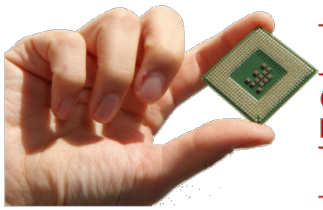
A GPU-based Algorithm-specific Optimization for High-performance Background Subtraction

Chulian Zhang, Hamed Tabkhi, Gunar Schirner

Northeastern



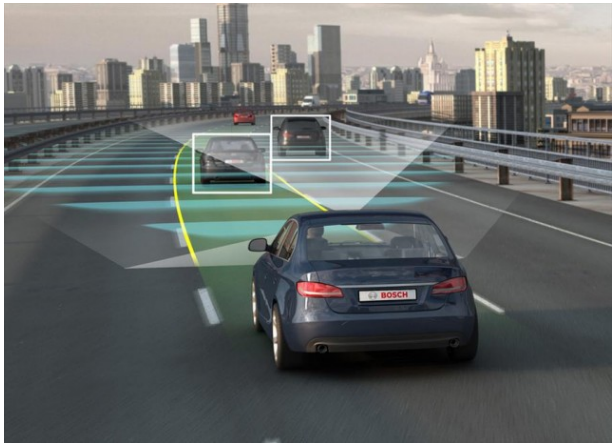
Embedded Systems Laboratory

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

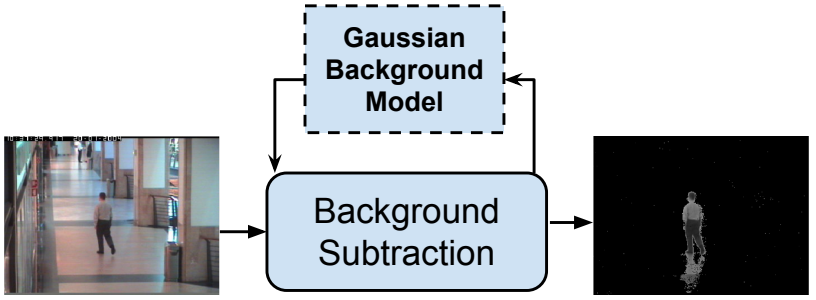


Outline

- 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



Number of Gaussian Components

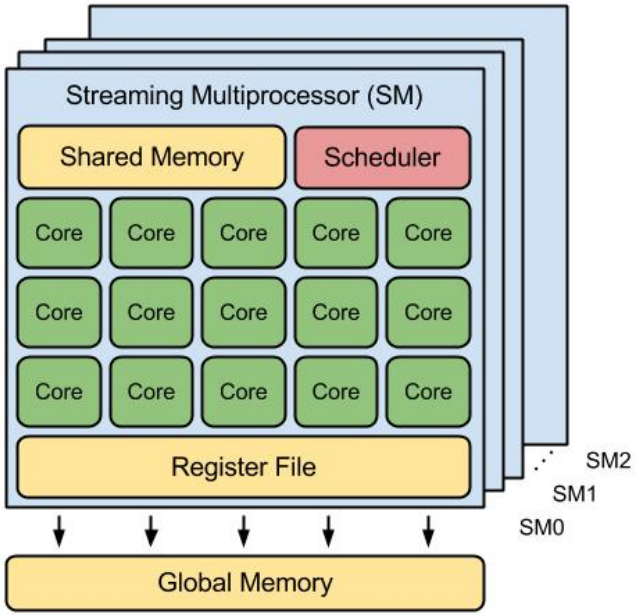
$$\sum_{i=1}^{K} \omega_{i,t} \cdot \eta(u; \mu_{i,t}, \sigma_{i,t})$$

Weight

Intensity mean

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-timed 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

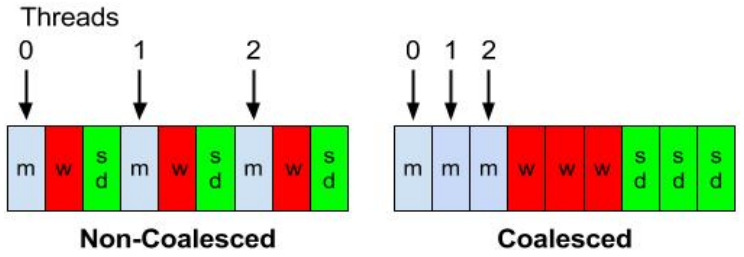
Experiment Setup

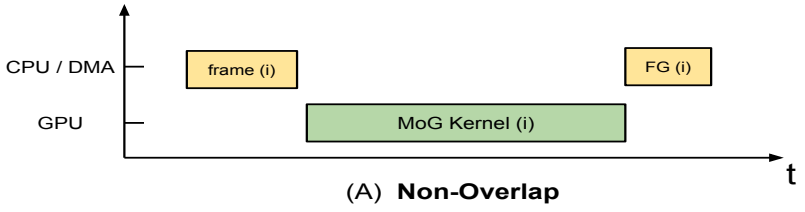
- Hardware Platform
- OS
  - RHEL 6.2
- Algorithm Setup
  - 3 Gaussians
  - Double precision
- Baseline
  - CPU
  - Single thread
  - -O3 optimization
- Input
  - HD frames (1080 x 1920)

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

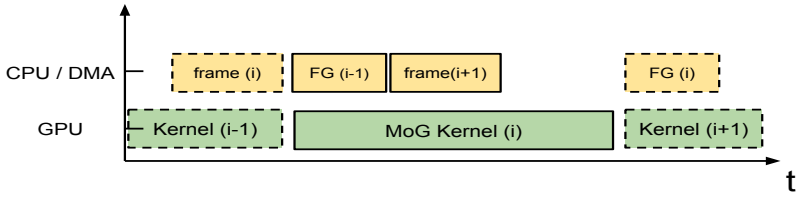
General Optimizations

- 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

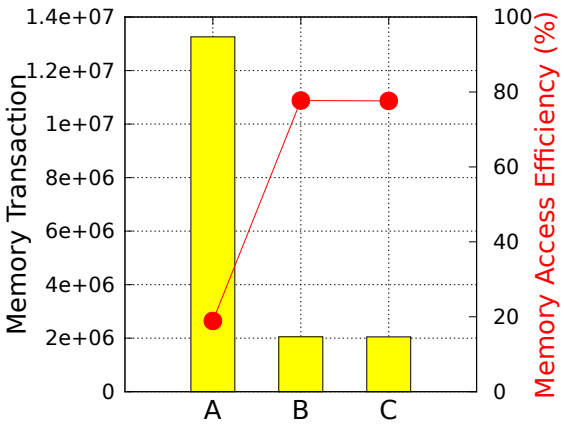




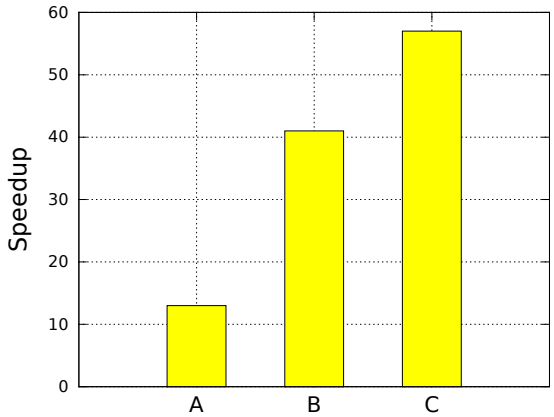
(A) **Non-Overlap**



(B) **Overlap**



Memory Performance



Speedup

	A	B	C
Base Implementation	✔	✔	✔
Memory Coalescing		✔	✔
Overlapped Execution			✔

Outline

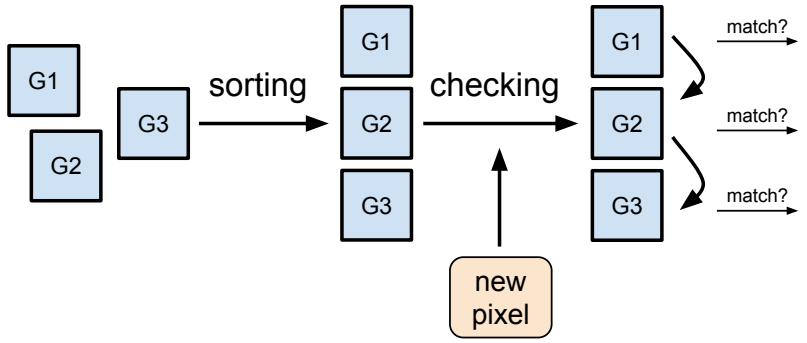
- Motivation
- Background
- Related Work
- Approach
  - General Optimization
  - **Algorithm-specific Optimization**
  - Shared Memory Optimization
- Quality Exploration
- Conclusion

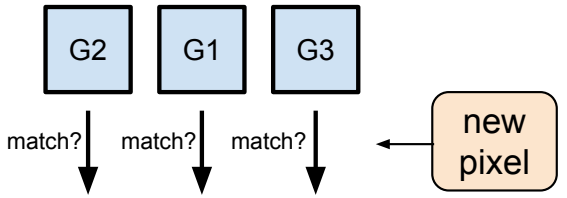
Algorithm-specific Optimization

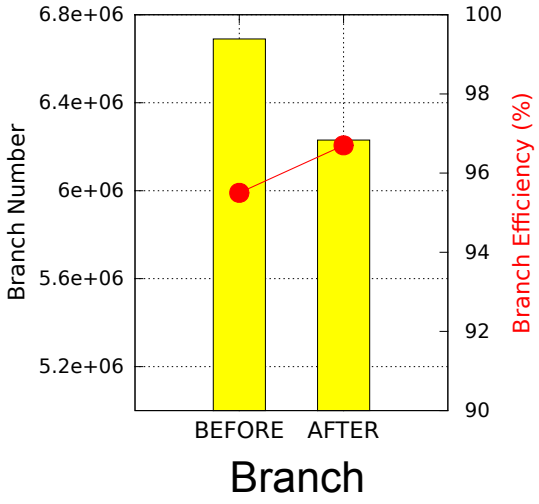
- General Optimization is not enough
  - GPU is not fully utilized
- Algorithm-specific optimization
  - Tune the algorithm to better fit the underlying architecture
- Examples
  - Vision algorithm has many branches
  - Branches reduce GPU utilization
- Solutions
  - Remove braches, even at the cost of extra computation
  - Overall performance will be better

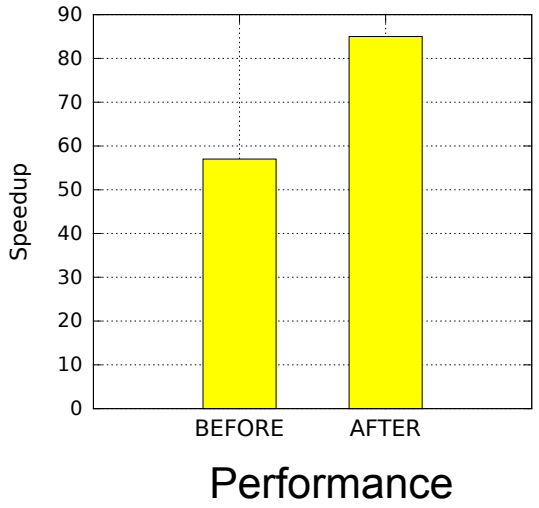
Branch Reduction

- Gaussian background checking (CPU code)
  1. Sorting
  2. Check from *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

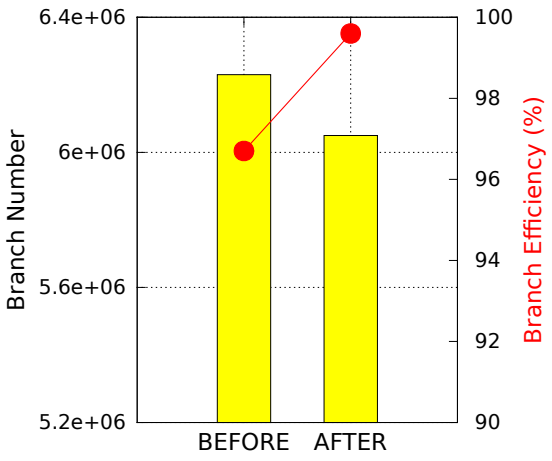
- Predicated instruction
  - Generated by GPU compiler
- Gaussian update
  - Compiler cannot detect it
  - Source-level predicated
    - Use *flag* to decide the effect
  - Remove branch
- Result

Algorithm 4 non-Predicated Execution

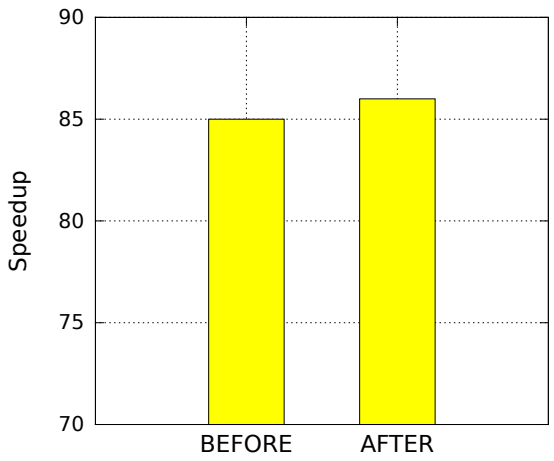
```
1:  for k = 0 to numGau do
2:      if match then
3:          w[k] = α * w[k] + (1 − α)
4:          tmp = (1 − α)/w[k]
5:          m[k] = f(tmp)
6:          sd[k] = g(tmp)
7:      else
8:          w[k]* = α
9:      end if
10: end for
```

Algorithm 5 Predicated Execution

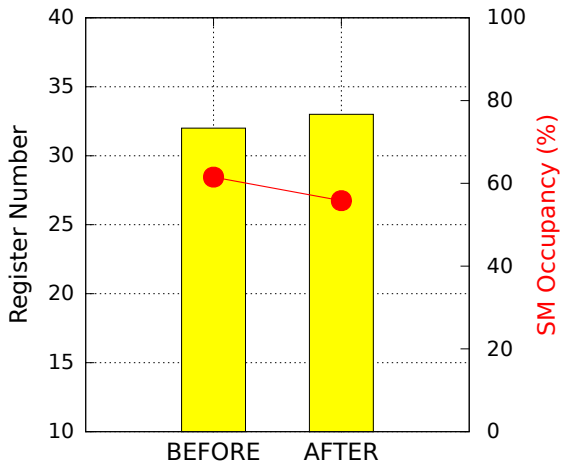
```
1: for k = 0 to numGau do
2:     w[k] = α * w[k] + match * (1 − α)
3:     tmp = (1 − α)/w[k]
4:     m[k] = (1 − match) * m[k] + match * f(tmp)
5:     sd[k] = (1 − match) * sd[k] + match * g(tmp)
6: end for
```



Branch



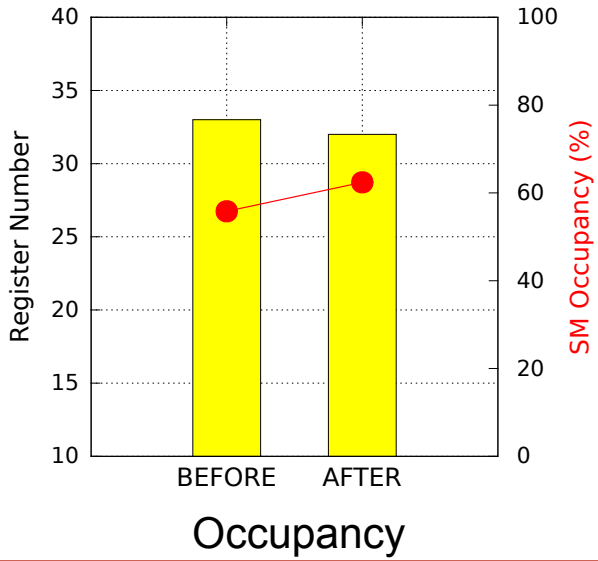
Performance



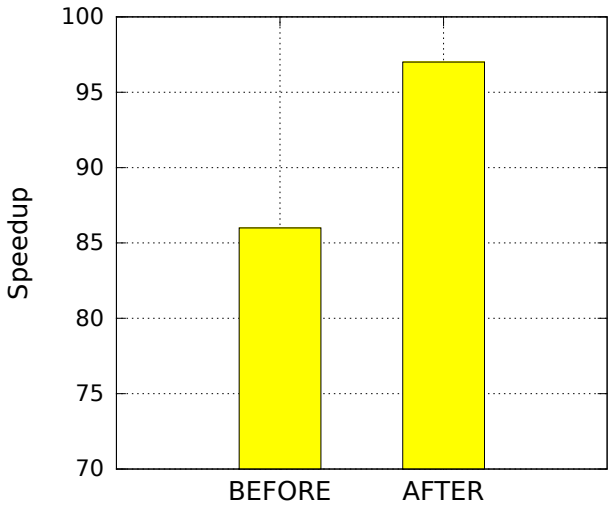
Register Usage

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



Occupancy



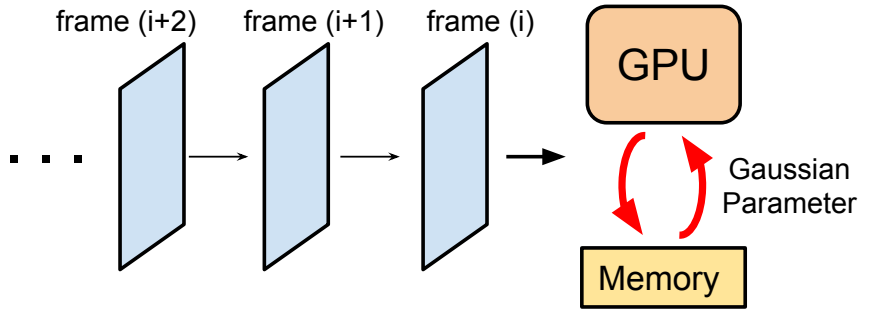
Performance

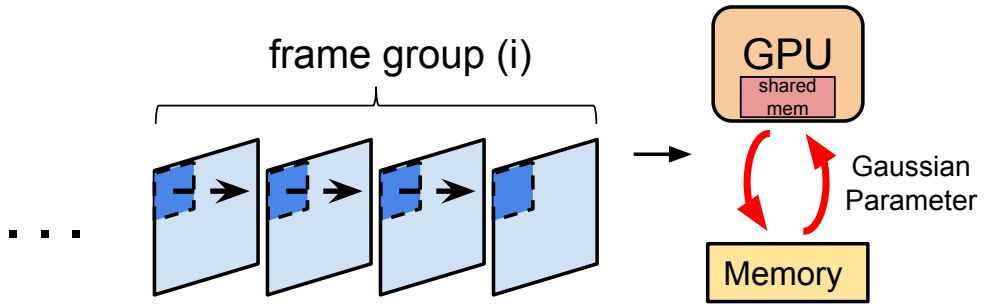
Outline

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

Shared Memory Optimization

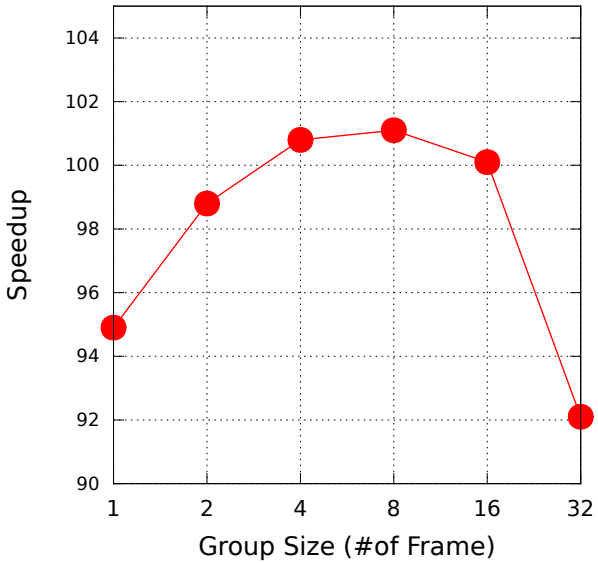
- Frame-based operation
  - One frame each time
    - Read / write Gaussian parameters from / to global memory
- Shared Memory
  - Small size
  - Need data reuse
- Window-based operation
  - Split one frame into smaller windows
    - Window size decided by shared memory size
  - Process the same window across many frames
    - Frame group
    - Increase data reuse
  - Shift to next frame group after finishing all windows

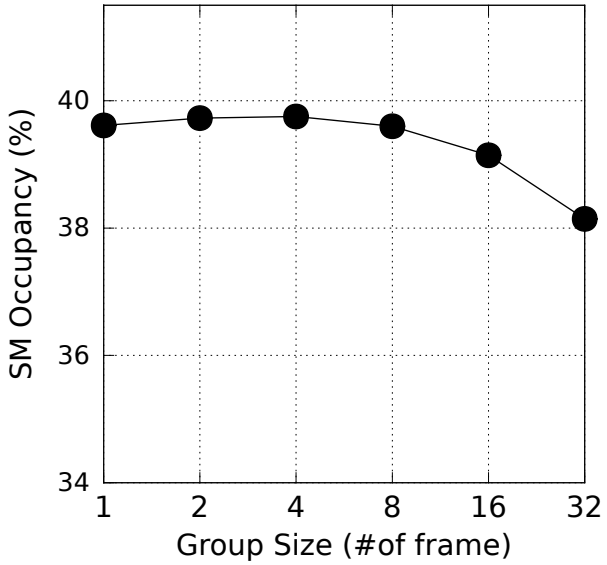




Shared Memory Optimization

- 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

	A	B	C	D	E	F
Background	99%	99%	99%	99%	99%	99%
Foreground	99%	99%	96%	97%	97%	95%

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