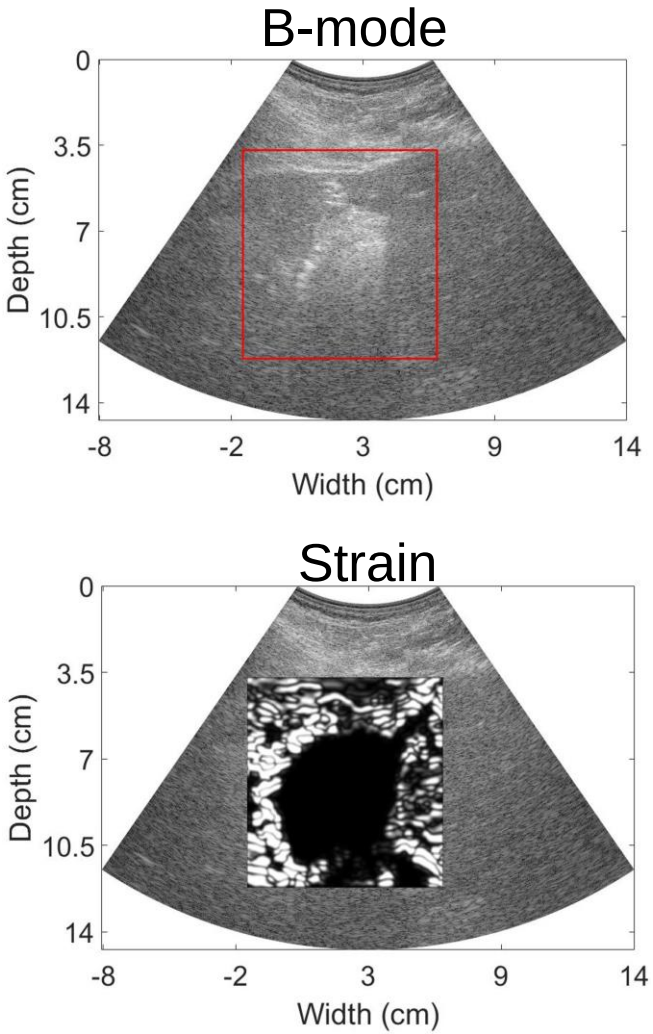
Accelerating Ultrasound Elasticity Imaging with CUDA-MATLAB Approach

Rashid Al Mukaddim, Michael Turney, Robert Pohlman
Advised by Dr. Tomy Varghese
Department of Medical Physics, Department of Electrical and Computer Engineering

12-21-2017

Problem Statement

- Elastography is a process of using Ultrasound to measure the elasticity of a material from an applied force.
- The elasticity can be used for imaging purposes and is an important aspect for clinicians to deduce the tissue properties for assessing current treatments or ailments.
- Current methods for calculating displacements are very computationally expensive and time consuming.



We aim to create a GPU implementation of ultrasound elasticity imaging to speed up the strain estimation process.

Displacement and Strain Estimation

Pre and Post Tissue Compression Image Acquisition

Normalized Cross-Correlation (NCC) between Windowed Pre and Post Image

Integer Displacement through peak finding of NCC matrix

Subsample Displacement through 2D Surface Fit

Strain through Least Square Fit of Displacement

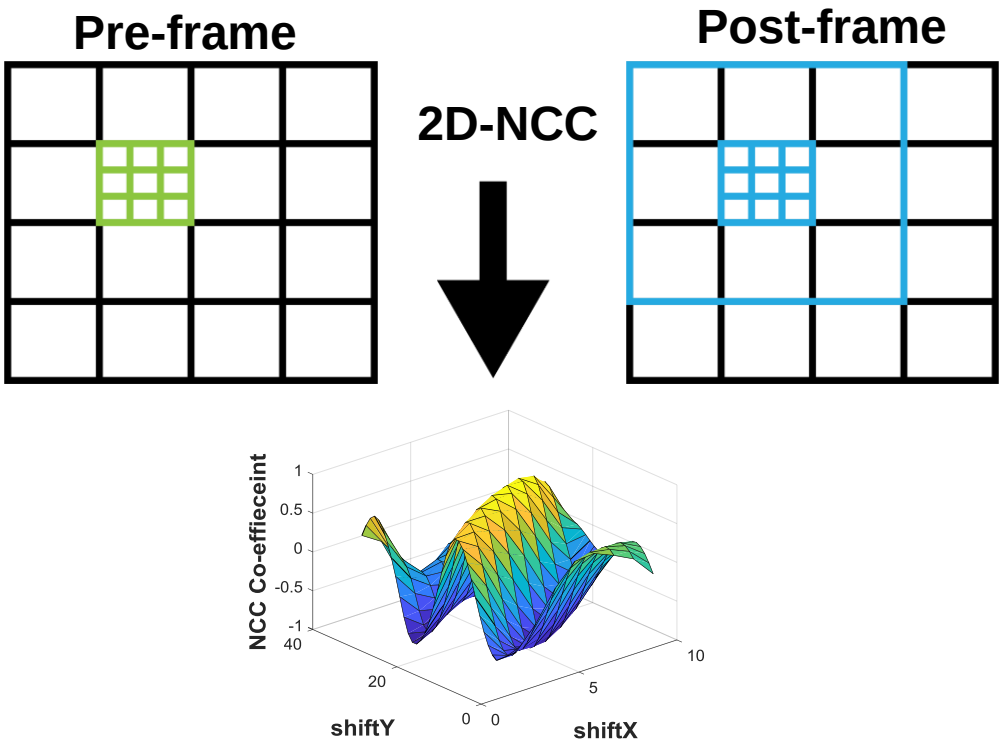
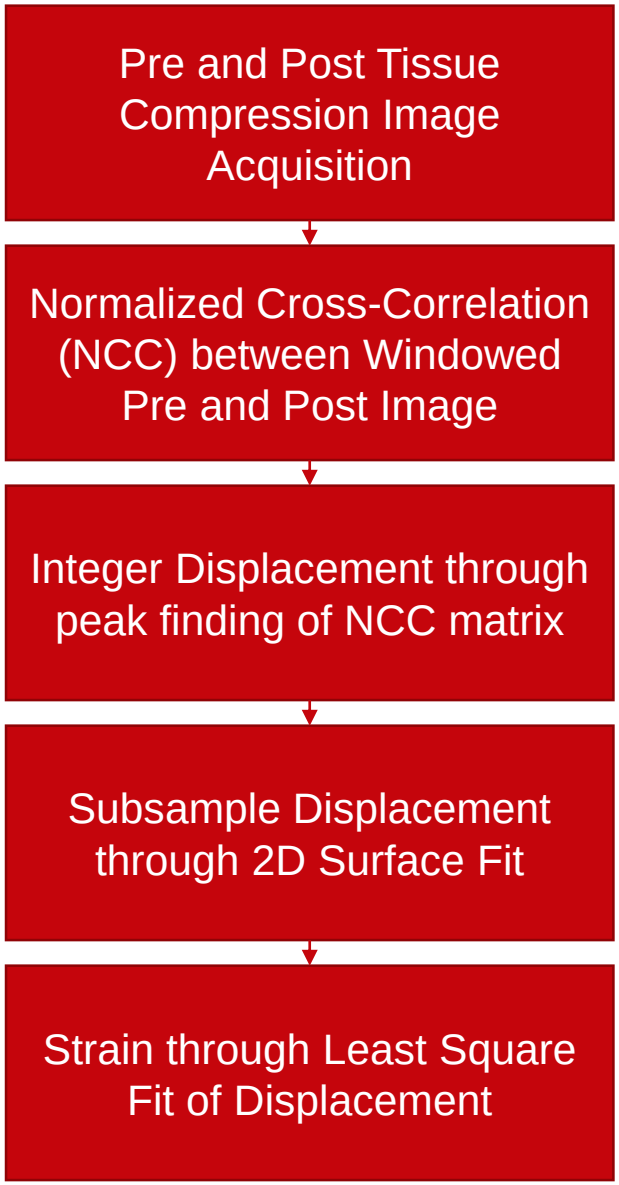


Figure illustrates NCC calculation for estimating one sample point of displacement image. Green window shows a kernel and blue window shows the search region. Process is repeated for estimating all other samples of displacement image

Displacement and Strain Estimation in GPU



Computationally Intensive Stages
→ Translated to GPU

- **`normXcorr_GPU`** Kernel

  Kernel for calculating localized normalized cross-correlation

- **`MaxElement`** Kernel

  Kernel for integer displacement estimation through peak finding

- **`Subsample`** Kernel

  Kernel for subsample displacement through 2D surface fit

Kernel Execution Models

- **`normXcorr_GPU`** Kernel
  - CUDA blocks = Total displacement points (pre determined based on user defined parameters)
  - CUDA threads/block = Number of shifts in the user defined search region
- **`MaxElement`** Kernel
  - CUDA blocks = Total displacement points (pre determined based on user defined parameters)
  - CUDA threads/block = Number of shifts in the user defined search region
- **`Subsample`** Kernel
  - CUDA Blocks = Total displacement points + 511 / 512
  - CUDA threads/block = 512

Optimizations

1. Relieving Thread Divergence

   Input images were zero-padded to reduce thread divergence. Speedup: ~2.7x

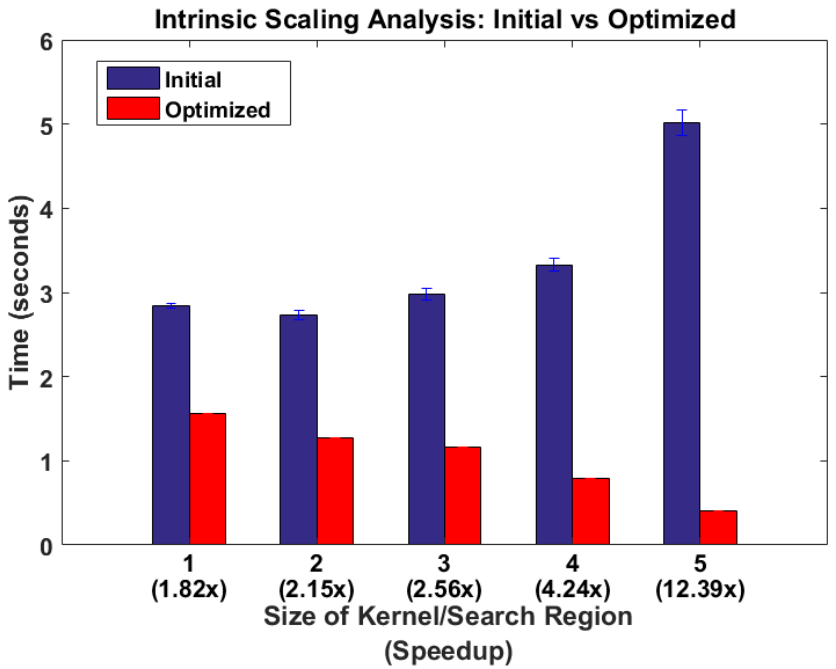
2. Shared Memory

   Since the pre input image is used by multiple threads, we took advantage of shared memory. Speedup: ~1.5x.

3. GPU Peak Finding and Subsample Displacement

   Implemented on the GPU for speedup and to reduce data transfer (no need to send CC matrices back to the host/MATLAB). Speedup: ~1.2x

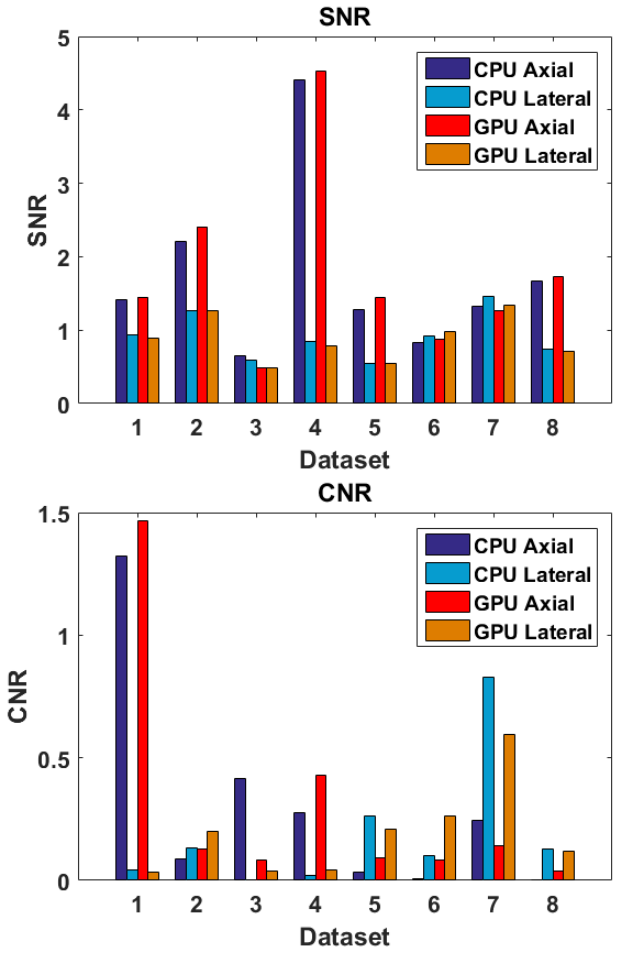
Bar Number	Kernel Y	Kernel X
1	161	9
2	129	9
3	97	9
4	65	9
5	33	9



The execution time of the initial GPU implementation was approximately 5 seconds using the largest kernel size. After optimizations above were completed, the execution time dropped to 1.5 seconds.

Results

Accuracy



Speedups

Kernel Size [x, y]	Search Size [x, y]	Average Speedup
[7, 171]	[5, 41]	8.53x
[9, 161]	[17, 49]	10.41x
[9, 129]	[17, 39]	10.72x
[9, 97]	[17, 31]	10.18x
[9, 65]	[17, 21]	12.04x
[9, 33]	[17, 11]	20.19x

We were successful in implementing the Multilevel Method on the GPU and comparing the results to the previous CPU implementation. The GPU showed similar results as the CPU implementation with speedups of 8-20x.