SD-VBS: The San Diego Vision Benchmark Suite

Sravanthi Kota Venkata, Ikkjin Ahn, Donghwan Jeon, Anshuman Gupta, Chris Louie, Saturnino Garcia, Serge Belongie, Michael B. Taylor

Computer Science and Engineering
University of California, San Diego

http://parallel.ucsd.edu/vision
Vision is an exciting application domain for many-core and multi-core systems

- “Enabling computers to see” will have a tangible and immediate impact on people’s lives

- Limitless thirst for computation
  - Larger image sizes
  - More accurate analyses
  - Match and search against larger databases
  - Run the algorithm in real-time, or even super-real-time

- Full of parallelism to put those idle cores to work
  - i.e. drive user demand for future multi-core processors and keep Moore’s Law going

- Enormous progress in computer vision research over the last decade, and more to come
  - Some problems are now even considered “solved.”
A Few Examples of Vision’s
Current and Potential Impact on Our Lives

• Auto-focus cameras and cell-phones through face detection
• Help doctors perform surgery on patients thousands of miles away
• Allow planes to fly themselves (e.g., UAVs)
• Enable “cars that can’t crash”
• Enable machines that automatically educate our kids (!)
SD-VBS: The San Diego Vision Benchmark Suite

- **Aim:** Make vision more accessible to multi-core researchers through a comprehensive and easy to use suite.
- **9 benchmarks in 4 different areas.**

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Concentration Area</th>
</tr>
</thead>
<tbody>
<tr>
<td>Disparity Map</td>
<td>Motion, Tracking and Stereo Vision</td>
</tr>
<tr>
<td>Feature Tracking</td>
<td>Motion, Tracking and Stereo Vision</td>
</tr>
<tr>
<td>Image Segmentation</td>
<td>Image Analysis</td>
</tr>
<tr>
<td>Scale Invariant Feature Transform (SIFT)</td>
<td>Image Analysis</td>
</tr>
<tr>
<td>Robot Localization</td>
<td>Image Understanding</td>
</tr>
<tr>
<td>Support Vector Machines (SVM)</td>
<td>Image Understanding</td>
</tr>
<tr>
<td>Face Detection</td>
<td>Image Understanding</td>
</tr>
<tr>
<td>Image Stitch</td>
<td>Image Processing and Formation</td>
</tr>
<tr>
<td>Texture Synthesis</td>
<td>Image Processing and Formation</td>
</tr>
</tbody>
</table>
Design goals of SD-VBS

- Clean implementations for ease of analysis and transformation by researchers or their tools
  - “Clean” C and MATLAB versions
  - No dependence on custom libraries
- Comprehensive collection of representative benchmarks
- Low complexity of porting and parallelizing to different platforms
SD-VBS: 9 applications, > 25 kernels

- Disparity Map
  - Feature Tracking
    - Gradient
    - Matrix Invert
    - Area Sum
  - Sort
    - SSD
    - Correlation

- Face Detection
  - Feature Tracking
    - Gradient
    - Matrix Invert
    - Area Sum
  - Adaboost
    - Integral Image
    - Rect Filter

- Texture Synthesis
  - Image Stitch
    - LS Solver
    - PCA
    - Sampling
  - SVD
    - ANMS
    - Gaussian Filter

- Image Segmentation
  - Robot Localization
    - Particle Filter
    - Physical Model
    - Adjacency Matrix
    - Filter Banks
    - EigenSolver
    - QR Factorization

- Support Vector Machines
  - SIFT
    - Interpolation
    - Affine Transforms
    - Conjugate Matrix
    - Matrix Ops
Rich Reuse of Kernels in Vision
Overview of Talk

• Introduction to SD-VBS
• Vision Benchmarks Overview
  – foreach { Feature Tracking, Disparity, Image Stitch, SIFT, Segmentation, SVM, Robot Localization, Texture Synthesis }:
    Brief Demo
    Algorithm Description
    Analysis of Characteristics and Hotspots
• Results
• Related Work
• Conclusion
Feature Tracking: Overview

- Process of locating moving object(s) across frames

Applications:
- Cruise Control
- Pedestrian Tracking
- Interactive Robots
Feature Tracking: Algorithm

- Image Preprocessing
- Gradient Edge Detection

- Noise filtered image
- Horizontal edge image
- Vertical edge image

Highly parallel: Each output pixel can be computed in parallel with others
Feature Tracking: Algorithm

- Image Preprocessing
- Gradient Edge Detection
- Feature Extraction

Features: \( \{x, y, \text{strength}\} \)
Feature Tracking: Algorithm

1. Image Preprocessing
2. Gradient Edge Detection
3. Feature Extraction
4. Feature Matching

Frame 1 Features:
- \(< x, y, strngt>\)
- \(< 0,0, 25>\)
- \(< ... >\)

Frame 2 Features:
- \(< x, y, strngt>\)
- \(< 6,6, 21>\)
- \(< 1,1, 12>\)
- \(< 0,2,22>\)

Select Nearby Features (e.g., within a range of 5x5) then pick one with closest strength.

Correlated, with Vector

Intro  B1  B2  B3  B4  B5  B6  B7  B8  B9  Results  Related Work  Conclusion
Feature Tracking: Algorithm

- Image Preprocessing
- Gradient Edge Detection
- Feature Extraction
- Feature Matching

Feature Tracking

- Gradient
- GaussianFilter
- IntegralImage
- Areasum

Percentage Execution Time

Relative input size 1X, 2X, 4X - (12k, 25k, 100k pixels)
Disparity Map: Overview

- Computes relative positions of objects in a scene captured by stereo cameras – depth
Disparity Map: Algorithm

Image Pre-Processing → Correlate left and right pairs → Find Disparity Map

Disparity Map

Percentage Execution Time

Relative input size 1X, 2X, 4X

Intro | B1 | B2 | B3 | B4 | B5 | B6 | B7 | B8 | B9 | Results | Related Work | Conclusion
Image Stitch: Overview

- Combining multiple images with overlapping view
Image Stitch: Overview

• Combining multiple images with overlapping view

Applications
• Movie making
• Google earth, GPS applications etc
Image Stitch: Analysis

Image pre-processing → Feature Detection → Match Features → Stitch Images

Image Stitching

- LSSolver
- SVD
- Convolution
- NonKernelWork

Percentage Execution Time vs. Relative input size 1X, 2X, 4X
Scale Invariant Feature Transform (SIFT): Overview

Detection and description of robust local image features

Applications

- Object recognition
- Motion Analysis
- Gesture Recognition
SIFT: Algorithm

Image Processing and Pyramid Generation → Feature Pyramid → Keypoint Detection → Remove low contrast key points

Difference of Gaussians Pyramid – Edge Images → Corner detection → Remove low contrast key points
Image Segmentation: Overview

Process of partitioning an image into meaningful segments, typically used to locate objects and boundaries.
Using the continuous edge image and high contrast matrix, we mark segments.
Support Vector Machine (SVM): Overview

Separation of a given set of data into two categories with maximal geometric margin.

Applications
- Machine Learning
- Neural networks
- Data classification
SVM: Algorithm

Train the classifier based on the training data

Evaluate the classifier based on the testing data

Refine the classifier based on the testing data

Qualify of data classification
Improves with number of Iterations used in learning stage.

The execution time depends on the number of data points and number of iterations.
Robot Localization: Overview

Process of evaluating path based on obstacles and a set of goals

Applications
- Space exploration programs
- Autonomous vehicles/systems
- Mobile robotics
Robot Localization: Algorithm

Global Position Estimation

Local Position Tracking

Probabilistic models to estimate position

Robot Localization

ParticleFilter
Sampling
NonKernelWork

Percentage Execution Time

Intro  B1  B2  B3  B4  B5  B6  B7  B8  B9  Results  Related Work  Conclusion
Texture Synthesis: Overview

Constructing large images from small image using structural content

Applications
- Movie making
- Graphics
- Computational Photography
Texture Synthesis: Algorithm

Image Pre-processing → Texture Analysis → Texture Synthesis

Texture Synthesis

Percentage Execution Time

- Sampling
- Conjugate Matrix
- PCA
- NonKernelWork

Intro | B1 | B2 | B3 | B4 | B5 | B6 | B7 | B8 | B9 | Results | Related Work | Conclusion

(Texture Analysis) (Texture Analysis) (Texture Synthesis)
Face Detection: Overview

Determines locations and sizes of human faces

Applications
• Object tracking
• Digital Camera, Facebook
• Content based image retrieval
Results: Execution Times

These times are for 1 or 2 frames of computation on a single core.

Lots of potential for making use of multi-core processors! Far enough away that we can use lots of cores, but close enough that it’s attainable with more cores.

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>12k pixels</th>
<th>25k pixels</th>
<th>100k pixels</th>
</tr>
</thead>
<tbody>
<tr>
<td>Feature tracking</td>
<td>2.77</td>
<td>5.0</td>
<td>19.4</td>
</tr>
<tr>
<td>Disparity Map</td>
<td>0.8</td>
<td>1.8</td>
<td>6.2</td>
</tr>
<tr>
<td>Image Stitch</td>
<td>0.7</td>
<td>10.1</td>
<td>23.4</td>
</tr>
<tr>
<td>SIFT</td>
<td>17.4</td>
<td>44.5</td>
<td>131</td>
</tr>
<tr>
<td>Texture Synthesis</td>
<td>18.5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Image Segmentation</td>
<td>8.3</td>
<td>9.2</td>
<td>8.4</td>
</tr>
<tr>
<td>SVM</td>
<td>Up to Thousands</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Seconds per Frame
### Results:

**Parallelism in Vision Kernels**

**Parallelism calculation:**
- We implemented a transformation pass in LLVM infrastructure track operands through the program and determine the longest dependence chain through memory, control and instruction dependences.
- Parallelism = # Instrs / Critical Path Length

The benchmarks shown on the right have lots of parallelism!

*(If you are curious, we encourage you to go ahead and download our benchmark suite!)*

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Kernel</th>
<th>Approx. Parallelism</th>
</tr>
</thead>
<tbody>
<tr>
<td>Disparity</td>
<td>Correlation</td>
<td>502</td>
</tr>
<tr>
<td></td>
<td>Integral Image</td>
<td>160</td>
</tr>
<tr>
<td></td>
<td>Sort</td>
<td>1,700</td>
</tr>
<tr>
<td></td>
<td>SSD</td>
<td>1,800</td>
</tr>
<tr>
<td>Feature Tracking</td>
<td>Gradient</td>
<td>71</td>
</tr>
<tr>
<td></td>
<td>Gaussian Filter</td>
<td>637</td>
</tr>
<tr>
<td></td>
<td>Integral Image</td>
<td>1,050</td>
</tr>
<tr>
<td></td>
<td>Area Sum</td>
<td>425</td>
</tr>
<tr>
<td></td>
<td>Matrix Inversion</td>
<td>171,000</td>
</tr>
<tr>
<td>SIFT</td>
<td>SIFT</td>
<td>180</td>
</tr>
<tr>
<td></td>
<td>Interpolation</td>
<td>502</td>
</tr>
<tr>
<td></td>
<td>Integral Image</td>
<td>16,000</td>
</tr>
<tr>
<td>Image Stitch</td>
<td>LS Solver</td>
<td>20,900</td>
</tr>
<tr>
<td></td>
<td>SVD</td>
<td>12,300</td>
</tr>
<tr>
<td></td>
<td>Convolution</td>
<td>4,500</td>
</tr>
<tr>
<td>SVM</td>
<td>Matrix Ops</td>
<td>1,000</td>
</tr>
<tr>
<td></td>
<td>Learning</td>
<td>851</td>
</tr>
<tr>
<td></td>
<td>Conjugate Matrix</td>
<td>502</td>
</tr>
</tbody>
</table>
Related Work: Other vision codes

• Performance-oriented benchmarks
  – PARSEC: Body Tracking
  – MediaBench – image/video compression algorithms
  – Spec2000 facerec

• Accuracy-oriented benchmarks (for vision research)
  – Berkeley Segmentation Database and Benchmark
  – PEIPA
  – Muscle
  – ImageCLEF

• Vision Libraries
  – OpenCV- Highly tuned vision toolbox
# Related Work: Intel OpenCV vs SD-VBS

<table>
<thead>
<tr>
<th></th>
<th>OpenCV</th>
<th>SD-VBS</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Goal</strong></td>
<td>Tuned, Feature-loaded implementations for commercial and academic vision applications</td>
<td>“Pure” versions for easy analysis, transformation and parallelization in multi-core architecture research</td>
</tr>
<tr>
<td><strong>Source Code</strong></td>
<td>C++</td>
<td>C or MATLAB, your choice</td>
</tr>
<tr>
<td><strong>Platform specific optimizations</strong></td>
<td>Highly Tuned for best performance on Intel and supporting architectures</td>
<td>Actively removed optimizations that increase code complexity</td>
</tr>
<tr>
<td><strong>Coding style</strong></td>
<td>Highly flexible; full of options to change behavior of each function</td>
<td>Clean code without features that deter analysis</td>
</tr>
<tr>
<td><strong>Ease of analysis</strong></td>
<td>Harder</td>
<td>Easier, because simpler implementations</td>
</tr>
</tbody>
</table>

**Example:** FILTER

- 2000 lines of code
- 204 conditional statements
- Pointer Operations

**Ease of analysis**

- Harder
- Easier, because simpler implementations
Conclusions

• Computer Vision is an exciting domain with immense potential
• Vision algorithms are full of parallelism, and can benefit from processors with greater and greater performance; which make them ideal for multi-core
• SD-VBS is a comprehensive and clean benchmark suite for vision, well suited for multi-core and many-core research.

Public release

http://parallel.ucsd.edu/vision
SD-VBS:
The San Diego Vision Benchmark Suite

http://parallel.ucsd.edu/vision

We would like to thank the following Vision researchers:

• Andrea Vedaldi
• Athanasois Noulas
• Tom Mitchell
• Javier Portilla
• Eero Simoncelli
• Jianbo Shi

For contributing their code and data sets to the benchmark suite.

Supported by NSF CAREER Award 0846152