Scalable SIFT for NUMA with Actors

Frank Feinbube, Lena Herscheid, Christoph Neijenhuis, Peter Tröger

Hasso Plattner Institute for IT Systems Engineering
What is Scale Invariant Feature Transform (SIFT) good for?
This is what SIFT was used for:

Large images. Many small descriptors. Batch processing.
This is what SIFT was used for:

Slow stream of small images.
Some large descriptors.
Knowledge about objects.
This is what we wanted to use SIFT for:

Fast stream of large images.
Lots of descriptors of various sizes!

Scalable SIFT for NUMA with Actors
Frank Feinbube, Research Assistant
How does Scale Invariant Feature Transform (SIFT) work?

“Distinctive Image Features from Scale-Invariant Keypoints”

International Journal of Computer Vision, 2004

David Lowe

Computer Science Department
2366 Main Mall
University of British Columbia
Vancouver, B.C., V6T 1Z4, Canada

E-mail: lowe@cs.ubc.ca
Scalable SIFT for NUMA with Actors
Frank Feinbube, Research Assistant
1. Create octaves of differently scaled copies
1. Create octaves of differently scaled copies
1. Create octaves of differently scaled copies

2. Apply different Gaussian blurs

SIFT algorithm
2. Apply different Gaussian blur
1. Create octaves of differently scaled copies

2. Apply different Gaussian blurs

3. Compute DoG within each octave
3. Compute DoG (difference of gaussians)

Two different Gaussian blur filters are applied to the same image.

The difference between the two resulting images highlights the main image characteristics.
3. Compute DoG within each octave

Two different Gaussian blur filters are applied to the same image.

The difference between the two resulting images highlights the main image characteristics.
1. Create octaves of differently scaled copies

2. Apply different Gaussian blurs

3. Compute DoG within each octave

4. Filter extrema

SIFT algorithm
4. Filter extrema

Extrema Detection:

- Darker than its neighbors
- Darker than its neighbors
- Darker than the ones in the other scales!
4. Filter extrema

Exrema Filtering:

- Due to rasterization extrema might be located at different pixels leading to different descriptors.

Using sub-pixel positions and sub-scale positions for interpolation increases the probability to recognize a detector about 10% to 25%.

[M. Brown and D. G. Lowe, „Invariant features from interest point groups,” in British Machine Vision Conference, 2002.]
1. Create octaves of differently scaled copies

2. Apply different Gaussian blurs

3. Compute DoG within each octave

4. Filter extrema

5. Detect gradients, normalize orientation

SIFT algorithm
5. Detect gradients, normalize orientation

Gradient histogram: the change of brightness in that direction

For the highest value and each value within 80% of it an accordingly oriented descriptor is created!
Visualization of a single descriptor

A descriptor comprises 4x4 gradient histograms describing the relative change of brightness in the area of a feature.
1. Create octaves of differently scaled copies
2. Apply different Gaussian blurs
3. Compute DoG within each octave
4. Filter extrema
5. Detect gradients, normalize orientation

**Output:** Feature descriptors (gradient histograms) + orientation + blur factor + interpolated x,y coordinates
Scalable SIFT for NUMA with Actors
Frank Feinbube, Research Assistant
What did we contribute?
Scala

- Productivity-focused high-level programming language
- Designed to allow for a high degree of parallelization and scalability

- Extensive actor library (Akka)
- Effortless distribution across multiple nodes
- Runs on JVM
Faster than OpenCV (C/C++)

- SIFT in OpenCV is optimized C/C++, but still sequential
- We benchmarked our Scala-based implementation in sequential mode

<table>
<thead>
<tr>
<th></th>
<th>OpenCV (C/C++)</th>
<th>Our Scala implementation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Runtime</td>
<td>Features</td>
</tr>
<tr>
<td>1920x1080</td>
<td>7670 ms</td>
<td>9460</td>
</tr>
<tr>
<td>800x600</td>
<td>1330 ms</td>
<td>1316</td>
</tr>
</tbody>
</table>

1.29 times faster for 1920x1080
1.18 times faster for 800x600
Data structure optimization – 2D array allocation

Scalable SIFT for NUMA with Actors
Frank Feinbube,
Research Assistant
Data structure optimization – 2D array allocation

Frank Feinbube, Research Assistant

Scalable SIFT for NUMA with Actors

![Graph](image)
Different strategies for various combinations of image size and cache size

- If less than 3 images fit into cache: order doesn’t matter
- 3 images: all blurs one after another -> all subtracts -> ...
- 4-6 images: all blurs one after another -> all subtracts one after another (in backwards order)
- >6 images: blur single image -> subtract -> blur next -> ...
- 16 images: order doesn’t matter

Has to be considered for each octave, since images are smaller for higher octaves
Algorithmic optimization – image flipping

<table>
<thead>
<tr>
<th>Image</th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Horizontal Gaussian Blur Filter</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2. Vertical Gaussian Blur Filter</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Our experiment shows that A and C perform similarly when executed serially, but B is 67% slower.

With six threads, B is still 35% slower than A, while C is 16% faster.
Optimization of the order of SIFT stages

- L2 cache: execute right after one another in one processing step
  - Extrema detection and interpolation
  - Computation of the orientation and the descriptor
- Smaller images = more blurred
  - Have a smaller amount of extrema -> less descriptors to compute
    - Probably even less than cores available
  - Collect extrema for all octaves first
Work distribution on NUMA nodes

- JVM
  - No control over NUMA environment (thread affinity)
    - Uncontrollable memory access latencies
  - Runtime and object management centralized in JVM instance

- We start one JVM per NUMA node

- Performance improvements:
  - 54% when using 2 JVMs instead of 1 on two NUMA nodes
  - 79% when using 4 JVMs instead of 1 on four NUMA nodes
Work distribution on NUMA nodes

- Actor model: distribution on multiple CPUs or multiple systems
  - One actor per JVM
    - Cache-aware and parallelized
  - Master actor decodes video stream and distributes frames
  - Work actors perform SIFT stages

- Video decoding is fast, disk access speed is the bottleneck for master
Work Distribution Strategy (3 types of actors)
## Related work and our contribution

<table>
<thead>
<tr>
<th>Image Size</th>
<th>Features</th>
<th>Processors</th>
<th>Cores / Processor</th>
<th>Cores</th>
<th>Speedup</th>
<th>Speedup Factor</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Feng et al. [3]</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HDTV</td>
<td>1038</td>
<td>4</td>
<td>4</td>
<td>16</td>
<td>9.7</td>
<td>0.61</td>
</tr>
<tr>
<td>720x576</td>
<td>700</td>
<td>4</td>
<td>4</td>
<td>16</td>
<td>11</td>
<td>0.69</td>
</tr>
<tr>
<td>HDTV</td>
<td>1038</td>
<td>1</td>
<td>64</td>
<td>64</td>
<td>52</td>
<td>0.81</td>
</tr>
<tr>
<td>720x576</td>
<td>700</td>
<td>1</td>
<td>64</td>
<td>64</td>
<td>39</td>
<td>0.61</td>
</tr>
<tr>
<td><strong>Zhang et al. [13]</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>640x480</td>
<td>200-1000</td>
<td>2</td>
<td>4</td>
<td>8</td>
<td>5.9-6.7</td>
<td>0.73-0.84</td>
</tr>
<tr>
<td><strong>Warn et al. [11]</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4136x1424</td>
<td>40000</td>
<td>2</td>
<td>4</td>
<td>8</td>
<td>2-3</td>
<td>0.25-0.38</td>
</tr>
<tr>
<td><strong>Our Approach (150 video frames)</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>854x480</td>
<td>1400-2700</td>
<td>1</td>
<td>6</td>
<td>6</td>
<td>7.15</td>
<td><strong>1.19</strong></td>
</tr>
<tr>
<td>854x480</td>
<td>1400-2700</td>
<td>2</td>
<td>6</td>
<td>12</td>
<td>13.21</td>
<td><strong>1.10</strong></td>
</tr>
<tr>
<td>854x480</td>
<td>1400-2700</td>
<td>4</td>
<td>6</td>
<td>24</td>
<td>24.23</td>
<td><strong>1.01</strong></td>
</tr>
</tbody>
</table>
Related Work

- Feng et al. [3]: OpenMP, performance optimizations, 4x4 HP DL580 G5
  - SIMD optimizations can halve the runtime
  - Thread affinity, false sharing removal and synchronization reduction yields a 25% performance improvement
  - Speedup factors of 9.7 for large pictures and 11 for small pictures
  - Scalability investigated with CMP simulator, 64 cores, shared L2
  - Speedup of 52 for large pictures and 39 for small pictures

- Zhang et al. [13]: OpenMP, 2x4 HP DL380 G5
  - Speedup of 5.9-6.7 depending on the feature density in the images
  - For 640x480 images speedup factor is slightly higher than that of Feng et al.'s implementation.
Warn et al. [11]: OpenMP, parallelization of the most expensive loops
- Works best with large satellite pictures
- Speedup of factor 2 on the 8-core test system.

Several SIFT implementations for GPU accelerators [5, 9, 11, 12]
- Bottleneck: data copy / move overhead
- Warn et al. [11]: execute only Gaussian blurring on the GPU
  - Copying overhead = 90% of the execution time
  - Still, GPU version is 13 times faster than the CPU version

Absolute performance over various hardware architectures is not well comparable
## Related work and our contribution

<table>
<thead>
<tr>
<th></th>
<th>Image Size</th>
<th>Features</th>
<th>Processors</th>
<th>Cores / Processor</th>
<th>Cores</th>
<th>Speedup</th>
<th>Speedup Factor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Feng et al. [3]</td>
<td>HDTV</td>
<td>1038</td>
<td>4</td>
<td>4</td>
<td>16</td>
<td>9.7</td>
<td>0.61</td>
</tr>
<tr>
<td></td>
<td>720x576</td>
<td>700</td>
<td>4</td>
<td>4</td>
<td>16</td>
<td>11</td>
<td>0.69</td>
</tr>
<tr>
<td></td>
<td>HDTV</td>
<td>1038</td>
<td>1</td>
<td>64</td>
<td>64</td>
<td>52</td>
<td>0.81</td>
</tr>
<tr>
<td></td>
<td>720x576</td>
<td>700</td>
<td>1</td>
<td>64</td>
<td>64</td>
<td>39</td>
<td>0.61</td>
</tr>
<tr>
<td>Zhang et al. [13]</td>
<td>640x480</td>
<td>200-1000</td>
<td>2</td>
<td>4</td>
<td>8</td>
<td>5.9-6.7</td>
<td>0.73-0.84</td>
</tr>
<tr>
<td>Warn et al. [11]</td>
<td>4136x1424</td>
<td>40000</td>
<td>2</td>
<td>4</td>
<td>8</td>
<td>2-3</td>
<td>0.25-0.38</td>
</tr>
<tr>
<td><strong>Our Approach</strong> (150 video frames)</td>
<td>854x480</td>
<td>1400-2700</td>
<td>1</td>
<td>6</td>
<td>6</td>
<td>7.15</td>
<td><strong>1.19</strong></td>
</tr>
<tr>
<td></td>
<td>854x480</td>
<td>1400-2700</td>
<td>2</td>
<td>6</td>
<td>12</td>
<td>13.21</td>
<td><strong>1.10</strong></td>
</tr>
<tr>
<td></td>
<td>854x480</td>
<td>1400-2700</td>
<td>4</td>
<td>6</td>
<td>24</td>
<td>24.23</td>
<td><strong>1.01</strong></td>
</tr>
</tbody>
</table>
Distribution over multiple systems / into the cloud

- No modifications needed due to application of actor programming model
  - Effortless reuse of such an implementation in various execution environments
- Communication / synchronization overhead
- Increased image throughput
- Widely known I/O optimizations for cluster computing to reduce latency
  - Fast interconnects, parallel file system, ...

- When distributed across 5 machines, we achieved a **speedup of 3.74**
Work per actor depends on feature density

- We make sure that each work actor always has 2 images (enough to hide memory latency effects)
Iterative vs. original-based blurring

- Iterative blurring (red) has a smaller filter size (less computations) and is thus favored by many implementations (almost similar accuracy)
- Leads to huge increase in pixel transfer overhead (51% for 16 tiles)
- Use original-based blurring in distributed scenarios
Example: shared belt of 45 pixels (ghost cells)

- Left: one of 4 image tiles, 13% overhead
- Right: one of 64 image tiles, 79% overhead
Where do we go from here?
Future Work: GPU / Accelerator Implementation

Dynamic Parallelism?

Scalable SIFT for NUMA with Actors
Frank Feinbube, Research Assistant
Future Work: Distributed, heterogeneous Actors

Scalable SIFT for NUMA with Actors

Frank Feinbube, Research Assistant
Data structure optimization: 2D array allocation

Optimization of the order of SIFT stages and caching

Algorithmic optimization: image flipping

NUMA node work distribution strategies

Multi-System work distribution

...
Scalable SIFT for NUMA with Actors

Frank Feinbube, Lena Herscheid, Christoph Neijenhuis, Peter Tröger

Hasso Plattner Institute for IT Systems Engineering