SAGE: Self-Tuning Approximation for Graphics Engines

Mehrzad Samadi¹, Janghaeng Lee¹, D. Anoushe Jamshidi¹, Amir Hormati², and Scott Mahlke¹

University of Michigan¹, Google Inc.²

December 2013
Approximate Computing

• Different domains:
  — Machine Learning
  — Image Processing
  — Video Processing
  — Physical Simulation
  — ...

Less work  ➔  Higher performance
  Lower power consumption
Ubiquitous Graphics Processing Units

• Wide range of devices

• Mostly regular applications

• Works on large data sets

Good opportunity for automatic approximation
SAGE Framework

• Simplify or skip processing
• Computationally expensive
• Lowest impact on the output quality

Self-Tuning Approximation on Graphics Engines

• Write the program once
• Automatic approximation
• Self-tuning dynamically
Overview
SAGE Framework

Input Program → Static Compiler → Approximation Methods → Approximate Kernels → Runtime System → Tuning Parameters
Static Compilation

Evaluation Metric
Target Output Quality (TOQ)
CUDA Code

Atomic Operation
Data Packing
Thread Fusion

Approximate Kernels
Tuning Parameters
Runtime System

CPU

GPU

Preprocessing

Tuning

Execution

Calibration

Tuning

T

T + C

T + 2C

Quality

TOQ

Speedup

Time
Runtime System

CPU

GPU

Preprocessing

Tuning

Execution

Calibration

Time

Quality

TOQ

Speedup

Tuning

T

T + C

T
Approximation Methods
Approximation Methods

- Evaluation Metric
- Target Output Quality (TOQ)
- CUDA Code

- Atomic Operation
- Data Packing
- Thread Fusion

Approximate Kernels
Tuning Parameters
• Atomic operations update a memory location such that the update appears to happen atomically

```c
// Compute histogram of colors in an image
__global__ void histogram(int n, int* color, int* bucket)
int tid = threadIdx.x + blockDim.x * blockIdx.x;
int nThreads = blockDim.x * gridSize.x;
for (int i = tid; tid < n; tid += nThreads)
    int c = colors[i];
    atomicAdd(&bucket[c], 1);
```
Atomic Operations

Threads
Atomic Operations

Threads
Atomic Operations

Threads
Atomic Operations

Threads
Atomic Operation Tuning

// Compute histogram of colors in an image
__global__ void histogram(int n, int* color, int* bucket)
{
    int tid = threadIdx.x + blockDim.x * blockIdx.x;
    int nThreads = gridDim.x * blockDim.x;
    for (int i = tid; tid < n; tid += nThreads)
    {
        int c = colors[i];
        atomicAdd(&bucket[c], 1);
    }
}
Atomic Operation Tuning

• SAGE skips one iteration per thread
• To improve the performance, it drops the iteration with the maximum number of conflicts
Atomic Operation Tuning

- SAGE skips one iteration per thread
- To improve the performance, it drops the iteration with the maximum number of conflicts

It drops 50% of iterations
Atomic Operation Tuning

Drop rate goes down to 25%

T0

T32

T64
## Dropping One Iteration

<table>
<thead>
<tr>
<th>Iteration No.</th>
<th>Conflicts</th>
<th>After optimization</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2</td>
<td>CD 0</td>
</tr>
<tr>
<td>1</td>
<td>8</td>
<td>CD 1</td>
</tr>
<tr>
<td>2</td>
<td>17</td>
<td>CD 0</td>
</tr>
<tr>
<td>3</td>
<td>12</td>
<td>CD 2</td>
</tr>
</tbody>
</table>

**Max conflict Iteration:** 2

**Conflict Detection:** CD 0, CD 1, CD 2, CD 3
Data Packing

Threads

Memory
Data Packing

Threads

Memory
Data Packing

Threads

Memory
Quantization

- Preprocessing finds min and max of the input sets and packs the data.

- During execution, each thread unpacks the data and transforms the quantization level to data by applying a linear transformation.
Quantization

• Preprocessing finds max and min of the input sets and packs the data

• During execution, each thread unpacks the data and transforms the quantization level to data by applying a linear transformation
## Approximation Methods

<table>
<thead>
<tr>
<th>Evaluation Metric</th>
<th>Atomic Operation</th>
<th>Approximate Kernels</th>
</tr>
</thead>
<tbody>
<tr>
<td>Target Output Quality (TOQ)</td>
<td>Data Packing</td>
<td>Tuning Parameters</td>
</tr>
<tr>
<td>CUDA Code</td>
<td>Thread Fusion</td>
<td></td>
</tr>
</tbody>
</table>
Thread Fusion

Memory

Threads

Memory
Thread Fusion

Memory

Threads

Memory
Thread Fusion

Computation

Output writing

T0 T1
Thread Fusion

Block 0

Block 1

T0 T1

T254 T255

Computation

Output writing
Reducing the number of threads per block results in poor resource utilization.
Block Fusion

Block 0 & 1 fused
Runtime
How to Compute Output Quality?

- High overhead
- Tuning should find a good enough approximate version as fast as possible
- Greedy algorithm
Tuning

TOQ = 90%

Tuning parameter of the First optimization

K(x,y)

K(0,0)

Quality = 100%

Speedup = 1x

Tuning parameter of the Second optimization
Tuning

TOQ = 90%

K(0,0)
Quality = 100%
Speedup = 1x

K(1,0)
94%
1.15X

K(0,1)
96%
1.5X
TOQ = 90%

K(0,0)  Quality = 100%
       Speedup = 1x

K(1,0)  94%
        1.15X

K(0,1)  96%
        1.5X

K(1,1)  94%
        2.5X

K(0,2)  94%
        1.6X

TOQ = 90%
**Tuning**

**TOQ = 90%**

- K(0,0)  
  - Quality = 100%  
  - Speedup = 1x

- K(1,0)  
  - 94%  
  - 1.15x

- K(0,1)  
  - 96%  
  - 1.5x

- K(1,1)  
  - 94%  
  - 2.5x

- K(2,1)  
  - 88%  
  - 3x

- K(0,2)  
  - 95%  
  - 1.6x

- K(1,2)  
  - 87%  
  - 2.8x

Final Choice: K(0,1)

Tuning Path: K(0,0) → K(1,1) → K(0,1)
Evaluation
Experimental Setup

- Backend of Cetus compiler

- GPU
  - NVIDIA GTX 560
    - 2GB GDDR 5

- CPU
  - Intel Core i7

- Benchmarks
  - Image processing
  - Machine Learning
After checking 50 samples, we will be 93% confident that 95% of the outputs satisfy the TOQ threshold.
Calibration Overhead

Calibration Overhead(%) vs Calibration Interval

- Gaussian
- K-Means
Performance

TOQ = 95%

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Loop Perforation</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>GAUSSIAN</td>
<td>1.7</td>
<td>SAGE</td>
</tr>
<tr>
<td>MEAN FILTER</td>
<td>1.7</td>
<td></td>
</tr>
<tr>
<td>DYNAMIC</td>
<td>3.3</td>
<td></td>
</tr>
<tr>
<td>BINARIZATION</td>
<td>1.3</td>
<td></td>
</tr>
<tr>
<td>MEANSHIFT</td>
<td>1.6</td>
<td>2.2</td>
</tr>
<tr>
<td>FUZZY</td>
<td>1.5</td>
<td>2.3</td>
</tr>
<tr>
<td>SVM</td>
<td>1.7</td>
<td></td>
</tr>
<tr>
<td>HISTOGRAM</td>
<td>1.4</td>
<td></td>
</tr>
<tr>
<td>NB</td>
<td>1.8</td>
<td>3.6</td>
</tr>
<tr>
<td>K-MEANS</td>
<td>1.3</td>
<td>1.8</td>
</tr>
</tbody>
</table>

TOQ = 90%

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Loop Perforation</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>GAUSSIAN</td>
<td>1.3</td>
<td>SAGE</td>
</tr>
<tr>
<td>MEAN FILTER</td>
<td>1.3</td>
<td></td>
</tr>
<tr>
<td>DYNAMIC</td>
<td>3.3</td>
<td></td>
</tr>
<tr>
<td>BINARIZATION</td>
<td>1.3</td>
<td></td>
</tr>
<tr>
<td>MEANSHIFT</td>
<td>1.4</td>
<td>2.2</td>
</tr>
<tr>
<td>FUZZY</td>
<td>1.5</td>
<td>2.7</td>
</tr>
<tr>
<td>SVM</td>
<td>1.3</td>
<td>1.8</td>
</tr>
<tr>
<td>HISTOGRAM</td>
<td>1.3</td>
<td>1.4</td>
</tr>
<tr>
<td>NB</td>
<td>1.3</td>
<td>3.5</td>
</tr>
<tr>
<td>K-MEANS</td>
<td>1.4</td>
<td>2.2</td>
</tr>
</tbody>
</table>

Geometric Mean

<table>
<thead>
<tr>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
</tbody>
</table>

6.4

TOQ = 90%

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Loop Perforation</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>GAUSSIAN</td>
<td>1.3</td>
<td>SAGE</td>
</tr>
<tr>
<td>MEAN FILTER</td>
<td>1.3</td>
<td></td>
</tr>
<tr>
<td>DYNAMIC</td>
<td>3.3</td>
<td></td>
</tr>
<tr>
<td>BINARIZATION</td>
<td>1.3</td>
<td></td>
</tr>
<tr>
<td>MEANSHIFT</td>
<td>1.4</td>
<td>2.2</td>
</tr>
<tr>
<td>FUZZY</td>
<td>1.5</td>
<td>2.5</td>
</tr>
<tr>
<td>SVM</td>
<td>1.3</td>
<td></td>
</tr>
<tr>
<td>HISTOGRAM</td>
<td>1.4</td>
<td></td>
</tr>
<tr>
<td>NB</td>
<td>1.6</td>
<td></td>
</tr>
<tr>
<td>K-MEANS</td>
<td>1.4</td>
<td></td>
</tr>
</tbody>
</table>

Geometric Mean

<table>
<thead>
<tr>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
</tbody>
</table>

4.8
Conclusion

• Automatic approximation is possible

• SAGE automatically generates approximate kernels with different parameters

• Runtime system uses tuning parameters to control the output quality during execution

• 2.5x speedup with less than 10% quality loss compared to the accurate execution
SAGE: Self-Tuning Approximation for Graphics Engines

Mehrzad Samadi¹, Janghaeng Lee¹, D. Anoushe Jamshidi¹, Amir Hormati², and Scott Mahlke¹

University of Michigan¹, Google Inc.²

December 2013
Backup Slides
What Does Calibration Miss?
SAGE Can Control The Output Quality

![Graph showing the speedup of Naïve Bayes, Fuzzy Kmeans, and Mean Filter against output quality.](image-url)
Distribution of Errors

- Histogram
- Kmeans
- Naïve Bayes
- Fuzzy Kmeans
- SVM
- Dynamic
- Mean Filter
- Gaussian
- Binarization
- Meanshift