Accelerating a MATLAB Application with Nvidia GPUs: a Case Study for GPU Library Construction

Nicholas Moore, Miriam Leeser
{nmoore,mel}@coe.neu.edu

23 September 2009

Introduction

Exploring GPGPU Library Construction Issues

- Libraries faster than automatic tools
- Many differences to traditional CPU libraries
  - What are the important library parameters?
  - GPUs are accelerator boards

Outline

- CUDA implementation issues
- Support Infrastructure: Matlab OpenCL API (MOCA)
- Tumor tracking MATLAB application
- Implementation status and performance
- Observations and future directions

Mapping Functionality to CUDA

- Balancing several factors
  - Kernel complexity & limited register count
  - Mostly explicit memory placement options
    - Shared memory: can be as fast as registers, possible memory bank conflict issues
    - Texture and constant memory: read only and cached, texture memory optimized for 2D and 3D accesses
    - Global memory: slowest
  - CUDA blocks and threads hierarchy
    - Threads have access to shared data
    - Want enough threads to cover latencies
    - Need enough blocks to fill current & future GPUs
- Detailed knowledge of hardware is necessary to get best performance

Memory Selection Issue

- Different memory types
  - Performance varies
  - Limitations (size) vary
- Shared memory internal to block
- Global, constant, and texture memory can be used for kernel inputs
- What is the optimal selection for memory types?
  - Large search space
  - Each memory type has distinct usage in code

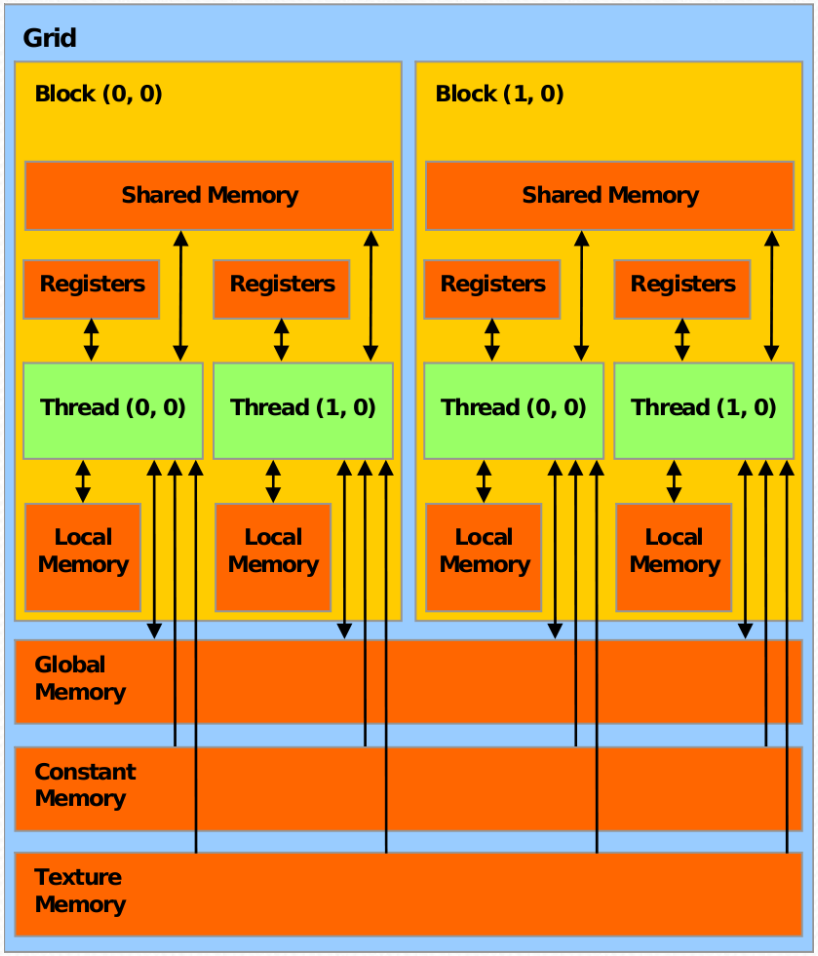


Image: NVIDIA CUDA Programming Guide 2.0
http://www.nvidia.com/object/cuda_develop.html

GPU Interfacing Issues

- Precision: GPU calculation generally in single precision floating point
  - Recent GPUs include support for double precision
    - 10x performance hit for Nvidia (less for ATI)
  - Likely many instances where single precision GPU is okay in a larger double precision application: prototyping, absolute speed, working with existing apps
    - Data type conversion needed
- Memory: CUDA requires CUDA-allocated host memory for asynchronous data transfers (also faster for large transfers)
  - Data copy may be introduced when the source memory type cannot be controlled

CPU/GPU Mapping Issues

- Want each algorithm step mapped to the right device (CPU or GPU) - affects:
  - Number/type of kernel invocations
  - Number and size of data transfers
  - Possibilities for concurrent execution
- Interplay of these issues can be complicated
  - Elimination of inefficient kernels
    - Extra data transfers for CPU computed values if base data still needed
  - Simultaneous CPU/GPU computation
    - Optimal mapping may put a given algorithm step on a suboptimal device

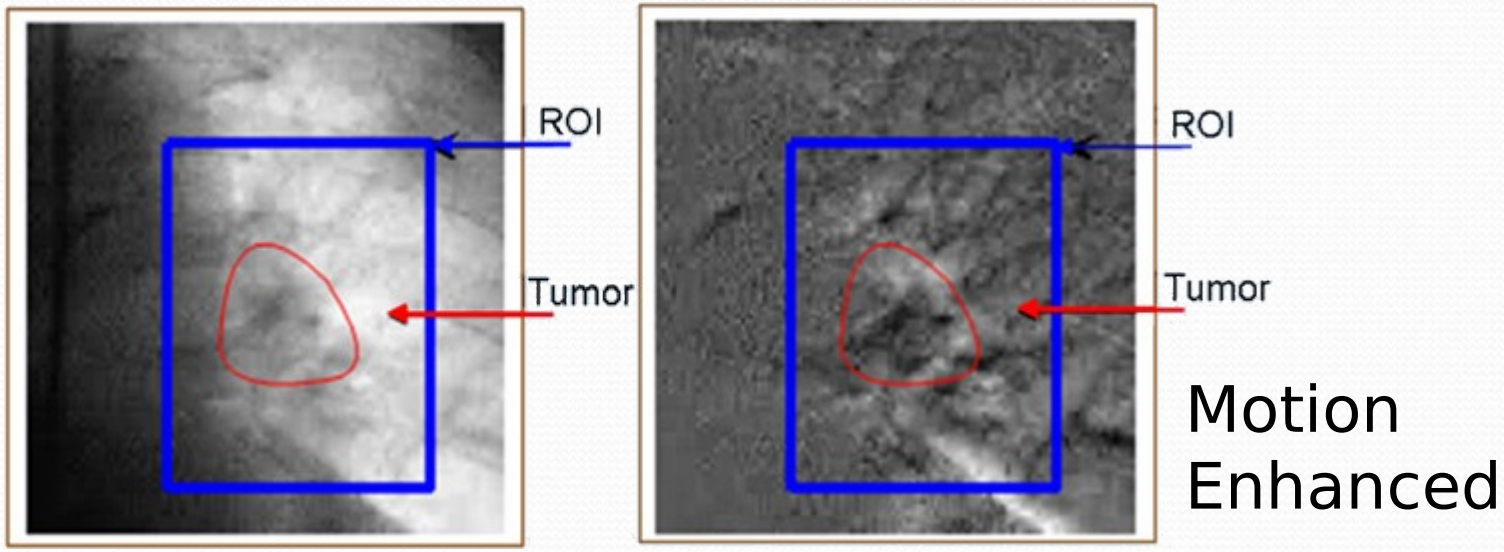
Matlab OpenCL API (MOCA)

- Large space for application mappings
  - Code management issues
    - Lots of similar code
    - Different code for different memory types
    - Managing host-side operations vs. GPU-kernel invocations
- MOCA aids implementation space exploration
  - Currently binds to CUDA, but aiming to be generic
  - Raises the level of abstraction for faster development
    - Data structures track multiple aspects of host and GPU resources
    - Functions wrap up numerous API calls into tasks
    - Front end catches some errors producing useful diagnostics
  - Isolates CUDA code for a given activity to one location
  - Focused on host code development

Lung Tumor Tracking

- Working with research by Ying Cui, Jennifer Dy, Gregory Sharp, Brian Alexander, and Steve Jiang
  - Represent Northeastern University, Massachusetts General Hospital, Harvard Medical School, and University of California San Diego
- Some tumors move significantly during breathing
  - Therapy targets large area with weak radiation – prevents damage normal tissue
  - Goal is to use higher-intensity focused radiation without implanted markers
- Two template-based tracking algorithms proposed:
  - Motion-enhanced templates and Pearson's correlation
  - Eigen templates and mean-squared error

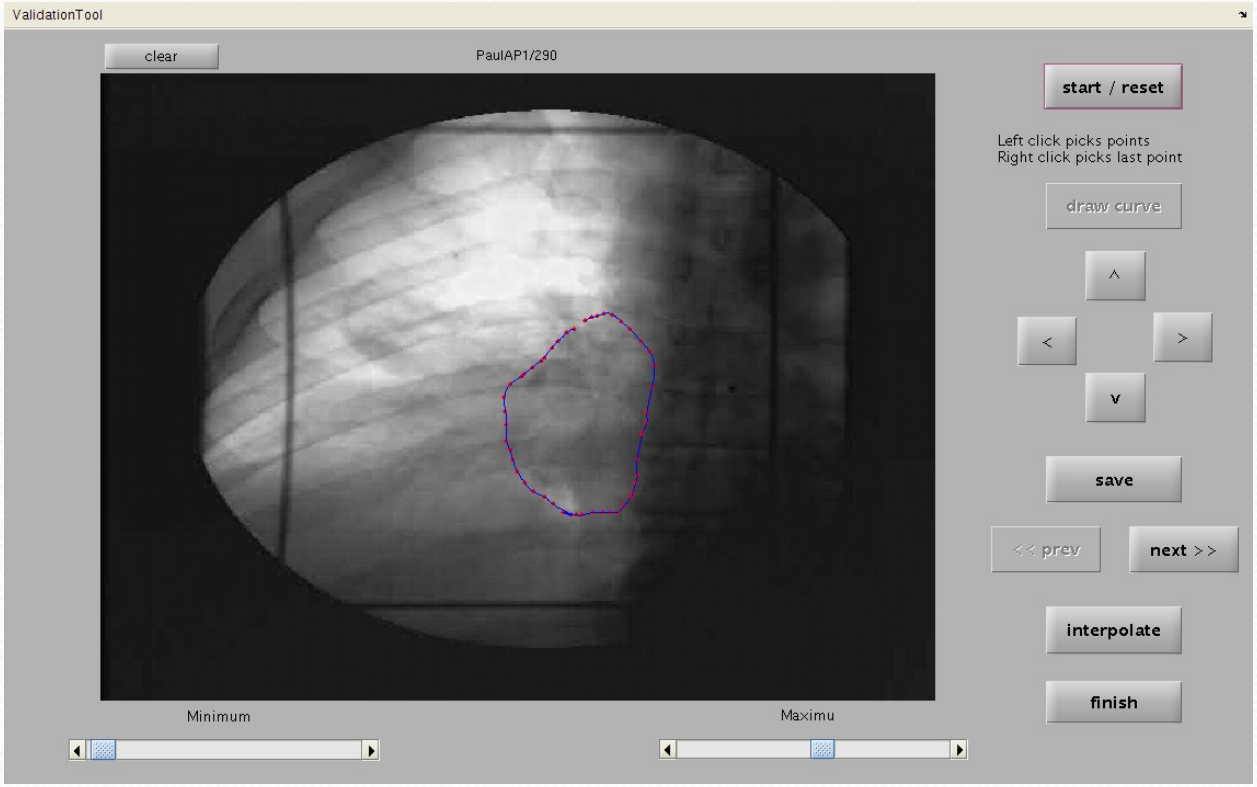
Motion Enhancement



- Motion enhanced image is difference between the image and average of images used for templates
  - Moving structures are emphasized
- Pearson's correlation is used to measure similarity between motion enhanced templates and frame ROIs

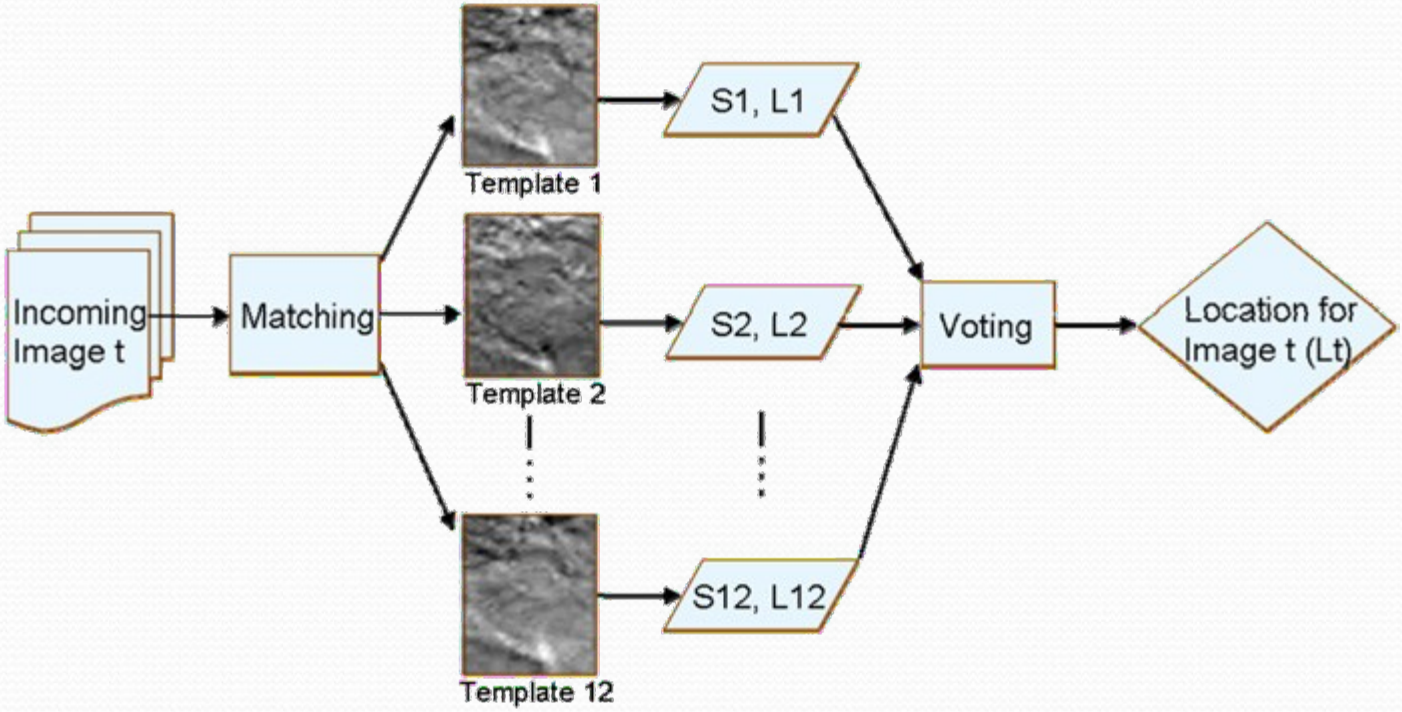
Image: Y. Cui, J. G. Dy, G. C. Sharp, B. Alexander, and S. B. Jiang, "Multiple Template Based Fluoroscopic Tracking of Lung Tumor Mass without Implanted Fiducial Markers," Physics in Medicine and Biology, Vol. 52, pp. 6229-6242, 2007.

Tumor Marking Tool



- Templates are generated with a MATLAB tool
- Would be created by clinician
  - Wide variation in template size based on tumor size and human factors

Multiple Templates

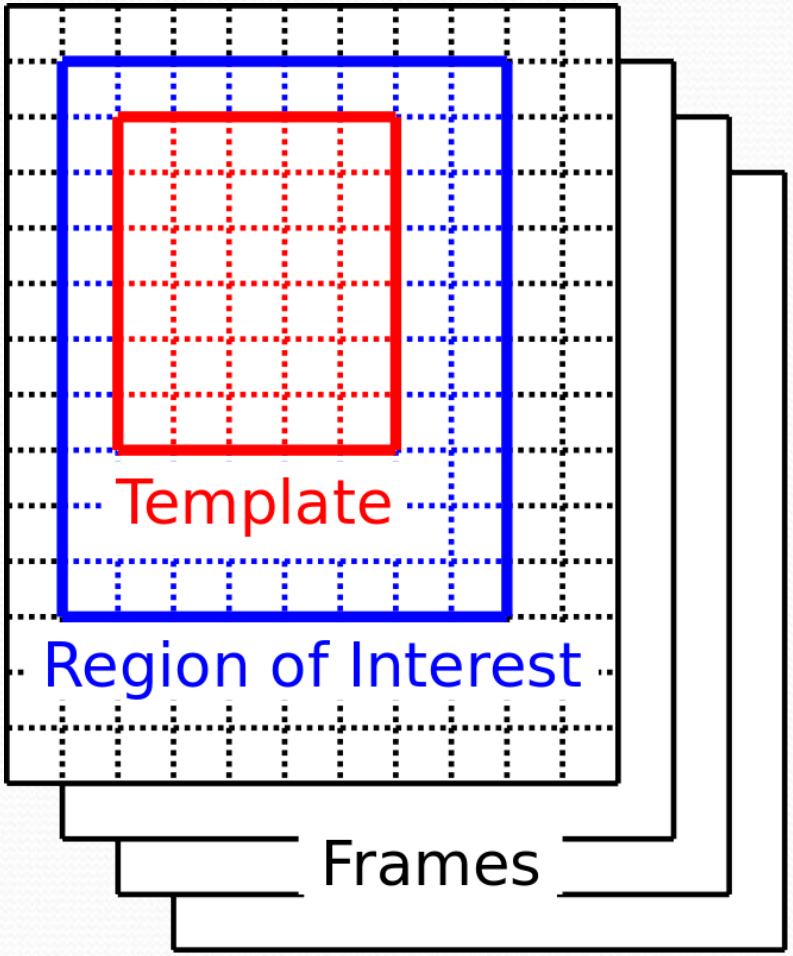


- Image brightness & tumor position varies during breathing
- Periodic: multiple representative templates from different points in the respiration cycle are used to compensate

Image: Y. Cui, J. G. Dy, G. C. Sharp, B. Alexander, and S. B. Jiang, "Multiple Template Based Fluoroscopic Tracking of Lung Tumor Mass without Implanted Fiducial Markers," Physics in Medicine and Biology, Vol. 52, pp. 6229-6242, 2007.

Sliding Window Correlation

- Also allow template location variation for improved tracking
  - All template positions within the region of interest (ROI) are checked
- Each template is applied to all positions within the frame data ROIs for each frame
  - Static datasets of a large number of images
- Interested in accelerating development of algorithms



MATLAB M-Code Implementation

- Pearson's correlation is corr2 from the Image Processing Toolbox
- Four nested for loops generate all the necessary correlations
- Large number of correlations represent 82% of runtime
- Only other significant contributor: image file I/O and conversion: 4.6%

```
for i=1:numFrames
 for j=1:numTemplates
  for k = -lrShift:lrShift
   for m= -udShift:udShift
    curCorr =
          corr2(curT,curF);
   end
  end
 end
end
```

Looping pseudocode without image indexing or similarity score keeping

corr2() Function (1)

$$\mathrm{corr2}(A,B)=\frac{\sum_M\sum_N (A_{MN}-\bar{A})(B_{MN}-\bar{B})}{\sqrt{\left(\sum_M\sum_N (A_{MN}-\bar{A})^2\right)\left(\sum_M\sum_N (B_{MN}-\bar{B})^2\right)}}$$

- All corr2 calls run at once to exploit parallelism on GPU
- Large amount of computation redundancy
  - Matrix averages reused frequently
  - Denominator calculation used repeatedly
  - Relatively few templates applied to many frames

corr2() Function (2)

$$\text{corr2}(A,B) = \frac{\sum_M \sum_N (A_{MN} - \bar{A})(B_{MN} - \bar{B})}{\sqrt{\left(\sum_M \sum_N (A_{MN} - \bar{A})^2\right)\left(\sum_M \sum_N (B_{MN} - \bar{B})^2\right)}}$$

- Possible to implement optimizations in MATLAB
  - Not typically done by MATLAB users
  - Optimized MATLAB parallel corr2() was created to determine correctness of GPU implementation
- Multiple kernels required
  - Hardware constraints
  - Take advantage of redundant computation

corr2() Function (3)

$$\mathrm{corr2}(A,B)=\frac{\sum_M\sum_N (A_{MN}-\bar{A})(B_{MN}-\bar{B})}{\sqrt{\left(\sum_M\sum_N (A_{MN}-\bar{A})^2\right)\left(\sum_M\sum_N (B_{MN}-\bar{B})^2\right)}}$$

- Six separate kernels:
  - Separate average and denominator kernels for frame and template data
  - Numerator
  - Final multiplication
- Grid represents up/down and left/right shifts of templates within a frame's ROI; thread for each frame

corr2() Function (3)

$$\mathrm{corr2}(A,B)=\frac{\sum_M\sum_N (A_{MN}-\bar{A})(B_{MN}-\bar{B})}{\sqrt{\left(\sum_M\sum_N (A_{MN}-\bar{A})^2\right)\left(\sum_M\sum_N (B_{MN}-\bar{B})^2\right)}}$$

- Six separate kernels:
  - Separate average and denominator kernels for frame and template data
  - Numerator
  - Final multiplication
- Grid represents up/down and left/right shifts of templates within a frame's ROI; thread for each frame

corr2() Function (3)

$$\mathrm{corr2}(A,B)=\frac{\sum_M\sum_N (A_{MN}-\bar{A})(B_{MN}-\bar{B})}{\sqrt{\left(\sum_M\sum_N (A_{MN}-\bar{A})^2\right)\left(\sum_M\sum_N (B_{MN}-\bar{B})^2\right)}}$$

- Six separate kernels:
  - Separate average and denominator kernels for frame and template data
  - Numerator
  - Final multiplication
- Grid represents up/down and left/right shifts of templates within a frame's ROI; thread for each frame

corr2() Function (3)

$$\mathrm{corr2}(A,B)=\frac{\sum_M\sum_N (A_{MN}-\bar{A})(B_{MN}-\bar{B})}{\sqrt{\left(\sum_M\sum_N (A_{MN}-\bar{A})^2\right)\left(\sum_M\sum_N (B_{MN}-\bar{B})^2\right)}}$$

- Six separate kernels:
  - Separate average and denominator kernels for frame and template data
  - Numerator
  - Final multiplication
- Grid represents up/down and left/right shifts of templates within a frame's ROI; thread for each frame

corr2() Function (3)

$$\mathrm{corr2}(A,B)=\frac{\sum_M\sum_N (A_{MN}-\bar{A})(B_{MN}-\bar{B})}{\sqrt{\left(\sum_M\sum_N (A_{MN}-\bar{A})^2\right)\left(\sum_M\sum_N (B_{MN}-\bar{B})^2\right)}}$$

- Six separate kernels:
  - Separate average and denominator kernels for frame and template data
  - Numerator
  - Final multiplication
- Grid represents up/down and left/right shifts of templates within a frame's ROI; thread for each frame

corr2() Function (3)

$$\mathrm{corr2}(A,B)=\frac{\sum_M\sum_N (A_{MN}-\bar{A})(B_{MN}-\bar{B})}{\sqrt{\left(\sum_M\sum_N (A_{MN}-\bar{A})^2\right)\left(\sum_M\sum_N (B_{MN}-\bar{B})^2\right)}}$$

- Six separate kernels:
  - Separate average and denominator kernels for frame and template data
  - Numerator
  - Final multiplication
- Grid represents up/down and left/right shifts of templates within a frame's ROI; thread for each frame

corr2() Function (3)

$$\mathrm{corr2}(A,B)=\frac{\sum_M\sum_N (A_{MN}-\bar{A})(B_{MN}-\bar{B})}{\sqrt{\left(\sum_M\sum_N (A_{MN}-\bar{A})^2\right)\left(\sum_M\sum_N (B_{MN}-\bar{B})^2\right)}}$$

- Six separate kernels:
  - Separate average and denominator kernels for frame and template data
  - Numerator
  - Final multiplication
- Grid represents up/down and left/right shifts of templates within a frame's ROI; thread for each frame

Reference Dataset

Patient	Frames	Templates	Template Dimensions	Shift V/H	corr2() Calls
1	442	12	53x54	18/9	3532464
2	348	13	23x21	11/5	1144572
3	259	10	76x45	9/4	442890
4	290	11	116x175	9/3	424270
5	293	12	78x109	11/6	979524
6	210	14	107x159	9/2	279300

- Reference data set includes six patients with manually specified tumor location for each frame
  - All the parameters vary
- Existing application parameters not based on powers of two

CUDA corr2() Implementation

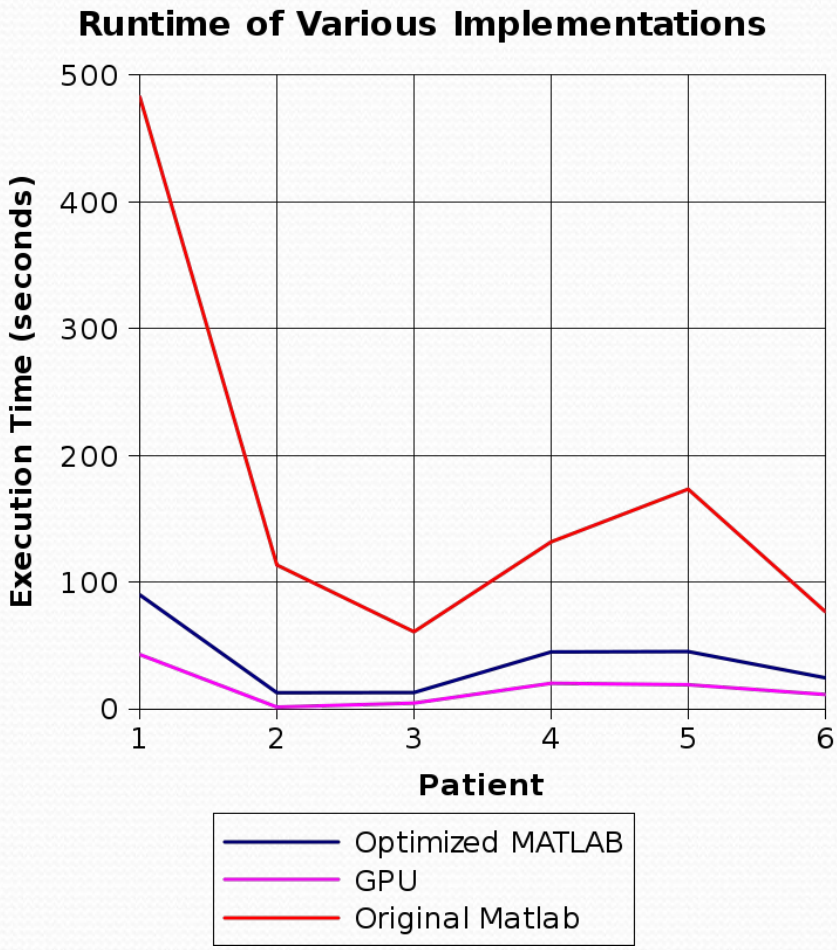
- Frames/templates mapped to threads in a block
  - Frames: not multiple of 64, but within appropriate range
  - Templates: not efficient
- Template locations in the frame data ROI are mapped to block grid
  - 95 to 703 blocks for frame kernels
  - 1 block for template kernels: not efficient
- Numerator and Final Multiplication combine frame and template statistics
  - 4-dimensional data
  - Kernels combines one frame set with one template
    - Invocation for each template required

Experimental Setup

- Used MOCA to examine multiple GPU memory mappings of the template matching kernels
  - Global memory only
  - Frame data in texture memory
  - Template data in texture memory
- Benchmarking Platform
  - Ubuntu 9.04 64-bit
  - GeForce 8800 GTX w/ CUDA 2.3
  - Intel Core 2 Duo E8400 (3 GHz, 6 MB L2)
  - GCC 4.2.4

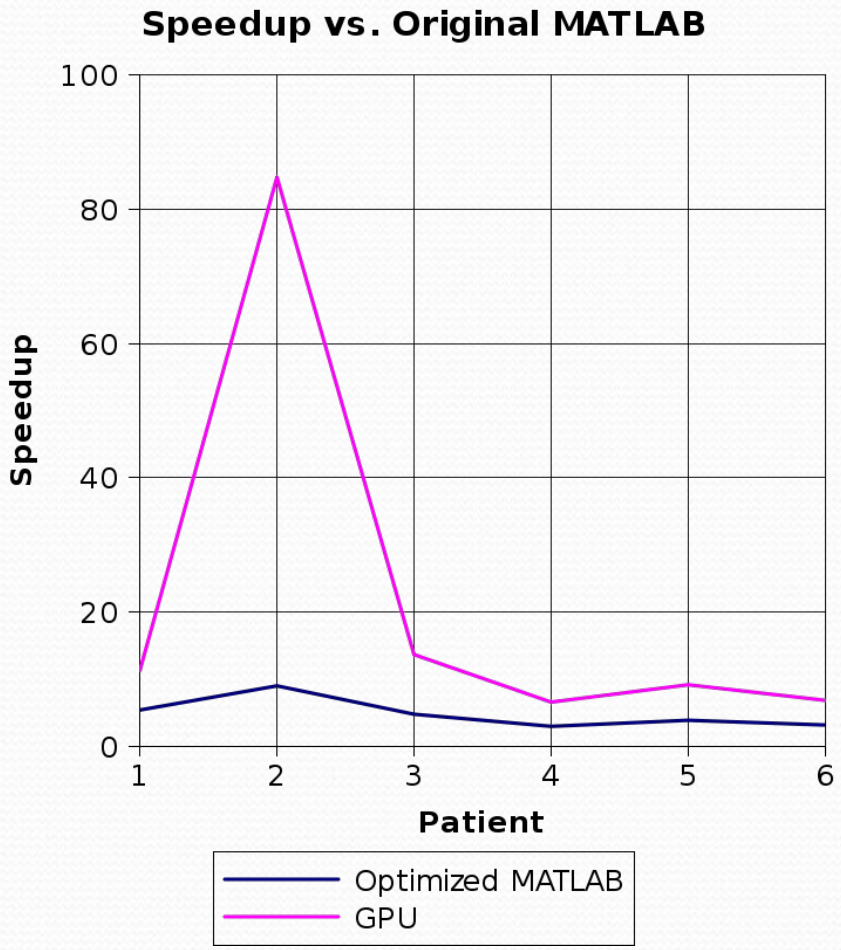
Global Memory Only

- Compared original MATLAB, optimized MATLAB and global only GPU version
- Optimized MATLAB implemented similarly to GPU
  - Six steps like GPU kernels
  - Loops flattened
  - Reuses averages and denominators
- GPU implementations vary in performance
  - Includes conversion from double to single precision
  - End-to-end timing, including data transfers



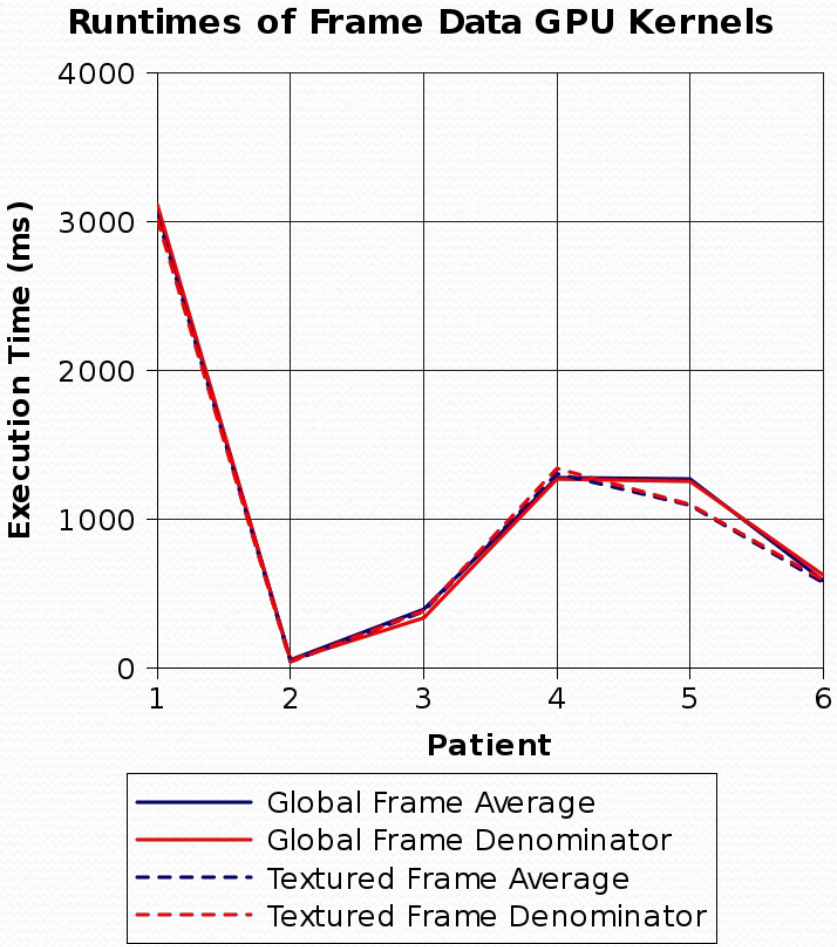
Global Memory Only

- Patient 2 template and frame ROI sizes are smallest, results in best memory read performance
- Non-coalesced data reads are the sticking point
  - Each thread works on a different frame
  - Stride between frames is large
  - Need to address: data reorganization or thread/grid mapping



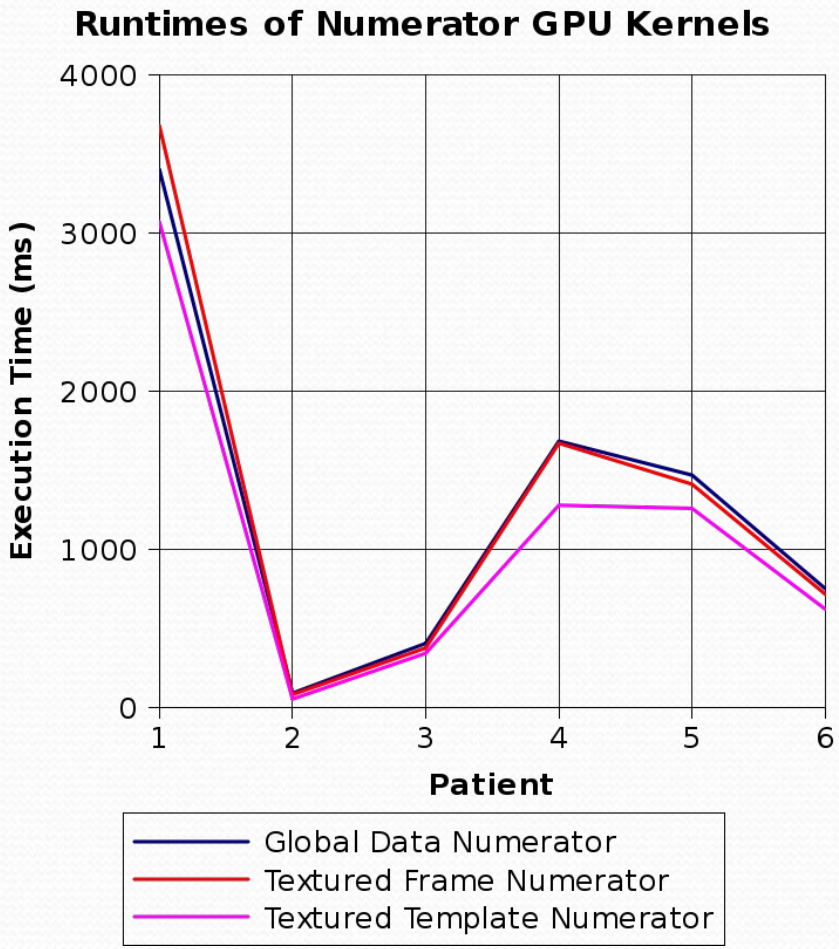
Textured Frame Memory

- There are many redundant frame data reads for any kernel touching the frame data
  - Moved frame data to texture memory
  - Minor performance differences
  - Locality not enough for textures to provide a benefit

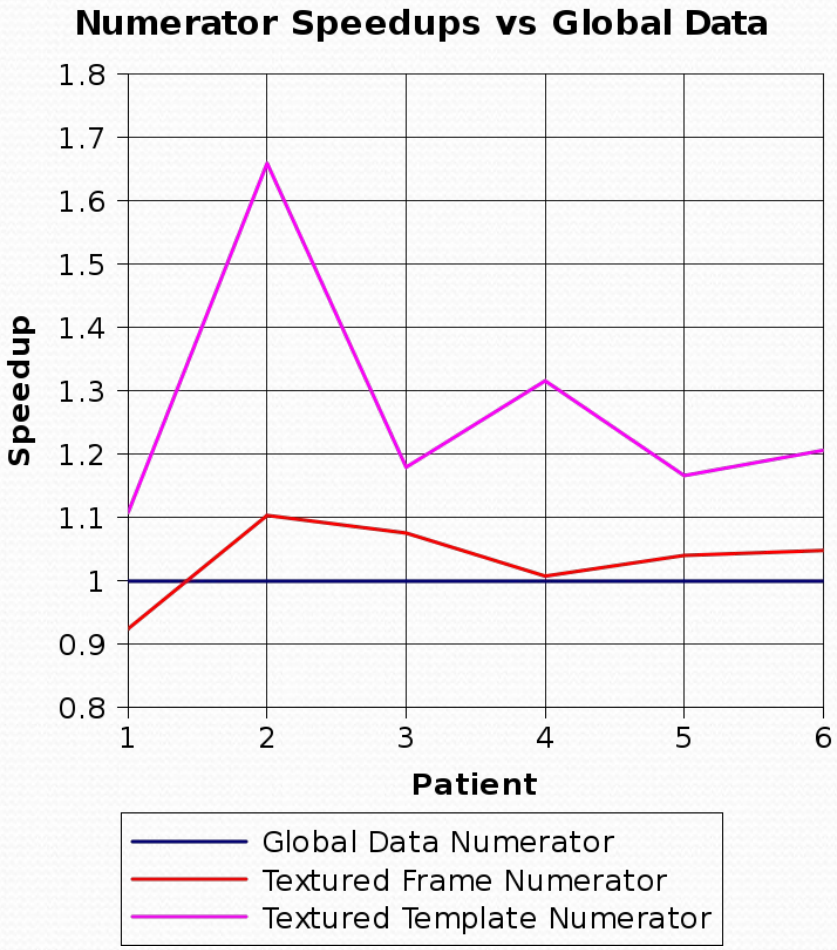


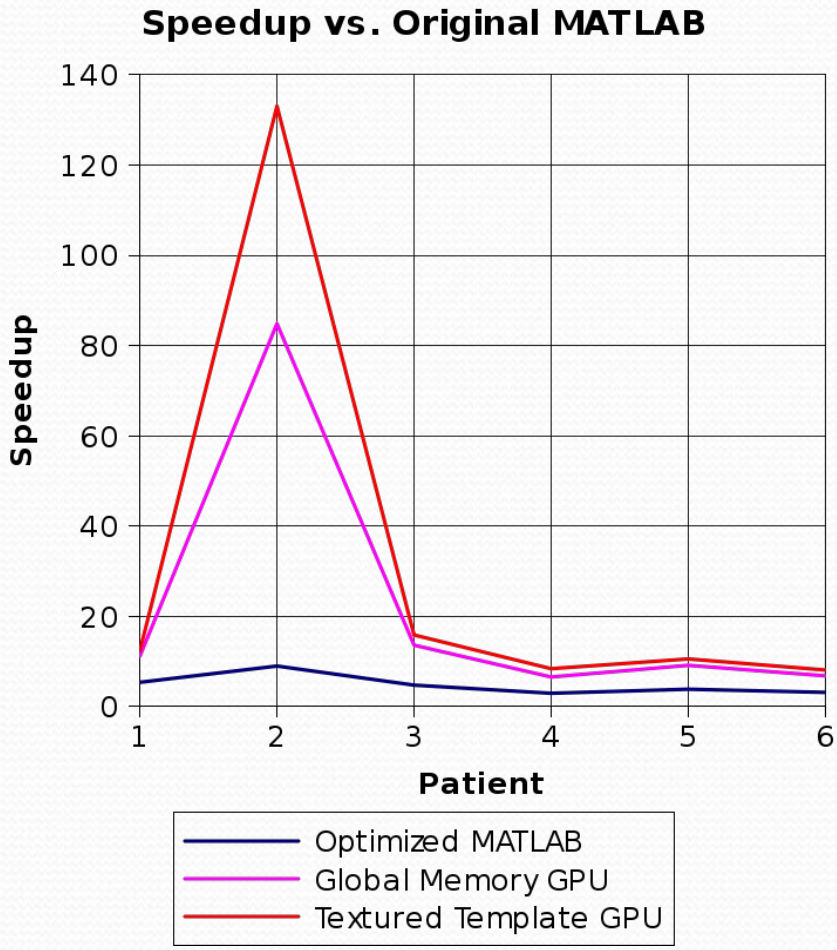
Textured Template Memory

- Numerator threads each read the same template
  - Global memory not cached
  - Currently too large for shared memory
- Putting template data in texture memory shows some improvement
  - Enough locality for textures to cache some reads
- MOCA makes it easier to explore implementations



Textured Template Memory





Conclusions

	Global Memory	Textured Frames	Textured Templates
Best Speedup	85	92	133
Average Speedup	22	23	31

- Memory placement is an important performance factor
  - Even with sub-optimal kernels
  - Important when applying a library of kernels to multiple problems
- MOCA aids prototyping and implementation space exploration

Future Work

- Template Matching
  - Add data reorganization for more coalesced memory accesses
    - Another host-side operation that may be common
- MOCA Improvements
  - Extend to support asynchronous operations
  - Increase automation of implementation space exploration
  - OpenCL back end
- Other applications: eigen template algorithm or new application
- Focus on the optimal dimensions for parameterization and representation within a library

Thank You

Nicholas Moore, Miriam Leeser
{nmoore,mel}@coe.neu.edu