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

Figure from [Report on American Consumers 09]

Cisco VNI Global Consumer Internet Traffic Forecast
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 needs parallelism

Need a comprehensive study on parallelism characteristics in multimedia retrieval

- To optimize them on current architectures
- To design future architectures for them
Image Retrieval

Image retrieval: also core of video retrieval
Image Retrieval

feature extraction
Image Retrieval

feature extraction + feature match
Two classes of algorithms

- Global feature based
  - Color features
  - Texture features

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

Robust & appealing: insensitive to scale and rotation transformation

[Mikolajczyk 05, Bauer 07]

~60% precision

[Wan 08]

Accurate but time consuming
SURF Overview

Input Image

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

Description
- Orientation Assignment
- Descriptor Vector Construction

Features
Integral Image

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

\[ g(x, y) \rightarrow I(x, y) \]

Input Image

Integral Image
**Scale Space Analysis**

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

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

• \(\text{Det}(x,y)\)

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

Insensitive to scale transformation
Interest Point Localization

Point with max det value
Orientation Assignment

Based on Haar Wavelet [Bay 06]

insensitive to rotation transformation
Descriptor Vector Construction
Descriptor Vector Construction

64-dimension feature vector

4-dimension vector calculated based on **Haar Wavelet**
Detection
Integral Image 1% time
Scale Space Analysis 24% time
Interest Point Localization 2% time

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

Features

Experiment Environment
✓ Prog: OpenSURF
✓ Input: 48 images
✓ HW: 16-core server 32GB memory
Interest Points Distribution

Imbalanced distribution for images/blocks

Average line
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

Detection

Description

Description

Description
3-stage Pipeline

Further divide Description into two stages

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

Pipeline parallelism does not scale

![Graph showing speedup for 2-stage and 3-stage pipeline parallelism.]

- Speedup for 2-stage pipeline
- Speedup for 3-stage pipeline
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
Results of Scale-level Parallelism

Not scale when exceeding 12 cores

- Imbalanced computation
- Non-trivial communication overhead
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

Image Block → Detection → Description

Image Block → Detection → Description

Sync between neighbor blocks
Block-level Parallelism

Input Image

Image Block

Image Block

Image Block

... block-level parallelism with synchronization (Block-Sync)

Sync between neighbor blocks
Block-level Parallelism

Input Image

Image Block → Detection → Description
Image Block → Detection → Description
Image Block → Detection → Description

Use additional computation to avoid sync
Block-level Parallelism

Block-level parallelism without synchronization (BlockPar)
Results of Block-level Parallelism

BlockPar scales well

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

**Speedup**

- **BlockPar**
- **Block-Sync**
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 comparison of Pipeline, ScalePar, and BlockPar speedup across 4-core, 8-core, 12-core, and 16-core systems.](image)

- **Pipeline**
- **ScalePar**
- **BlockPar**
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

Speedup

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

Pipeline
ScalePar
BlockPar
Combination of SIMD to Others

Use ICC to generate SIMD instructions

11% Speedup

- Pipeline+SIMD
- ScalePar+SIMD
- BlockPar+SIMD

Speedup

<table>
<thead>
<tr>
<th>Core</th>
<th>Pipeline+SIMD</th>
<th>ScalePar+SIMD</th>
<th>BlockPar+SIMD</th>
</tr>
</thead>
<tbody>
<tr>
<td>4-core</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8-core</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12-core</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16-core</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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

![Graph showing speedup for 4-core, 8-core, 12-core, and 16-core systems. BlockPar, Block+Pipe, BlockPar+SIMD, and Block+Pipe+SIMD are compared. The 16-core system shows a significant speedup of 13x.]
Combination of Task & Pipeline

BlockPar + Pipeline is the most efficient

- Fewer computation
- Better locality

Speedup

<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>BlockPar</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Block+Pipe</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BlockPar+SIMD</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Block+Pipe+SIMD</td>
<td></td>
<td></td>
<td></td>
<td>13x</td>
</tr>
</tbody>
</table>
Comparison to Prior Work

Compared to P-SURF [Zhang 10] on multi-core CPU
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
- BlockPar on GPGPU
- Sequential CPU + GPU

*Can be downloaded from [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>Step</th>
<th>Execution Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initialization</td>
<td>53%</td>
</tr>
<tr>
<td>BlockPar on GPGPU</td>
<td>47%</td>
</tr>
<tr>
<td>Sequential CPU + GPU</td>
<td></td>
</tr>
</tbody>
</table>

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

Our implementation on GPGPU

* Can be downloaded form 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
Compared to **CUDA SURF**\(^*\) on **GPGPU** (Nvidia GTX 260)

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

* Can be downloaded from 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