A GPU-based Algorithm-specific Optimization for High-performance Background Subtraction

Chulian Zhang, Hamed Tabkhi, Gunar Schirner
Outline

• Motivation
• Background
• Related Work
• Approach
  – General Optimization
  – Algorithm-specific Optimization
  – Shared Memory Optimization
• Quality Exploration
• Conclusion
Motivation

• Computer Vision
  – Huge market
    • Surveillance, ADAS, HCI, traffic monitoring
  – Vision algorithm properties
    • Embarrassingly parallel
    • Demand high performance

• Possible Solutions for CV
  – CPU
    • low performance, high power
  – FPGA
    • High performance, low power
    • Hard to implement
  – GPU
    • High performance, relatively low power
    • Massively parallel cores
    • Suitable for throughput oriented applications
    • Easy to program
• Motivation
• Background
  • Related Work
  • Approach
    – General Optimization
    – Algorithm-specific Optimization
    – Shared Memory Optimization
• Quality Exploration
• Conclusion
Background Subtraction

• Background subtraction
  – Primary vision kernel
  – Frequently used in many market

• Mixture of Gaussians (MoG)
  – Gaussian background model for each pixel
    • Multiple Gaussians
    • Weight, mean, standard deviation
  – Adaptive
    • Operate on single video frame, recursively update the model
  – Embarrassingly parallel

• Advantages
  • Learning based algorithm
  • Deals better with gradual variations
GPU Architecture

• Streaming Multiprocessor
  – Many cores
    • Single inst. multiple thread (SIMT)
  – Large RF
  – Shared memory
    • On-chip, fast
    • Communication & sync

• Global Memory

• Observation
  – GPU architecture is fundamentally different from CPU
  – Application optimized for CPU single thread is not suitable for GPU

• Demand for algorithm-specific optimization
  – Adjust vision algorithms to get maximum performance
Outline

• Motivation
• Background

• Related Work
  • Approach
    – General Optimization
    – Algorithm-specific Optimization
    – Shared Memory Optimization
  • Quality Exploration
• Conclusion
Related Work

• Focus on general GPU optimizations
  – Memory coalescing  [Kumar, 13] [Pham, 10] [Li, 12]
  – Overlapped execution  [Kumar, 13] [Pham, 10]
  – Shared memory  [Kumar, 13] [Pham, 10] [Li, 12]
    • No detail performance analysis
  – Limited performance speedup
    • 20x

• Algorithm optimizations  [Azmat, 12]
  – Reduce costly operations
    • Quality loss

• [Kumar, 13], “Real-time Moving Object Detection Algorithms on High-resolution Videos using GPUs”
• [Pham, 10], “GPU Implementation of Extended Gaussian Mixture Model for Background Subtraction”
• [Poremba, 10], “Accelerating adaptive background subtraction with GPU and CEBA architecture”
• [Li, 12], “Three-level GPU accelerated Gaussian Mixture Model for Background Subtraction”
• OpenCV
• [Azmat, 12], “Accelerating adaptive background modeling on low-power integrated GPUs ”
Outline

• Motivation
• Background
• Related Work
• Approach
  – General Optimization
    – Algorithm-specific Optimization
    – Shared Memory Optimization
• Quality Exploration
• Conclusion
Experiment Setup

• Hardware Platform
• OS
  – RHEL 6.2
• Algorithm Setup
  – 3 Gaussians
  – Double precision
• Baseline
  – CPU
  – Single thread
  – -O3 optimization
• Input
  – HD frames (1080 x 1920)

<table>
<thead>
<tr>
<th></th>
<th>CPU</th>
<th>GPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processor</td>
<td>Intel Xeon E5-2620</td>
<td>nVidia Tesla C2075</td>
</tr>
<tr>
<td>Cores</td>
<td>6</td>
<td>448</td>
</tr>
<tr>
<td>Frequency</td>
<td>2.5 GHz</td>
<td>1.15 GHz</td>
</tr>
<tr>
<td>GFLOPS</td>
<td>120.3</td>
<td>1030</td>
</tr>
<tr>
<td>Cache</td>
<td>L2 (256K)</td>
<td>L1(16/48K)</td>
</tr>
<tr>
<td></td>
<td>L3 (15M)</td>
<td>L2 (768K)</td>
</tr>
<tr>
<td>Memory BW</td>
<td>12.8 GB/s</td>
<td>144 GB/s</td>
</tr>
</tbody>
</table>

Embedded Systems Laboratory, Northeastern
General Optimizations

• Memory Coalescing
  – Non-coalesced
    • Locality within each thread
    • Optimized for CPU single-thread
  – Coalesced
    • Locality across different threads
    • GPU massively-parallel threads
      – Request will be coalesced

• Overlapped Execution
  – Overlap comm. & computation

• Result
Outline

• Motivation
• Background
• Related Work
• Approach
  – General Optimization
  – Algorithm-specific Optimization
    – Shared Memory Optimization
• Quality Exploration
• Conclusion
Algorithm-specific Optimization

• General Optimization is not enough
  – GPU is not fully utilized

• Algorithm-specific optimization
  – Tune the algorithm to better fit the underlying architecture

• Examples
  – Vision algorithm has many branches
  – Branches reduce GPU utilization

• Solutions
  – Remove branches, even at the cost of extra computation
  – Overall performance will be better
Branch Reduction

• Gaussian background checking (CPU code)
  1. Sorting
  2. Check from $\text{HighestRank}$
  3. Stop when a match happens

• Problem
  – New pixel has higher chance to match with higher rank Gaussian
  – CPU-style optimization
  – Add branches

• GPU code
  – Remove sorting
  – Check all components
  – Preserve correctness

• Result
  – Branch number reduced
  – Branch efficiency increased
Source-level Predicated Execution

- Predicated instruction
  - Generated by GPU compiler
- Gaussian update
  - Compiler cannot detect it
  - Source-level predicated
    - Use flag to decide the effect
    - Remove branch
- Result

Algorithm 4 non-Predicated Execution

1: for $k = 0$ to $\text{numGau}$ do
2:     if $\text{match}$ then
3:         $w[k] = \alpha \cdot w[k] + (1 - \alpha)$
4:         $\text{tmp} = (1 - \alpha)/w[k]$
5:         $m[k] = f(\text{tmp})$
6:         $sd[k] = g(\text{tmp})$
7:     else
8:         $w[k]* = \alpha$
9:     end if
10: end for

Algorithm 5 Predicated Execution

1: for $k = 0$ to $\text{numGau}$ do
2:     $w[k] = \alpha \cdot w[k] + \text{match} \cdot (1 - \alpha)$
3:     $\text{tmp} = (1 - \alpha)/w[k]$
4:     $m[k] = (1 - \text{match}) \cdot m[k] + \text{match} \cdot f(\text{tmp})$
5:     $sd[k] = (1 - \text{match}) \cdot sd[k] + \text{match} \cdot g(\text{tmp})$
6: end for

Fig. 7: Performance for Alg-specific Optimization

(a) Branch
(b) Memory
(c) Reg. & Occup.

Algorithm 5 Predicated Execution

1: for $k = 0$ to $\text{numGau}$ do
2:     $w[k] = \alpha \cdot w[k] + \text{match} \cdot (1 - \alpha)$
3:     $\text{tmp} = (1 - \alpha)/w[k]$
4:     $m[k] = (1 - \text{match}) \cdot m[k] + \text{match} \cdot f(\text{tmp})$
5:     $sd[k] = (1 - \text{match}) \cdot sd[k] + \text{match} \cdot g(\text{tmp})$
6: end for
Register Usage Reduction

• Solution
  – Reduce registers per thread

• Approach
  – Cascade arithmetic operations
  – Remove intermediate variables
  – Guided by source code
    • Compiler cannot find the minimum # register that yield best performance

• Result

![Bar chart showing occupancy and performance before and after the solution](chart.png)
Outline

• Motivation
• Background
• Related Work
• Approach
  – General Optimization
  – Algorithm-specific Optimization
  – Shared Memory Optimization
• Quality Exploration
• Conclusion
Shared Memory Optimization

• Frame-based operation
  – One frame each time
    • Read / write Gaussian parameters from / to global memory

• Shared Memory
  – Small size
  – Need data reuse

• Window-based operation
  – Split one frame into smaller windows
    • Window size decided by shared memory size
  – Process the same window across many frames
    • Frame group
    • Increase data reuse
  – Shift to next frame group after finishing all windows
Shared Memory Optimization

- **Positive**
  - Memory access goes to shared memory

- **Negative**
  - Latency per frame is increased

- **Optimal group size?**
  - 8 frames
  - Occupancy decreases with increasing group size
Outline

• Motivation
• Background
• Related Work
• Approach
  – General Optimization
  – Algorithm-specific Optimization
  – Shared Memory Optimization
• Quality Exploration
• Conclusion
Quality Exploration

- Quality Assessment
  - Ground Truth
    - CPU result with double precision
  - Multi-Scalar Structural SIMilarity (MS-SSIM)
    - Quantify the structural similarity between two images

- Quality Result
  - Background result is always 99%
  - Foreground
    - Slightly drop due to algorithm tuning

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>Background</td>
<td>99%</td>
<td>99%</td>
<td>99%</td>
<td>99%</td>
<td>99%</td>
<td>99%</td>
</tr>
<tr>
<td>Foreground</td>
<td>99%</td>
<td>99%</td>
<td>96%</td>
<td>97%</td>
<td>97%</td>
<td>95%</td>
</tr>
</tbody>
</table>
Outline

• Motivation
• Background
• Related Work
• Approach
  – General Optimization
  – Algorithm-specific Optimization
  – Shared Memory Optimization
• Quality Exploration

→ Conclusion
Conclusion

• GPU is perfect for Computer Visions
  – massively parallel

• Existing vision algorithms optimized for CPU

• Demand for optimal performance from GPU
  – Understand the vision algorithms
  – Tune algorithm to fit architecture

• Future work
  – Apply same principles to other vision algorithms
  – Develop & evaluate optimizations for embedded GPU
Thank you!

Q & A