A Hybrid CPU-GPU Approach to Fourier-Based Image Stitching of Optical Microscopy Images

Walid Keyrouz, Timothy Blattner, Bertrand Stivala, Joe Chalfoun, and Mary Brady

Software and Systems Division, Information Technology Laboratory
National Institute of Standards and Technology
Gaithersburg, MD 20899-8070
Contact: walid.keyrouz@nist.gov

Abstract: We present a hybrid CPU-GPU approach for the Fourier-based stitching of optical microscopy images. This system achieves sub-minute stitching rates with large grids; it stitches a grid of 59x42 tiles in 26 seconds on a two-CPU (8 physical cores) & two-GPU machine. This is a speedup factor of more than 24x; the optimized sequential implementation takes more than 10 minutes to perform the same task. The system scales to take advantage of additional CPU cores or GPU cards. For the sake of comparison, ImageJ/Fiji, which uses a similar algorithm, exceeds 3.6 hours on the same workload.

Project
Goal: Use computing to enable and accelerate biological measurements
Objectives:
- Image stitching of optical microscopy images at interactive rates
- General purpose library
Motivation:
- Scientists are now frequently acquiring tiled images
- Stitched images essential for measurements based on acquired images

Contributions
Separate execution pipeline per GPU
- Scalable across multiple GPUs
Host-side pipelined threading organization
- Overlaps compute and data transfers
- Uses all available system resources

Evaluation Platform
Hardware
- Dual Intel® Xeon® E-5620 CPUs
  - Quad-core, 2.4 GHz, hyper-threading
- 48 GB RAM
- Dual NVIDIA® Tesla® C2070 cards
Software
- Ubuntu 12.04/x64, kernel 3.2.0
- Libb6 2.21, libbtdib+6.4.6
- FFTW & CUFFT 5.0

Data Set:
- Grid of 59x42 images (2478)
- 1040x1392 16-bit grayscale images (2.8 MB per image)
- Total: ~6.7 GB

Image Stitching Problem
Three phases of image stitching:
1. Compute the X & Y translations for all image tiles
2. Eliminate over-constraint through global optimization
3. Apply the computed translations & compose into one image
- Main focus is on first phase.

Algorithm
Loop over all images:
1. Read an image
2. Compute its Forward 2D Fourier Transform (FFT-2D)
3. Compute correlation coefficients with west and north neighbors
   - Depends on FFT-2D
   - NCC & index of max give (x,y) disp.

Fourier Transforms

<table>
<thead>
<tr>
<th>FFTW (fftw.org)</th>
<th>Planning Time</th>
<th>Execution Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Auto-tuning FFT-2D plan</td>
<td>0.02 s</td>
<td>137.7 ms</td>
</tr>
<tr>
<td>Planning mode specifies effort to find &quot;best&quot; FFT algorithm</td>
<td>Measure 4 min 23 s</td>
<td>66.1 ms</td>
</tr>
<tr>
<td>Amortized planning cost</td>
<td>Patient 4 min 23 s</td>
<td>66.1 ms</td>
</tr>
<tr>
<td>Run prior to stitching computation</td>
<td>Exhaustive 7 min 1 s</td>
<td>66.1 ms</td>
</tr>
</tbody>
</table>

Implementations
Reference: Sequential Implementation
Simple Multi-Threaded Simple GPU Pipelined Multi-Threaded Pipelined GPU

Results

<table>
<thead>
<tr>
<th>Operation</th>
<th>w/o Intrinsics</th>
<th>SSE Intrinsics</th>
<th>GPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NCC</td>
<td>Avg. Time</td>
<td>Time Speedup</td>
<td>Avg. Time</td>
</tr>
<tr>
<td>Max Reduction</td>
<td>27 ms</td>
<td>5.9 ms</td>
<td>45.7 x</td>
</tr>
</tbody>
</table>

Simple Multi-Threaded Implementation

- Spatial domain decomposition, one thread per partition
- Explicit handling of inter-partition dependencies (red arrows in figure below)

Pipelined Multi-Threaded Implementation

- 5-stage pipeline:
  - Stages communicate via queues with synchronization using mutexes
  - 8 threads 1 for stages 1, 2, 3, & 4; 1 book-keeping thread for each of stages 3 & 5
- 3-stage pipeline version merges stages 2, 4, 5 & 3. Had minimal performance change

Pipelined GPU Implementation

- Adapts pipelined multi-threaded implementation to GPU
- 1 execution pipeline per GPU
- Distribute image grid evenly between GPUs
- Seven stage pipeline:
  - 1 queue and CPU thread per stage per GPU for read, copy, FFT, and Disp.
  - Overlaps copies and compute on GPUs
  - BK gathers FFTs and distributes pairs of FFTs to GPUs using tile grid
  - BK frees GPU memory using tile grid
  - BK & BK use a spinlock on tile grid to prevent race conditions
- CDFs computed with pool of GPU threads

Remarks

- No benefit from direct port of sequential version to GPU
- Simple multi-threaded suffers from load imbalance
- Load imbalance handled by pipelined implementation
- FFTW & CUFFT sensitive to vector sizes (should be powers of 2, 3, or 5)

Future Work
- Explore padding & single precision complex FFTs
- Use different data sets & compare with other benchmarks
- Experiment with alternative GPU architectures & accelerator cards

Disclaimer: No approval or endorsement of any commercial product by the National Institute of Standards and Technology (NIST) is intended or implied. Certain commercial software, products, and systems are identified in this report to facilitate better understanding. Such identification does not imply recommendations or endorsement by NIST nor does it imply that the software and products identified are necessarily the best available for the purpose.