A Comprehensive Analysis and Parallelization of an Image Retrieval Algorithm

Zhenman Fang, Donglei Yang, Weihua Zhang, Haibo Chen, Binyu Zang

Parallel Processing Institute, Fudan University
Exploding Multimedia Data

Cisco VNI Global Consumer Internet Traffic Forecast

Figure from [Report on American Consumers 09]
Multimedia Retrieval App.

Important to retrieve useful data
  - E.g. medical imagery, video recommendation

Data-intensive and computing-intensive

Significant challenges for real-time retrieval
Multi-core Era Is Coming

Figure from Michael MoCool’s (Intel) many core slides
New Opportunities

Multi-core era has opened new opportunities

Need a comprehensive study on parallelism characteristics in multimedia retrieval

- To optimize them on current architectures
- To design future architectures for them
Contributions

A comprehensive analysis of different parallelism
  - Pipeline/task/data parallelism and their combinations

Evaluation of the effectiveness of each parallelism

Parallel implementation on both CPU and GPU
  - Notable performance speedup

Some insights to parallelizing other algorithms
Image Retrieval

Image retrieval: also core of video retrieval
Image Retrieval

feature extraction
Image Retrieval

feature extraction + feature match
Image Retrieval (cont.)

Two classes of algorithms

- Global feature based
  - Color features
  - Texture features

- Local feature based
  - Shape context
  - SIFT Features
  - SURF Features

~60% precision [Wan 08]
accurate but time consuming

most robust & appealing
[Mikolajczyk 05, Bauer 07]
SURF Overview

Input Image

Detection
- Integral Image
- Scale Space Analysis
- Interest Point Localization

Description
- Orientation Assignment
- Descriptor Vector Construction

Features
Integral Image

\[ \sum g(x,y) \]

Input Image

\[ g(x,y) \]

Integral Image

\[ I(x,y) \]
Scale Space Analysis

Hessian Matrix for \((x,y)\) [Bay 06]

\[
\begin{pmatrix}
D_{xx} & D_{xy} \\
D_{xy} & D_{yy}
\end{pmatrix}
\]

\[
\text{Det}(x,y) = D_{xx}D_{yy} - 0.81D_{xy}D_{xy}
\]
Interest Point Localization

Ipoint with max det value
Orientation Assignment

Based on Haar Wavelet [Bay 06]
Descriptor Vector Construction
Descriptor Vector Construction

64-dimension feature vector

$$(\sum dx, \sum dy, \sum |dx|, \sum |dy|)$$

dx, dy based on **Haar Wavelet**
Execution Profile of SURF

Input Image

Detection
- Integral Image 1% time
- Scale Space Analysis 24% time
- Interest Point Localization 2% time

27% time

Description
- Orientation Assignment 20% time
- Descriptor Vector Construction 53% time

73% time

Features

Experiment Environment
- Prog: OpenSURF
- Input: 48 images
- HW: 16-core server 32GB memory
Parallel Analysis

Pipeline Parallelism

Task Parallelism
  - Scale-level Parallelism
  - Block-level Parallelism

Combination of Different Parallelism
  - Combination of SIMD and Other Parallelism
  - Combination of Pipeline and Task Parallelism
2-stage Pipeline

Detection writes interest point to the buffer

Description reads interest point from the buffer
3-stage Pipeline

Further divide Description into two stages

Detection → Orientation Assignment → Descriptor Vector Construction

Detection → Orientation Assignment → Descriptor Vector Construction
Results of Pipeline Parallelism

Pipeline parallelism does not scale

![Speedup Chart]

- 2-stage
- 3-stage
Why Poor Scalability?

Sync between Detection & Description

Detection

Description

Normal speed
Why Poor Scalability?

Sync between Detection & Description

Detection

Description

Waiting!

Too fast
Why Poor Scalability?

Sync between Detection & Description

Detection

Description

Too slow

Waiting!
Why Poor Scalability?

Sync between Detection & Description

2-stage pipeline: 3 Description components to ease the synchronization

Waiting!
Parallel Analysis

Pipeline Parallelism

Task Parallelism
- Scale-level Parallelism
- Block-level Parallelism

Combination of Different Parallelism
- Combination of SIMD to Others
- Combination of Task and Pipeline Parallelism
Scale-level Parallelism

Each scale computed concurrently

Describe each group of interest points concurrently

Integral Image
Scale Space Analysis
Interest Point Localization
Description
Results of Scale-level Parallelism

Not scale when exceeding 12 cores

![Bar chart showing speedup for 4-core, 8-core, 12-core, and 16-core systems]
Results of Scale-level Parallelism

Not scale when exceeding 12 cores

- Imbalanced computation
- Non-trivial communication overhead

![Graph showing speedup for 4-core, 8-core, 12-core, and 16-core systems. The graph indicates that the speedup is highest at 12 cores, with speedup values of 8, 6, 6, and 6 respectively.]
Parallel Analysis

Pipeline Parallelism

Task Parallelism
- Scale-level Parallelism
- Block-level Parallelism

Combination of Different Parallelism
- Combination of SIMD to Others
- Combination of Task and Pipeline Parallelism
Block-level Parallelism

Input Image

Image Block → Detection → Description

Sync between neighbor blocks
Block-level Parallelism

Block-level parallelism with synchronization (Block-Sync)

Input Image

Sync between neighbor blocks
Block-level Parallelism

Input Image

Image Block

Detection

Description

... (repeated)

Use additional computation to avoid sync
Block-level Parallelism

Block-level parallelism without synchronization (BlockPar)

Use additional computation to avoid sync
Results of Block-level Parallelism

BlockPar scales well

![Graph showing speedup for BlockPar and Block-Sync across different core counts](image)

- BlockPar
- Block-Sync

- Speedup
- 4-core
- 8-core
- 12-core
- 16-core
Results of Block-level Parallelism

BlockPar scales well

Communication overhead between cores is non-trivial; and it could be reduced by additional computation.
Comparison for Each Parallelism

Block-level parallelism is more efficient

![Graph showing speedup comparison for different parallelisms across core counts. The graph indicates that block-level parallelism (BlockPar) is more efficient across 4-core, 8-core, 12-core, and 16-core configurations.]
Parallel Analysis

Pipeline Parallelism

Task Parallelism
- Scale-level Parallelism
- Block-level Parallelism

Combination of Different Parallelism
- Combination of SIMD to Others
- Combination of Task and Pipeline Parallelism
Combination of SIMD to Others

Use ICC to generate SIMD instructions

- Pipeline
- ScalePar
- BlockPar

<table>
<thead>
<tr>
<th></th>
<th>4-core</th>
<th>8-core</th>
<th>12-core</th>
<th>16-core</th>
</tr>
</thead>
<tbody>
<tr>
<td>Speedup</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Combination of SIMD to Others

Use ICC to generate SIMD instructions

11% Speedup

- Pipeline+SIMD
- ScalePar+SIMD
- BlockPar+SIMD
Parallel Analysis

Pipeline Parallelism

Task Parallelism
  - Scale-level Parallelism
  - Block-level Parallelism

Combination of Different Parallelism
  - Combination of SIMD to Others
  - Combination of Task and Pipeline Parallelism
Combination of Task & Pipeline

BlockPar + Pipeline is the most efficient

- BlockPar
- Block+Pipe
- BlockPar+SIMD
- Block+Pipe+SIMD

Speedup vs. Core Count

- 4-core
- 8-core
- 12-core
- 16-core

13X speedup for 16-core configuration.
Combination of Task & Pipeline

BlockPar + Pipeline is the most efficient

- Fewer computation
- Better locality

13X

Speedup

4-core  8-core  12-core  16-core
Comparison to Prior Work

Compared to P-SURF [Zhang 10] on multi-core CPU

Speedup

P-SURF

Our BlockPar

Our Block+Pipe

4-core 8-core 12-core 16-core
Comparison to Prior Work

Compared to P-SURF [Zhang 10] on multi-core CPU

- 1.84X Speedup over P-SURF
- Non-trivial communication overhead
Comparison to Prior Work (cont.)

Our implementation on **GPGPU**

- Sequential SURF on CPU
- Initialization on CPU
- BlockPar on GPGPU
- Sequential CPU + GPU

* Can be downloaded from http://www.mis.tu-darmstadt.de/surf
Comparison to Prior Work (cont.)

Our implementation on **GPGPU**

<table>
<thead>
<tr>
<th></th>
<th>Init</th>
<th>SURF</th>
<th>Sequential CPU + GPU</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Execution Time</strong></td>
<td>53%</td>
<td>47%</td>
<td></td>
</tr>
</tbody>
</table>

- After BlockPar on GPGPU
- Initialization on CPU
- BlockPar on GPGPU

* Can be downloaded form [http://www.mis.tu-darmstadt.de/surf](http://www.mis.tu-darmstadt.de/surf)
Comparison to Prior Work (cont.)

Our implementation on **GPGPU**

<table>
<thead>
<tr>
<th>Execution Time</th>
<th>Init</th>
<th>SURF</th>
<th>CPU + GPU Pipeline</th>
</tr>
</thead>
<tbody>
<tr>
<td>After BlockPar on GPGPU</td>
<td>53%</td>
<td>47%</td>
<td></td>
</tr>
</tbody>
</table>

* Can be downloaded from [http://www.mis.tu-darmstadt.de/surf](http://www.mis.tu-darmstadt.de/surf)
Comparison to Prior Work (cont.)

Compared to CUDA SURF* on GPGPU (Nvidia GTX 260)

* Can be downloaded from http://www.mis.tu-darmstadt.de/surf
Comparison to Prior Work (cont.)

Compared to \textit{CUDA SURF*} on \textit{GPGPU} (Nvidia GTX 260)

- 1.53X speedup over CUDA SURF
- CPU+GPU Pipeline not exploited

\* Can be downloaded from \url{http://www.mis.tu-darmstadt.de/surf}
Summary

First parallelism analysis of image retrieval

- Pipeline Parallelism
- Task Parallelism at scale-level & block-level
- Data Parallelism, i.e., SIMD
- Their combinations

BlockPar + Pipeline is the most efficient

- 13X speedup on 16-core CPU, 1.84X faster than P-SURF
- 46X speedup on GPU, 1.53X faster than CUDA SURF
Conclusion and Future Work

Additional computation to avoid synchronization

Cooperation between CPU & GPU

Future work

- Apply parallel analysis to speech recognition
- Design some energy-efficient architecture, such as FPGA, to accelerate multimedia retrieval
Thanks

Parallel Processing Institute
http://ppi.fudan.edu.cn
Average Execution Time
# of Interest Points per Image

<table>
<thead>
<tr>
<th>Images</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>AVG</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bark</td>
<td>903</td>
<td>866</td>
<td>812</td>
<td>991</td>
<td>913</td>
<td>945</td>
<td>905</td>
</tr>
<tr>
<td>Bikes</td>
<td>698</td>
<td>602</td>
<td>433</td>
<td>482</td>
<td>348</td>
<td>1292</td>
<td>643</td>
</tr>
<tr>
<td>Boat</td>
<td>1292</td>
<td>1246</td>
<td>1153</td>
<td>648</td>
<td>626</td>
<td>615</td>
<td>930</td>
</tr>
<tr>
<td>Graf</td>
<td>1365</td>
<td>1495</td>
<td>1508</td>
<td>1550</td>
<td>1586</td>
<td>1586</td>
<td>1515</td>
</tr>
<tr>
<td>Leuven</td>
<td>802</td>
<td>544</td>
<td>544</td>
<td>428</td>
<td>324</td>
<td>252</td>
<td>482</td>
</tr>
<tr>
<td>Trees</td>
<td>1431</td>
<td>1421</td>
<td>1509</td>
<td>1557</td>
<td>1550</td>
<td>867</td>
<td>1389</td>
</tr>
<tr>
<td>Ubc</td>
<td>845</td>
<td>836</td>
<td>867</td>
<td>921</td>
<td>921</td>
<td>1015</td>
<td>901</td>
</tr>
<tr>
<td>Wall</td>
<td>1142</td>
<td>1142</td>
<td>1110</td>
<td>1205</td>
<td>1143</td>
<td>1143</td>
<td>1148</td>
</tr>
</tbody>
</table>
Interest Points Distribution

- 22%
- 26%
- 27%
- 25%
- 11%
- 12%
- 15%
- 13%
- 8%
- 8%
- 8%
- 8%
- 9%
- 10%
- 7%
- 6%
- 8%
- 6%
- 8%
- 7%
- 8%
- 8%
- 6%
- 6%
- 6%
- 5%
- 5%
- 6%
- 6%
- 7%
- 8%
- 7%
- 7%
- 6%
- 7%
- 6%
- 6%
- 7%
Results of Pipeline Parallelism

Pipeline at image level doesn’t perform well

![Speedup Graph]

- **NoSIMD**
- **SIMD**

<table>
<thead>
<tr>
<th></th>
<th>point level</th>
<th>image level</th>
</tr>
</thead>
<tbody>
<tr>
<td>NoSIMD</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SIMD</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Scale Space Analysis

Octave 0
Octave 1
Octave m

Scale

60