Using GPUs for Image Processing

Ben Baker

Sponsored by:

Agenda

- Background
  - GPU Computing
  - Digital Image Processing at FamilySearch
- Potential GPU based solutions
- Performance Testing Results
- Conclusions and Future Work

FAMILYSEARCH
WHERE GENERATIONS MEET

CPU vs. GPU Architecture

CPU

- General purpose processors
- Optimized for instruction level parallelism
- A few large processors capable of multi-threading

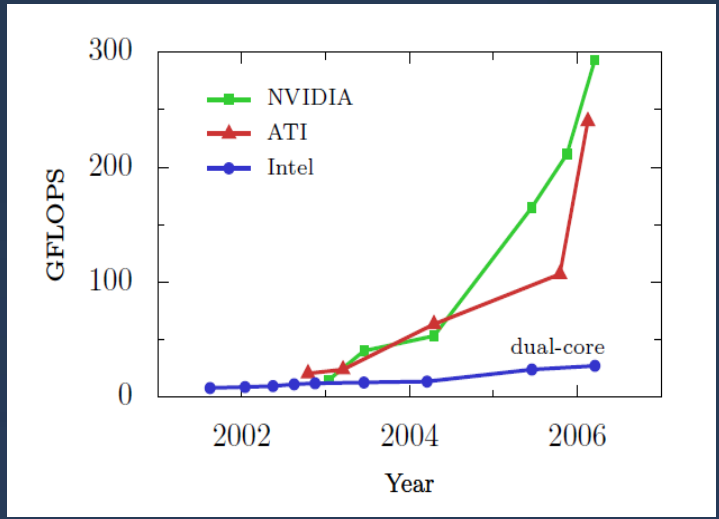
GPU

- Special purpose processors
- Optimized for data level parallelism
- Many smaller processors executing single instructions on multiple data (SIMD)

FAMILYSEARCH
WHERE GENERATIONS MEET

High Performance GPU Computing

- GPUs are getting faster more quickly than CPUs
- Being used in industry for weather simulation, medical imaging, computational finance, etc.
- Amazon is now offering access to Tesla GPUs as a service



Using GPUs as general purpose parallel processors

http://gpgpu.org

FAMILYSEARCH
WHERE GENERATIONS MEET

Marketing from NVIDIA

- "World's Fastest 1U Server"

  "Compared to typical quad-core CPUs, Tesla 20 series computing systems deliver equivalent performance at 1/10th the cost and 1/20th the power consumption."
- "Personal Supercomputer"

  "250x the computing performance of a standard workstation"
- Dell is now selling a 3U rack mount unit capable of holding 16 GPUs connected to 8 servers

Computer Graphics vs. Computer Vision

- Approximate inverses of each other:
  - Computer graphics – converting "numbers into pictures"
  - Computer vision – converting "pictures into numbers"
- GPUs have traditionally been used for computer graphics – (Ex. Graphics intensive computer games)
- Recent research, hardware and software are using GPUs for computer vision (Ex. Using Graphics Devices in Reverse)
- GPUs generally work well when there is ample data-level parallelism

Digital Processing Center (DPC)

- Collection of multiple servers in a data center used by FamilySearch to continuously process millions of images annually
- Computer Vision types of Image Processing performed include
  - Automatic skew correction
  - Automatic document cropping
  - Image sharpening
  - Image scaling (thumbnail creation)
  - Encoding into other image formats
- CPU is current bottleneck (~12 sec/image)

Promising "Drop In" Technologies

CPU Technologies

- IPP (Intel Performance Primitives)
  - Use DCT functions to accelerate JPEG compression
- OpenCV
- Kakadu for JPEG-2000 encoding/decoding

GPU Technologies

- NPP (NVIDIA Performance Primitives)
  - No license for SDK like Intel's IPP
  - Has DCT functions that could be used for JPEG compression
- Dec 2010 release of OpenCV has GPU module
- GPUCV
- Cuj2k (CUDA JPEG-2000)

IPP to NPP Conversion

- NVIDIA replicated API of Intel's IPP, so implemented methods are fairly easy to use
- Learning curve about copying to/from GPU and allocating memory on GPU
- Didn't have time yet to try other libraries or direct programming on GPU in CUDA
- Got cropping and sharpening operations implemented

Example – Convolution Filter

```
… [Create padded image]
… [Create Gaussian kernel]
```

```
// Allocate blurred image of appropriate size
Ipp8u* blurredImg = ippiMalloc_8u_C1(img.getWidth(),
     img.getHeight(),
      &blurredImgStepSz);
// Perform the filter
ippiFilter32f_8u_C1R(paddedImgData,
     paddedImage.getStepSize(), blurredImg,
     blurredImgStepSz, imgSz, kernel, kernelSize,
     kernelAnchor);
```

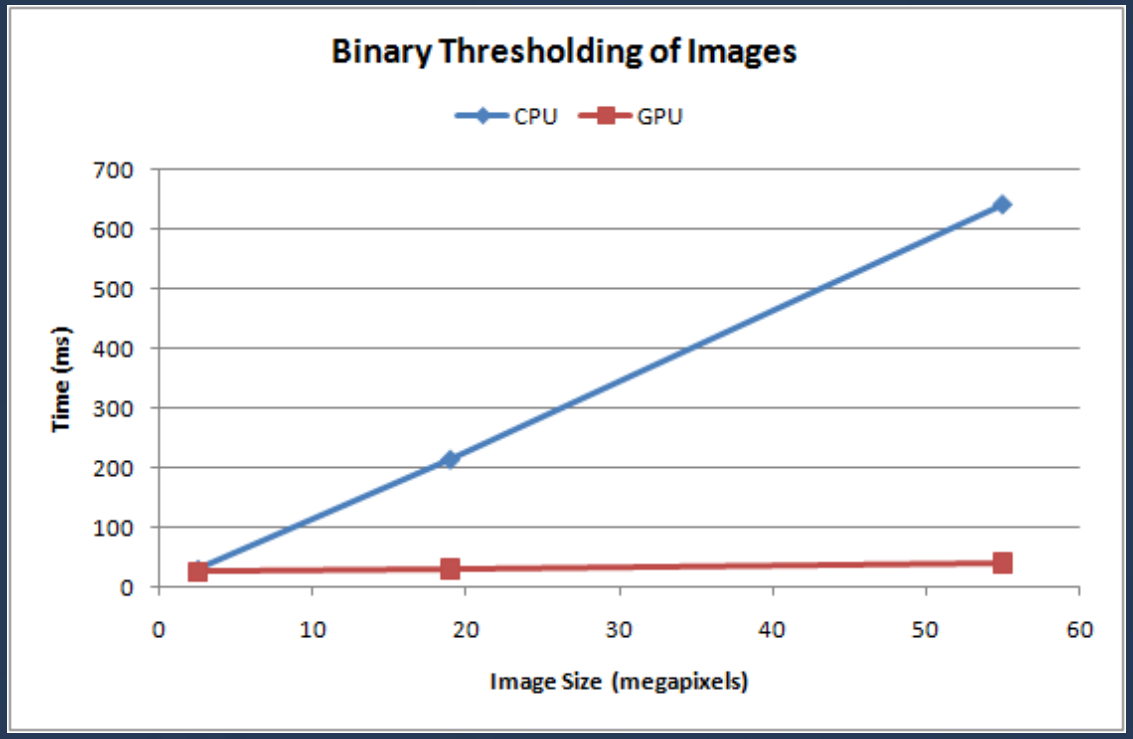
```
// Declare a host object for an 8-bit grayscale image
npp::ImageCPU_8u_C1 hostSrc;
// Load grayscale image from disk
npp::loadImage(sFilename, hostSrc);
// Declare a device image and upload from host
npp::ImageNPP_8u_C1 deviceSrc(hostSrc);

… [Create padded image]
… [Create Gaussian kernel]

// Copy kernel to GPU
cudaMemcpy2D(deviceKernel, 12, hostKernel,
     kernelSize.width * sizeof(Npp32s), kernelSize.width
     * sizeof(Npp32s), kernelSize.height,
     cudaMemcpyHostToDevice);
// Allocate blurred image of appropriate size (on GPU)
npp::ImageNPP_8u_C1 deviceBlurredImg(imgSz.width,
     imgSz.height);

// Perform the filter
nppiFilter_8u_C1R(paddedImg.data(widthOffset,
     heightOffset), paddedImg.pitch(),
     deviceBlurredImg.data(), deviceBlurredImg.pitch(),
     imgSz, deviceKernel, kernelSize, kernelAnchor,
     divisor);
// Declare a host image for the result
npp::ImageCPU_8u_C1
     hostBlurredImage(deviceBlurredImg.size());
// Copy the device result data into it
deviceBlurredImg.copyTo(hostBlurredImg.data(),
     hostBlurredImg.pitch());
```

Performance Testing Methodology

- Test System
  - Dual Quad Core Intel® Xeon® 2.83GHz E5440 CPUs (8 cores total)
  - 16 GB RAM
  - Debian Linux operating system
  - Single Tesla C1060 Compute Processor (240 processing cores total)
  - PCI-Express x16 Gen2 slot
- Three representative grayscale images of increasing size
  - Small image – 1726 x 1450 (2.5 megapixels)
  - Average image – 4808 x 3940 (18.9 megapixels)
  - Large image – 8966 x 6132 (55.0 megapixels)

Cropping

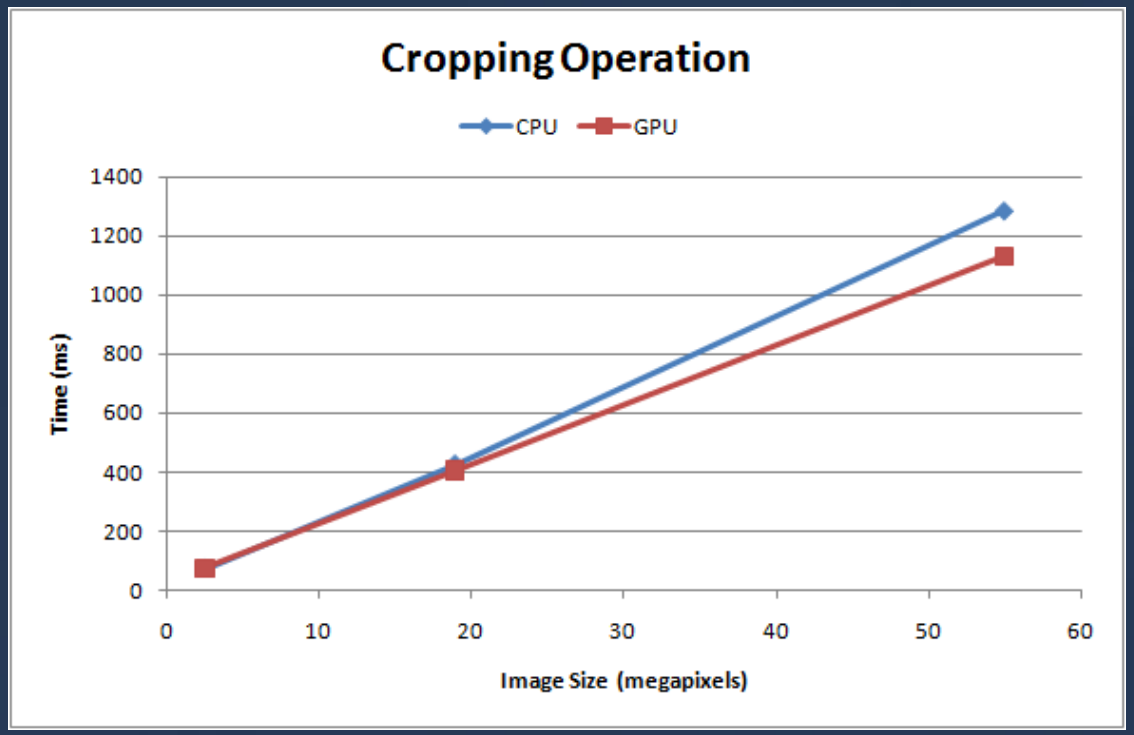
1. Compute a threshold value
2. Binarize the image based on the computed threshold
3. Compute a bounding box that encloses all pixels determined as part of the document



Good News ☺

Creating a binarized image is up to 16x faster on the GPU!

The larger the image, the more effective using the GPU is



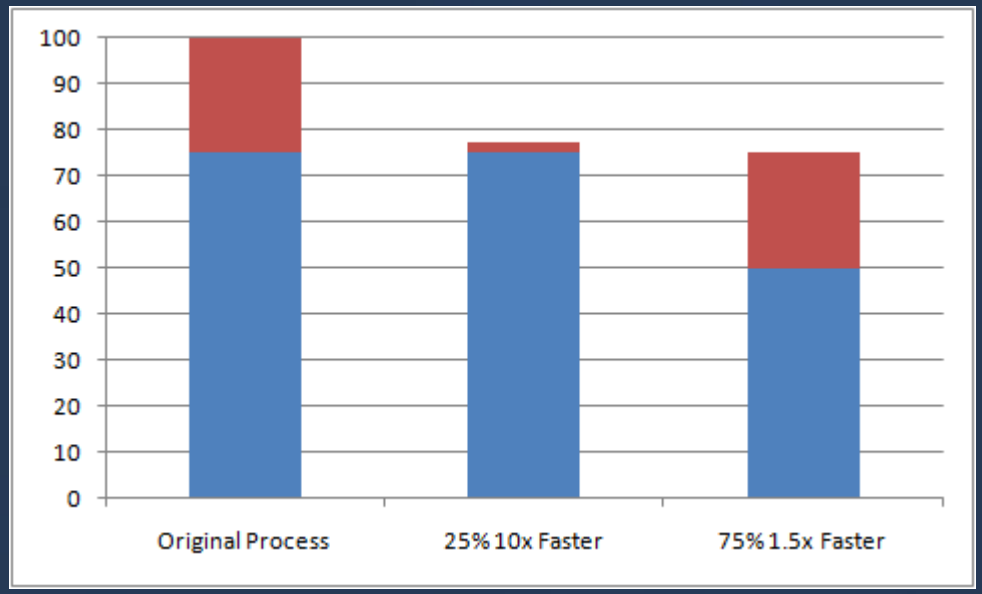
Bad News ☹

Large portion of operation not yet optimized on the GPU

Result is only up to 14% faster than the CPU implementation

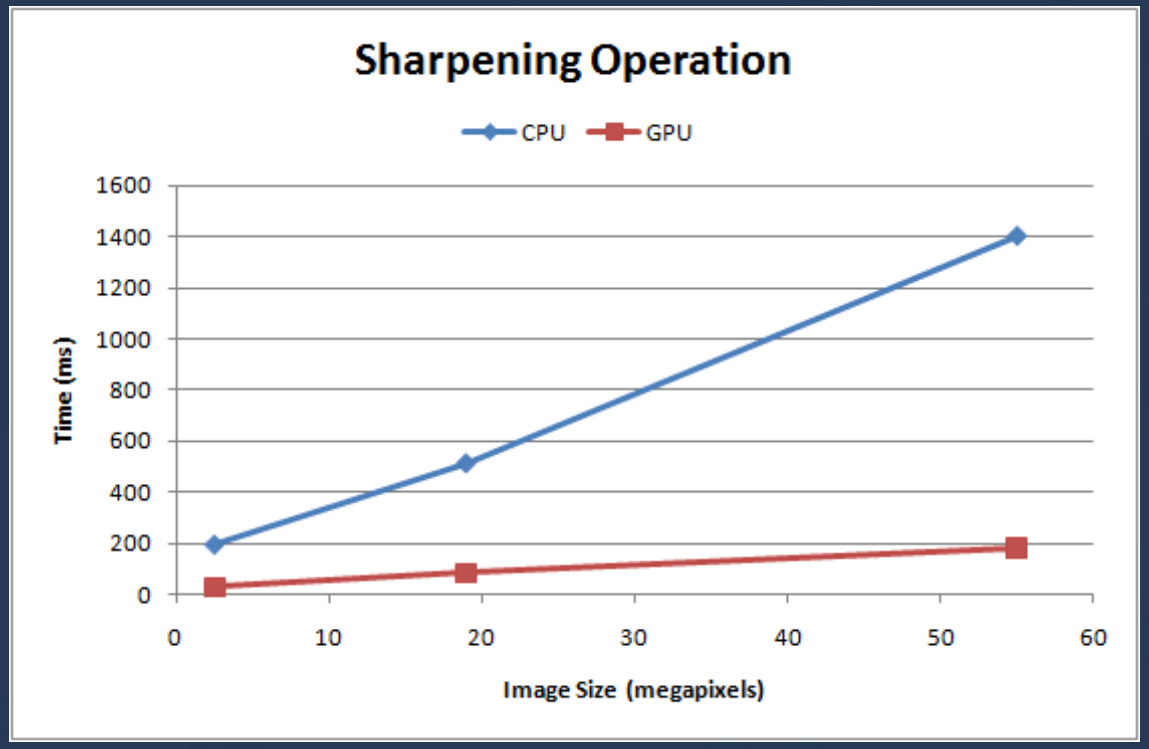
Amdahl's Law

Speeding up 25% of an overall process by 10x is less of an overall improvement than speeding up 75% of an overall process by 1.5x



Sharpening (Unsharp Mask)

1. Perform a Gaussian Blur on the source image
2. Take the difference of the blurred image from the original and multiply it by a specified amount
3. Add the image produced from the previous step and clamp any values back to the displayable range of [0,255]

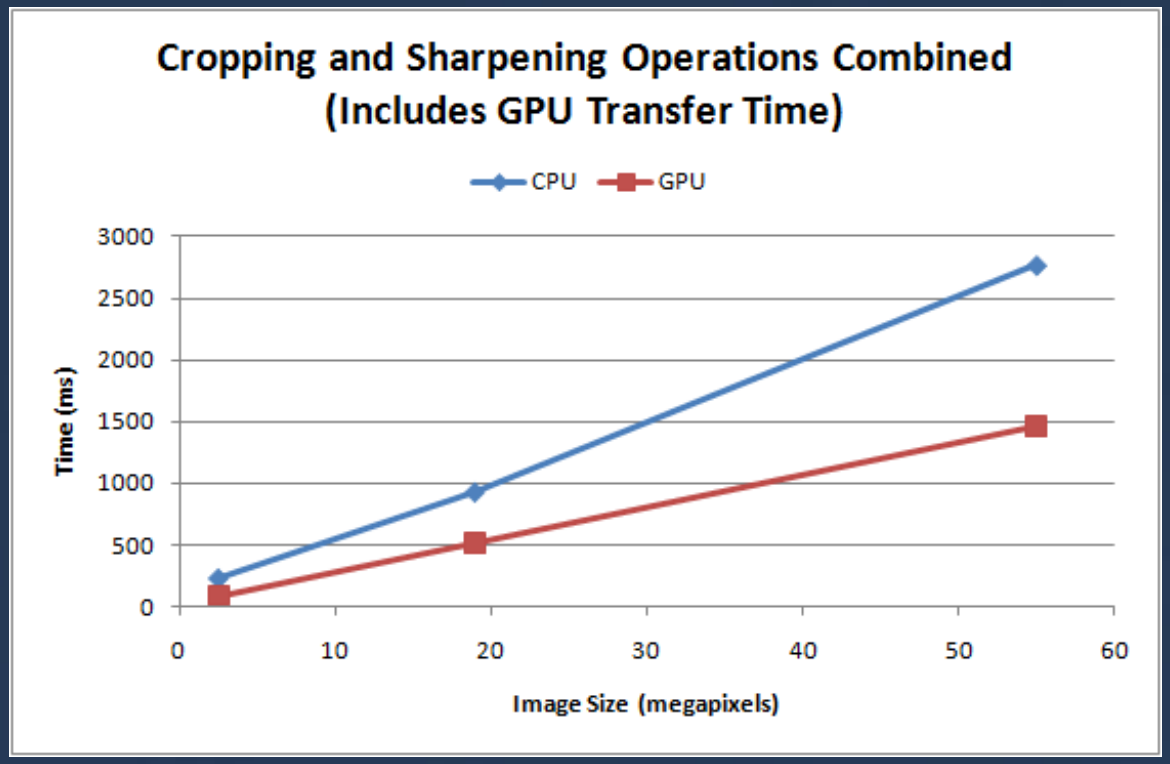


Able to completely run sharpening operation on GPU

Blur operation so fast it wasn't even a millisecond (infinite speedup ☺)

6 – 7x faster for all image sizes over current library

FAMILYSEARCH
WHERE GENERATIONS MEET



Operations in tandem result in roughly 2x speed increase

Will want to minimize transfer time to/from GPU

Conclusions and Future Work

- Significant increase in performance by parallelizing image processing operations for execution on GPUs
- Relatively easy implementation, but dependent on maturing libraries and tools

- Need to implement entire set of DPC operations (including image encoding/decoding) on GPU to fully assess viability
- Need to assess actual throughput in production-like environments

FAMILYSEARCH
WHERE GENERATIONS MEET

Other Potential Uses

- Accelerated image processing on workstations directly at archives
- Ability to use more sophisticated (and time consuming) algorithms
- Improve computationally intensive portions of client applications (Ex. image audit)
- Probably more

FAMILYSEARCH

WHERE GENERATIONS MEET

Thank You.

Sponsored by: