High Quality Real Time Image Processing Framework on Mobile Platforms using Tegra K1

Eyal Hirsch
SagivTech Ltd. proprietary information - for internal use only

SagivTech Snapshot

• Established in 2009 and headquartered in Israel

• Core domain expertise: GPU Computing and Computer Vision

• What we do:
  - Technology
  - Solutions
  - Projects
  - EU Research
  - Training

• GPU expertise:
  - Hard core optimizations
  - Efficient streaming for single or multiple GPU systems
  - Mobile GPUs
Mobile is everywhere

- The new era of mobile
As mobile devices get smarter

• In the beginning: I can talk from anywhere!
• A bit later: My phone can take pictures!
• Now:
  – Advanced camera
  – More compute power
  – Fast device – cloud communication
• What can be done with those advancements?
Project Tango

• Mission: Running a depth sensing technology on a mobile platform

• Challenge: First time on NVIDIA’s Tegra K1

• Extreme optimizations on a CPU-GPU platform to allow the device to handle other tasks in parallel

• Expertise:
  • Mantis Vision – the algorithms
  • NVIDIA – the Tegra K1 platform
  • SagivTech – the GPU computing expertise

• Bottom line: Depth sensing running in real time in parallel to other compute intensive applications!
Project Tango

Credits: http://techaeris.com
Mobile Crowdsourcing Video Scene Reconstruction

• If you’ve been to a concert recently, you’ve probably seen how many people take videos of the event with mobile phone cameras

• Each user has only one video – taken from one angle and location and of only moderate quality
The Idea behind SceneNet

• Leverage the power of multiple mobile phone cameras
• Create a high-quality 3D video experience that is sharable via social networks
Creation of the 3D Video Sequence

The scene is photographed by several people using their cell phone camera.

The video data is transmitted via the cellular network to a High Performance Computing server.

Following time synchronization, resolution normalization and spatial registration, the several videos are merged into a 3-D video cube.
Algorithms implemented on the TK1

- Enabling the 3D reconstruction for SceneNet required various algorithms to run on the TK1 GPU
  - FREAK: Fast Retina Key point
  - BRISK: Binary Robust Invariant Scalable Key points
  - DoG: Difference of Gaussians

- Algorithms had to run in real-time
- Algorithms are image processing building blocks for various image processing tasks
Freak & DoG performance on the TK1

- **DoG:**
  - Input: 480 x 640 RGB Image
  - Output: ~32K key points

- **Freak:**
  - Input: ~32K key points, Image
  - Output: Descriptor per key point

- **Majority of the code on the GPU**

- **Off loading to the GPU allows for real time processing, not possible on the CPU**
DoG performance on the TK1

- DoG flow:
  - Gaussian
  - DiffImage
  - Find Key points

- Total: 10.83 ms

<table>
<thead>
<tr>
<th>Kernel</th>
<th>Avg. time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Misc</td>
<td>0.3</td>
</tr>
<tr>
<td>Gaussian: Conv2D</td>
<td>4.8</td>
</tr>
<tr>
<td>Gaussian: DownSampleBilinear</td>
<td>0.6</td>
</tr>
<tr>
<td>DiffImage</td>
<td>1.7</td>
</tr>
<tr>
<td>FindKeyPoints</td>
<td>3.43</td>
</tr>
<tr>
<td>Total DoG</td>
<td>10.83</td>
</tr>
</tbody>
</table>
FREAK performance on the TK1

- **FREAK flow:**
  - IntegralImage
  - Extract descriptors

- **Total:** 2.4 ms

<table>
<thead>
<tr>
<th>Kernel</th>
<th>Avg. time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>IntegralImage</td>
<td>1.5</td>
</tr>
<tr>
<td>GetDescriptors</td>
<td>0.9</td>
</tr>
<tr>
<td>Total FREAK</td>
<td>2.4</td>
</tr>
</tbody>
</table>

- **Total DoG + FREAK:** 13.23 ms
Freak &DoG performance on the TK1

- 13 ms means real time processing on Ardbeg development board !!!
- Room for more tasks to run in the background
- Opens up possibilities for many mobile applications
- Having real time performance is not enough
- Need to evaluate power consumption as well
Performance is also GFlops/WATT
Programming the TK1 GPU

- CUDA – NVIDIA
- OpenCL – Khronos
- RenderScript – Developed by Google
Programming the TK1 - CUDA

- Most rules and methods that apply to discrete cards, apply to the TK1 GPU
- Code and libraries (such as cuFFT, cuBLAS, cuSPARSE, CUB, Thrust, etc) should work out of the box for the TK1
- Develop on Windows/Linux with discrete card and then migrate to the TK1
- Use the profiler
Programming the TK1 - OpenCL

• Most of the tips for CUDA applies to OpenCL
• Runs nicely and shows nice performance
• Migrated the in-house Bilateral filter from CUDA to OpenCL in less than a day
• 2D separable convolution yield nice performance gains (compared to an optimized Neon implementation)
2D separable convolution on the TK1

- Used 4 tests configuration to evaluate performance
  - Highly optimized reference library utilizing the NEON (CPU)
  - SagivTech’s in-house Neon implementation (CPU)
  - SagivTech’s in-house OpenCL implementation (GPU)

<table>
<thead>
<tr>
<th>Test configuration</th>
<th>1K x 1K</th>
<th>2K x 2K</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reference library</td>
<td>22</td>
<td>97</td>
</tr>
<tr>
<td>ST single core NEON</td>
<td>23.5</td>
<td>99</td>
</tr>
<tr>
<td>ST 4 cores NEON</td>
<td>10.8</td>
<td>48</td>
</tr>
<tr>
<td>ST OpenCL</td>
<td>4</td>
<td>9</td>
</tr>
</tbody>
</table>
Programming the TK1 – RenderScript - 1

• Google’s way of doing Compute on a mobile platform
• Quick CUDA to RenderScript acronym translation:
  – User manages allocations (a.k.a buffers)
  – User manages data transfer/copies to/from allocations
  – User sets runtime parameters (a.k.a kernel params)
  – User launches kernels much like OpenCL/CUDA
• Code ran on the GPU and yielded impressive performance boost (still lags behind CUDA)
• CUDA to RS migration fairly easy
• Google does NOT mandate which SoC component will run the RS code
• Developer has no control where RS code will run
• Depends on specific hardware, vendors, code, etc
• To test RS on TK1, locked GPU clocks in different configurations and run RS sparse matrix vector multiplication benchmark
• Performance of the RS code under different clocks, would reveal which component ran RS code
Programming the TK1 – RenderScript - 3

• Sparse matrix vector multiplication using Render script

• Used 3 test configurations
  – Naive C++ CPU code
  – SagivTech RS
  – NVIDIA’s cuSparse

• RS running on GPU

• RS shows nice performance
Programming the TK1 – Optimization tips

• Only one SMX
• We’ve seen cases where different optimizations behave differently on the TK1 than on equivalent discrete card (such as __ldg etc)
• Try various optimizations, in some cases we got better performance when using atomics rather than shared memory
• Always optimize on the TK1 and not on discrete used for the development phase
The future

• Real time image processing of even complex algorithms is achievable on the TK1
• Easy migration from mature discrete GPU code to new and exiting field of mobile compute
• Maxwell is already planed for next mobile generation, bringing more power efficiency and performance
• It works!!
Thank You

For more information please contact

Eyal Hirsch

eyal@sagivtech.com
Programming the TK1 – General tips

• TK1 hardware CC is 3.2
• Tools and compilation chain is quite different. Need some time to get started
• Strive to do the CUDA and managing app/code in Windows/Linux using a discrete card and then migrate to Android
• Always have a reference code in naive, single thread C++ to compare the results of the parallel algorithm
Computational Photography: examples …

• Background substitution
FREAK – Fast Retina Keypoint

- Binary feature descriptor
- Hamming distance matcher
- Sampling pattern
  - Overlapping receptive fields
  - Exponential change in size
- Rotation invariant
BRISK – Binary Robust Invariant Scalable Key points

- Binary feature descriptor
- Hamming distance matcher
- Sampling pattern
  - Equally spaced in circles
  - Gaussian kernel size relative to distance from feature
DoG – Difference of Gaussians

- Feature detector
- Local minima/maxima of the image convolved with difference of gaussians
- Acts as a 2D band-pass filter over the image
- Enhances corners

![Difference Of Gaussians](image)