Accelerating Image Feature Comparisons using CUDA on Commodity Hardware

Seth Warn, Wesley Emeneke, John Gauch, Jackson Cothren, Amy Apon

University of Arkansas
Outline

• Background
• GPU kernel implementation
• Using the kernel
Image Analysis
Matching
Challenge

- Very large images (gigapixels)
- Lots of features (tens of millions)
SIFT Features
SIFT Features
SIFT Features
SIFT Features
SIFT Features
Comparing Features

The usual method
Euclidean (L2) Distance

There is a better way...
CEMD

Circular Earth Mover’s Distance

A

\[
\begin{array}{cccc}
1 & 5 & 6 & 7 \\
\end{array}
\]

B

\[
\begin{array}{cccc}
3 & 5 & 6 & 5 \\
\end{array}
\]

\[
\text{EMD} \quad 2 \times 3 = 6
\]

\[
\text{CEMD} \quad 2 \times 1 = 2
\]
CEMD vs. L2

CEMD gives us better results, but...

Comparing 2048 × 2048 Descriptors (4,194,304 comparisons)

<table>
<thead>
<tr>
<th></th>
<th>seconds</th>
<th>comp/sec</th>
</tr>
</thead>
<tbody>
<tr>
<td>CEMD</td>
<td>12.8</td>
<td>328 k</td>
</tr>
<tr>
<td>L2</td>
<td>2.78</td>
<td>1.51 M</td>
</tr>
</tbody>
</table>

CEMD is 4.6 times slower than L2
Benchmarking Platform

- NVIDIA GTX 295
  - using a single GPU
  - times reported with and without transfer
- Intel Xeon E5520
  - using a single core
  - data fits in cache and starts hot
GPU Kernels

- GPU-INITIAL
- GPU-SHMEM
- GPU-MEMOPT
- GPU-INDEX
- GPU-UNROLL
- GPU-FINAL
GPU-INITIAL

- Unoptimized kernel code
- $16 \times 16$ thread blocks
- Read $16 \times 16$ descriptors from memory
- Write $16 \times 16$ distances to memory
Comparing 2048 × 2048 Descriptors (4,194,304 comparisons)

<table>
<thead>
<tr>
<th></th>
<th>seconds</th>
<th>comp/sec</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CPU</strong></td>
<td>12.8</td>
<td>328 k</td>
</tr>
<tr>
<td><strong>GPU (exec)</strong></td>
<td>1.61</td>
<td>2.61 M</td>
</tr>
<tr>
<td><strong>GPU (total)</strong></td>
<td>1.69</td>
<td>2.48 M</td>
</tr>
</tbody>
</table>

Speedup: 7.95x (exec) or 7.57x (total)
GPU-SHMEM

Kernel uses shared memory

• Read 16 + 16 descriptors from global memory into shared memory

• Perform histogram generation on descriptors in shared memory
GPU-SHMEM

Comparing 2048 × 2048 Descriptors (4,194,304 comparisons)

<table>
<thead>
<tr>
<th></th>
<th>seconds</th>
<th>comp/sec</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>12.8</td>
<td>328 k</td>
</tr>
<tr>
<td>GPU (exec)</td>
<td>0.444</td>
<td>9.45 M</td>
</tr>
<tr>
<td>GPU (total)</td>
<td>0.527</td>
<td>7.95 M</td>
</tr>
</tbody>
</table>

Speedup: 28.8x (exec) or 24.3x (total)
GPU-MEMOPT

Optimize global memory reads

- GPU-SHMEM global reads are single-byte loads with a stride of 128 bytes
- GPU-MEMOPT reads consecutive 4-byte words from global memory
# GPU-MEMOPT

Comparing 2048 × 2048 Descriptors (4,194,304 comparisons)

<table>
<thead>
<tr>
<th></th>
<th>seconds</th>
<th>comp/sec</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>12.8</td>
<td>328 K</td>
</tr>
<tr>
<td>GPU (exec)</td>
<td>0.429</td>
<td>9.77 M</td>
</tr>
<tr>
<td>GPU (total)</td>
<td>0.510</td>
<td>8.22 M</td>
</tr>
</tbody>
</table>

**Speedup:** 29.8x (kernel) or 25.0x (overall)
Memory Bandwidth

- Each block does 256 comparisons
- Each block reads 32 descriptors, for a total of 4096 bytes
- $\frac{4096}{256} = 16$ bytes/comparison
- At 9.8 M comp/sec, that’s 156.8 MB/s

$156.8 \text{ MB/s} \ll 112 \text{ GB/s}$
Shared Memory
Shared Memory

Bank 0
Shared Memory

Bank 1
Shared Memory

Bank 2
Shared Memory

Bank 15
Shared Memory

Descriptors
Shared Memory

Thread 0
Thread 1
Thread 2
Thread 3
Shared Memory
# GPU-INDEX

Comparing 2048 × 2048 Descriptors (4,194,304 comparisons)

<table>
<thead>
<tr>
<th></th>
<th>seconds</th>
<th>comp/sec</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CPU</strong></td>
<td>12.8</td>
<td>328 K</td>
</tr>
<tr>
<td><strong>GPU (exec)</strong></td>
<td>0.535</td>
<td>7.85 M</td>
</tr>
<tr>
<td><strong>GPU (total)</strong></td>
<td>0.619</td>
<td>6.78 M</td>
</tr>
</tbody>
</table>

20% slower than GPU-MEMOPT!

Avoiding Bank Conflicts

Before “Optimization”

method=[  _Z17cudaMultiDescMemdPA16_A8_KhmS2_mPj ]
gputime=[  428524.562 ] occupancy=[  0.500 ]
gld_request=[  17504 ] warp_serialze=[  156639670 ]

After “Optimization”

method=[  _Z17cudaMultiDescMemdPA16_A8_KhmS2_mPj ]
gputime=[  534386.750 ] occupancy=[  0.250 ]
gld_request=[  17504 ] warp_serialze=[  31420482 ]
__device__ uint32_t
cudaSingleDistance(const histo_t & a, const histo_t & b, size_t start)
{
    int32_t a_acc = a[start];
    int32_t b_acc = b[start];
    uint32_t distance = abs(a_acc - b_acc);

    for (size_t i = start+1; i < HISTO_DIM; i++) {
        a_acc += a[i];
        b_acc += b[i];
        distance += abs(a_acc - b_acc);
    }

    for (size_t i = 0; i < start; i++) {
        a_acc += a[i];
        b_acc += b[i];
        distance += abs(a_acc - b_acc);
    }

    return distance;
}
template <size_t start>
__device__ uint32_t
cudaSingleDistance(const histo_t & a, const histo_t &b) {
    int32_t a_acc = a[start];
    int32_t b_acc = b[start];
    uint32_t distance = abs(a_acc - b_acc);

    if (start < 1) { a_acc += a[1]; b_acc += b[1];
        distance += abs(a_acc - b_acc); }
    "* * * *
    if (start < 7) { a_acc += a[7]; b_acc += b[7];
        distance += abs(a_acc - b_acc); }
    "* * * *
    if (start > 0) { a_acc += a[0]; b_acc += b[0];
        distance += abs(a_acc - b_acc); }
    "* * * *
    if (start > 6) { a_acc += a[6]; b_acc += b[6];
        distance += abs(a_acc - b_acc); }

    return distance;
}
template <size_t start>
__device__ uint32_t
cudaSingleDistance(const histo_t & a, const histo_t &b)
{
    int32_t a_acc = a[start];
    int32_t b_acc = b[start];
    uint32_t distance = abs(a_acc - b_acc);

#define unrollSingle(z, loop, gt_lt) \
    if (start gt_lt loop) { \
        a_acc += a[loop]; \
        b_acc += b[loop]; \
        distance += abs(a_acc - b_acc); } \

    BOOST_PP_REPEAT_FROM_TO(1, HISTO_DIM, unrollSingle, <) \
    BOOST_PP_REPEAT_FROM_TO(0, BOOST_PP_DEC(HISTO_DIM), unrollSingle, >) \

    return distance;
}
Loop Unrolling

```c
__device__ uint32_t cudaMemDistance(const histo_t & a, const histo_t & b) {
    uint32_t min_distance = cudaSingleDistance(a, b, 0);
    
    for (size_t i = 1; i < HISTO_DIM; i++) {
        uint32_t i_distance = cudaSingleDistance(a, b, i);
        min_distance = min(i_distance, min_distance);
    }

    return min_distance;
}
```
Loop Unrolling

```c
__device__ uint32_t
cudaMemDistance(const histo_t & a, const histo_t & b) {
    uint32_t min_distance = cudaSingleDistance<0>(a, b);
    uint32_t i_distance;

    i_distance = cudaSingleDistance<1>(a, b);
    min_distance = i_distance < min_distance ? i_distance : min_distance;
    i_distance = cudaSingleDistance<2>(a, b);
    min_distance = i_distance < min_distance ? i_distance : min_distance;
    i_distance = cudaSingleDistance<3>(a, b);
    min_distance = i_distance < min_distance ? i_distance : min_distance;
    i_distance = cudaSingleDistance<4>(a, b);
    min_distance = i_distance < min_distance ? i_distance : min_distance;
    i_distance = cudaSingleDistance<5>(a, b);
    min_distance = i_distance < min_distance ? i_distance : min_distance;
    i_distance = cudaSingleDistance<6>(a, b);
    min_distance = i_distance < min_distance ? i_distance : min_distance;
    i_distance = cudaSingleDistance<7>(a, b);
    min_distance = i_distance < min_distance ? i_distance : min_distance;

    return min_distance;
}
```
__device__ uint32_t
cudaMemDistance(const histo_t & a, const histo_t & b)
{
    uint32_t min_distance = cudaSingleDistance<0>(a, b);

#define unrollMemd(z, loop, _)
min_distance = min(cudaSingleDistance<loop>(a, b), min_distance);

    BOOST_PP_REPEAT_FROM_TO(1, HISTO_DIM, unrollMemd, _)

    return min_distance;
}

GPU-UNROLL

Comparing 2048 $\times$ 2048 Descriptors (4,194,304 comparisons)

<table>
<thead>
<tr>
<th></th>
<th>seconds</th>
<th>comp/sec</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>9.19</td>
<td>457 k</td>
</tr>
<tr>
<td>GPU (exec)</td>
<td>0.0843</td>
<td>49.8 M</td>
</tr>
<tr>
<td>GPU (total)</td>
<td>0.166</td>
<td>25.2 M</td>
</tr>
</tbody>
</table>

Speedup: 109x (exec) or 55.4x (total)
GPU-UNROLL

Comparing 2048 × 2048 Descriptors (4,194,304 comparisons)

method=\[_Z17cudaMultiDescMemdPA16_A8_KhmS2_mPj\]
gputime=[ 84151.711 ] occupancy=[ 0.500 ]
gld_request=[ 17504 ] warpSerialize=[ 20171239 ]

... and it reduced our shared memory conflicts!
Instruction Throughput

Total: 319 instructions / histogram
Instruction Throughput

16 histograms / descriptor

15 add

Ignoring loop overhead

Total: 5119 instructions / descriptor
Instruction Throughput

The GT200 chip in the GTX295 has 240 processors running at 1.24 GHz: 297.6 billion instructions per second

The 4,194,304 comparisons in the test require 21,470,642,176 instructions

Minimum time for those comparisons: 0.072 seconds
Instruction Throughput

Minimum Time:
not including global reads, global writeback

0.072 seconds

Measured Time:
including global reads, global writeback

0.084 seconds
However!

Algorithmic improvements are still possible.
Comparing 2048 × 2048 Descriptors (4,194,304 comparisons)

<table>
<thead>
<tr>
<th></th>
<th>seconds</th>
<th>comp/sec</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>5.82</td>
<td>720 k</td>
</tr>
<tr>
<td>GPU (exec)</td>
<td>0.0771</td>
<td>54.4 M</td>
</tr>
<tr>
<td>GPU (total)</td>
<td>0.165</td>
<td>25.5 M</td>
</tr>
</tbody>
</table>

Speedup: 75.5x (exec) or 35.2x (total)
Comparisons/sec

Number of comparisons

GPU-FINAL

GPU
GPU with overhead
CPU
Final Notes

• Cluster Implementation: 6 nodes, 24 GPUs, over 1.2 billion comparisons per second
• Overlapping kernel launches recover some of the overhead of memory transfer
• One “Fermi” GTX 480: 114 M comp/sec
Conclusions

Comparisons / second

GPU

CPU

INIT  SHMEM  MEMOPT  UNROLL  FINAL