A Cross-Input Adaptive Framework for GPU Program Optimizations

Yixun Liu, Eddy Z. Zhang, Xipeng Shen

Computer Science Department
The College of William & Mary
Outline

- **GPU overview**
- G-Adapt Framework
- Evaluation
- Related & Future Work
- Conclusion
GPU (Graphics Processing Unit)

- **Architecture**
  - SIMD parallel
  - Multithreaded
  - Many core

- **Feature**
  - Tremendous computational horsepower
  - High mem badwidth

- **Applications**
  - Traditional graphic rendering
  - Emerging: general data parallel computing
Programming GPU

- High level model
  - Abstraction to multithread platform
  - C-like programming
  - No explicit mapping to graphics rendering
  - E.g, CUDA, Brook+, openCL

- NVIDIA CUDA
  - Kernel func. on GPU
  - Threads->blocks->grids
Optimization Challenges

• Goal
  ▫ Maximize throughput
    • Increase occupancy, reduce latency, dynamic instr.

• Difficulties
  ▫ Hard to predict optimization effects
    • Non-linearity, coupling, undisclosed CUDA details
  ▫ GPU hardware complexities
    • Limits: 512 threads per block, 768 threads per SM, etc
    • Various types of memories: constant, texture, etc
  ▫ Input sensitivity
Matrix-Vector Multiplication

Mat Size = 64x32768

Mat Size = 32768x64
Outline

• GPU Overview
• **G-Adapt Framework**
• Evaluation
• Related & Future Work
• Conclusion
G-ADAPT

• Empirical search-based optimization
  ▫ Three obstacles to address
    • Construction of the optimization space
    • Space pruning
    • Cross-input adaptation
G-ADAPT: Overview

- Source-to-source compiler
- Cross-input adaptation
- Automatic search & transformations
- Easy integration of user knowledge through pragmas
G-ADAPT Pragmas

- Supports a programmer-compiler synergy

- Covers 2 levels of optimizations
  - Execution configurations
    - E.g, thread block dimensions
  - Code transformations
    - E.g, loop tile size, unrolling levels
Pragma Examples

```
#pragma erange 64, 512, 2
#define BLKSZ 256

#pragma lpur_lrange 0, min(BLKSZ, 16), 2
For (i=1; i < BLKSZ; i++) {
    ......  
}
```

BLKSZ ranges from 64 to 512, increasing exponentially, doubling each time.

The loop unrolling level ranges from 0 to min(BLKSZ, 16), increasing linearly, each time by 2.
Stage I: Search & Collect

- Code with pragmas & inputs
  - G-ADAPT compiler
    - Optimized GPU code
      - Performance Calibrator
        - Performance
          - Perf. DB
  - Optimization Parameters
    - Optimization Agent
G-ADAPT Compiler

- Two functionalities
  - Recognize opt. space
  - Program transformations
- Based on Cetus [Purdue Univ]
- Source-to-source
- GPU extensions
- Support G-ADAPT pragmas
Performance Calibrator

- Invokes CUDA compiler and runs the executable
- Collect running time and GPU occupancy
Optimization Agent

- Determines the optimization param. to try next
- Uses hill climbing to overcome space explosion problem
G-ADAPT: Overview

**Stage 1**
- Code with pragmas & input
- Empirical search & data collection
- <Input, best optimizations>

**Stage 2**
- Pattern recognition & code generation
- Optimized input-adaptive GPU program
Stage II: PR & Code Gen

- Pattern recognizer
  - Recognize
    - input \(\leftrightarrow\) best parameters
  - Regression Trees with Least Mean Square

- Options for code generator
  - Multiple versions
  - JIT compilers
  - Linker
Outline

- GPU overview
- G-Adapt Framework
- **Evaluation**
- Related & Future Work
- Conclusion
Evaluation - Platform

- **GPU: NVIDIA GeForce 8800 GT**
  - 14 multiprocessors (MP), 112 cores
  - 512M global mem, 16KB shared mem/MP, 8192 registers/MP
  - CUDA 2.0

- **Host: Intel Xeon 3.6 GHz, Suse Linux 2.6.22**
# Benchmarks

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Description</th>
<th># of Inputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Convolution</td>
<td>Convolution filter of a 2D signal</td>
<td>10</td>
</tr>
<tr>
<td>matrixMul</td>
<td>Dense matrix multiplication</td>
<td>9</td>
</tr>
<tr>
<td>mvMul</td>
<td>Dense matrix vector multiplication (by Fujimoto)</td>
<td>15</td>
</tr>
<tr>
<td>reduction</td>
<td>Sum of an array</td>
<td>15</td>
</tr>
<tr>
<td>scalarProd</td>
<td>Scalar products</td>
<td>7</td>
</tr>
<tr>
<td>transpose</td>
<td>Matrix transpose</td>
<td>18</td>
</tr>
<tr>
<td>Transpose-co</td>
<td>Coalescing matrix transpose</td>
<td>18</td>
</tr>
</tbody>
</table>
# Training and Prediction

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Training iterations</th>
<th>Training time (s)</th>
<th>Prediction accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>convolution</td>
<td>200</td>
<td>2825</td>
<td>100%</td>
</tr>
<tr>
<td>matrixMul</td>
<td>196</td>
<td>2539</td>
<td>100%</td>
</tr>
<tr>
<td>mvMul</td>
<td>124</td>
<td>124</td>
<td>93.3%</td>
</tr>
<tr>
<td>reduction</td>
<td>75</td>
<td>29</td>
<td>80%</td>
</tr>
<tr>
<td>scalarProd</td>
<td>93</td>
<td>237</td>
<td>100%</td>
</tr>
<tr>
<td>transpose</td>
<td>54</td>
<td>1639</td>
<td>100%</td>
</tr>
<tr>
<td>Transpose-co</td>
<td>54</td>
<td>631</td>
<td>100%</td>
</tr>
</tbody>
</table>
Matrix Vector Multiplication

Best Parameter V.S. Input

Optimal Block Size

Speed up V.S. Input
Speed up over default

![Bar chart showing speed up over default for different operations]

- convolution
- matrixMul
- mvMul
- reduction
- scalarProd
- transpose
- transpose-co

Minimal Speed up
Maximal Speed up
Speed up over default

![Speed Up Chart]

- convolution
- matrixMul
- mvMul
- reduction
- scalarProd
- transpose
- transpose-co
Speed up over default

![Speed up over default chart]

http://www.cs.wm.edu/~xshen/Publications/ipdps09.pdf
Outline

- GPU overview
- G-Adapt Framework
- Evaluation
- **Related & Future Work**
- Conclusion
Related Work

- Ryoo+: CGO’08
  - Efficiency and utilization model for search
  - Manual transformation; assumptions on applications
- Baskaran+: ICS’08
  - Polyhedral model for optimizing memory access
  - Limited to affine loop nests

Features of G-Adapt
First generally applicable framework
Cross-input adaptation
Future Work

• More optimization options
  ▫ Algorithm Selection
  ▫ Memory optimization
  ▫ Divergence Elimination

• General Support
  ▫ Cetus – ANSI C compiler
    • Non ANSI C features, C++
  ▫ CUDA built-in types
    • E.g. float4, texture and etc
Outline

- GPU overview
- G-Adapt Framework
- Evaluation
- Related & Future Work
  - Conclusion
Conclusion

• A general tool for GPU optimization
• Cross-input adaptation
• Synergy between compilers and programmers
• Alternative of manual tuning, enabling easy adaptation across architectures
Acknowledgement

• Cetus authors at Purdue
  ▫ Group led by Eigenmann and Midkiff
• John Owens
• NVIDIA
  ▫ donation of device
• NSF grants
Thank you!