nodes are scheduled
Breadth-First Scheduling

- Example

- Repeat the process until all nodes are scheduled
Breadth-First Scheduling

• Example

\[ \text{T0} \quad \text{T1} \quad \text{T2} \quad \text{T3} \]

• Repeat the process until all nodes are scheduled
Breadth-First Scheduling

- Example

- Repeat the process until all nodes are scheduled
Breadth-First Scheduling

• Example

Repeat the process until all nodes are scheduled
Breadth-First Scheduling

• Example

• Repeat the process until all nodes are scheduled
Breadth-First Scheduling

• Example

T0    T1    T2    T3

• Repeat the process until all nodes are scheduled
Breadth-First Scheduling

• Example

• Repeat the process until all nodes are scheduled
Breadth-First Scheduling

- Tie breaking in the group
  - Schedule nodes with smaller thread ID first
- Good spacial locality
  - Adjacently scheduled nodes usually come from the same level
- Poor temporal locality
  - No dependences among nodes in the same group
Depth-First Scheduling

• Example

• Find the node with the largest weight
Depth-First Scheduling

- Example

Find the node with the largest weight
Depth-First Scheduling

- Example

- Find the node of largest weight

- Schedule it, remove the node and its outgoing edge
Depth-First Scheduling

• Example

Find the node of largest weight

Schedule it, remove the node and its outgoing edge
Depth-First Scheduling

• Example

Find the node of largest weight

Schedule it, remove the node and its outgoing edge

move to the next node in the same thread if possible
Depth-First Scheduling

- Example

Find the node of largest weight

Schedule it, remove the node and its outgoing edge

move to the next node in the same thread if possible
Depth-First Scheduling

• Example

• Find the node of largest weight

• Schedule it, remove the node and its outgoing edge

• move to the next node in the same thread if possible
Depth-First Scheduling

- Example

- Find the node of largest weight

- Schedule it, remove the node and its outgoing edge

- move to the next node in the same thread if possible
Length-First Scheduling

• Example

• Similar as Depth-First scheduling but explore inter-thread dependences

• Schedule the longest path with no interruption

• Maximally leverage temporal locality
Length-First Scheduling

- Example

- Similar as Depth-First scheduling but explore inter-thread dependences

- Schedule the longest path with no interruption

- Maximally leverage temporal locality
Length-First Scheduling

• Example

• Similar as Depth-First scheduling but explore inter-thread dependences

• Schedule the longest path with no interruption

• Maximally leverage temporal locality
Length-First Scheduling

• Example

• Similar as Depth-First scheduling but explore inter-thread dependences

• Schedule the longest path with no interruption

• Maximally leverage temporal locality
Code Shape Optimization

- A TLDG usually consists of some repetitive patterns
- Use a loop surrounding the patterns to reduce code size
Experiment Setup

• 5 CUDA programs
  ❖ Reduction, Transpose, SortingNetworks (SDK)
  ❖ CG (a rewritten NPB openmp benchmark)
  ❖ Nw (Rodinia)

• Environment
  ❖ quad-core Intel Xeon E5460
  ❖ Linux 2.6.33
  ❖ GCC 4.1.2
Results

With Redundancy Removal

- Up to 14 times speedup achieved for CG
- Slight slowdown for transpose
- Scheduling orders matter
Results

With Redundancy Removal

• Up to 14 times speedup achieved for CG
• Slight slowdown for transpose
• Scheduling orders matter
• Up to 14 times speedup achieved for CG

• Slight slowdown for transpose

• Scheduling orders matter
Results

With Redundancy Removal

• Up to 14 times speedup achieved for CG
• Slight slowdown for transpose
• Scheduling orders matter
Results

With Redundancy Removal

• Up to 14 times speedup achieved for CG
• Slight slowdown for transpose
• Scheduling orders matter
Results

With Redundancy Removal

<table>
<thead>
<tr>
<th></th>
<th>Splitting-Oriented</th>
<th>Breadth-First</th>
<th>Depth-First</th>
<th>Length-First</th>
</tr>
</thead>
<tbody>
<tr>
<td>reduction</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>cg</td>
<td>14</td>
<td>14</td>
<td>14</td>
<td>14</td>
</tr>
<tr>
<td>sortingNetwork</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>transpose</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>nw</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

Speedup
Results

With Redundancy Removal

![Graph showing speedup with redundancy removal for different algorithms and benchmarks.](image)
Results

With Redundancy Removal

Without Redundancy Removal
Results

- Redundancy removal plays an important role in CG and reduction
- There are no redundancies in the other three benchmarks
Contributions

• Identified two sources of inefficiency in GPU-CPU translation
  ❖ Overly strong synchronizations
  ❖ Redundant computations

• A set of TLDG-based techniques
  ❖ Three scheduling algorithms
  ❖ Redundancy removal
  ❖ Code shape optimization

• The first comprehensive solution to both inefficiency problems
Thanks!