Parallel Computing Class Homework Assignment Proposal

Mike Roberts

Goals

Reasonable coverage of important parallel primitives and interesting image processing applications

Emphasis on empirical performance evaluation

Parallel primitives covered in this proposal

Map

Scatter/Gather

Stencil

Reduce

Scan

Segmented Scan

Compact

Sort

Image processing applications covered in this proposal

Color Conversion

Gaussian Blur

Bilateral Filtering

Red Eye Removal

Tonemapping

Poisson Blending

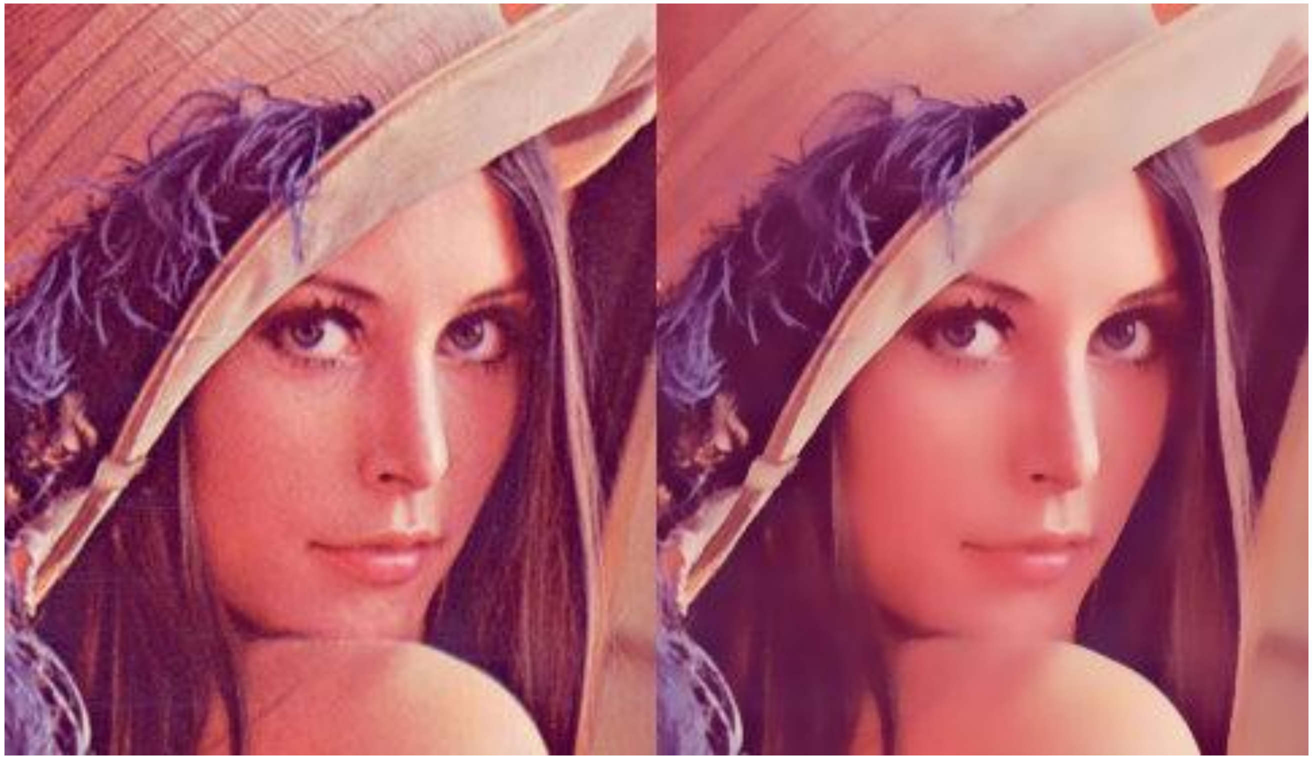
Graph Cuts

HW1: Greyscale Conversion

HW1: Greyscale Conversion

- Hello CUDA
- Convert color image to greyscale
- Parallel primitives covered:
  - Map
- Performance evaluation questions:
  - How does performance vary with block size?
  - How does performance vary with image size?
- Estimated difficulty: Easy

HW2: Gaussian Blur and Smart Blur using the Bilateral Filter



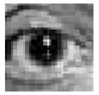
HW2: Gaussian Blur and Smart Blur using the Bilateral Filter

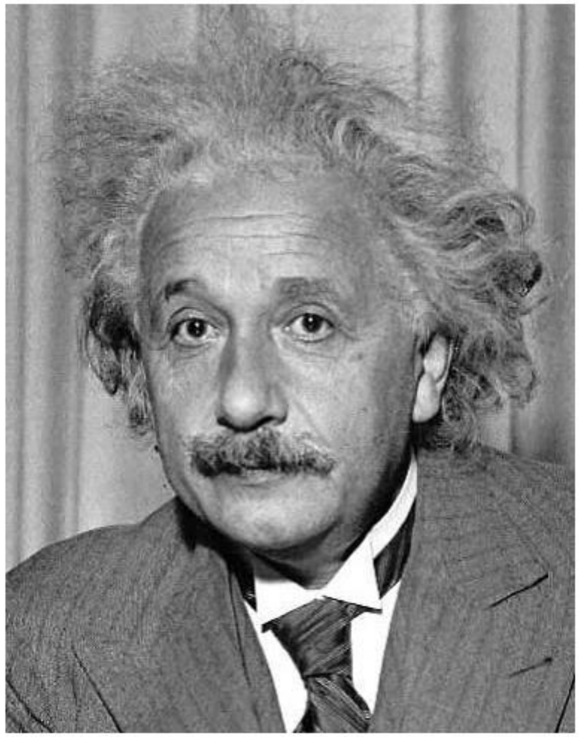
- Typical Guassian Blur using convolution
- Bilateral filter is just like Gaussian blur but with data-dependent kernel weights.
- See Joe Stam’s GTC 2009 talk, “Convolution Soup: A Case Study in CUDA Optimization” for possible optimizations
- Parallel primitives covered:
  - Map, Scatter/Gather, Stencil
- Performance evaluation questions:
  - What memory space to put blur kernel?
  - How does performance vary with kernel size?
  - Can we put the image to be blurred in texture memory?
  - How does performance vary with image size?
- Estimated difficulty: Moderate

HW2: Gaussian Blur and Smart Blur using the Bilateral Filter

- Decided against anisotropic diffusion for implementing smart blur
  - Although anisotropic diffusion can produce comparable results to bilateral filtering, it is less common in the literature
  - Anisotropic diffusion also has more parameters to tune and requires iteratively solving a partial differential equation over the image domain
  - In contrast, bilateral filtering has fewer parameters to tune and requires only one pass over the image.
  - Therefore, I think it is more useful to expose students to bilateral filtering

HW3: Automatic Red Eye Removal Using Template Matching (optional – we might want to replace this one)

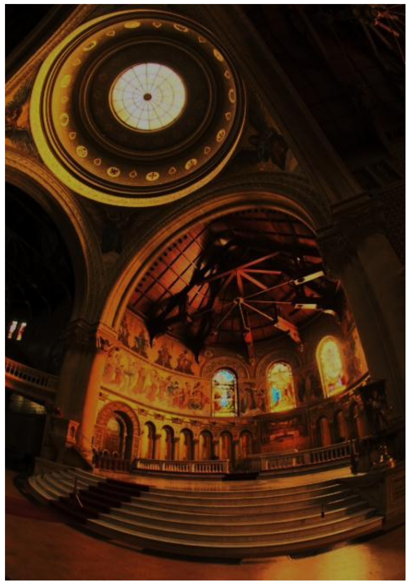
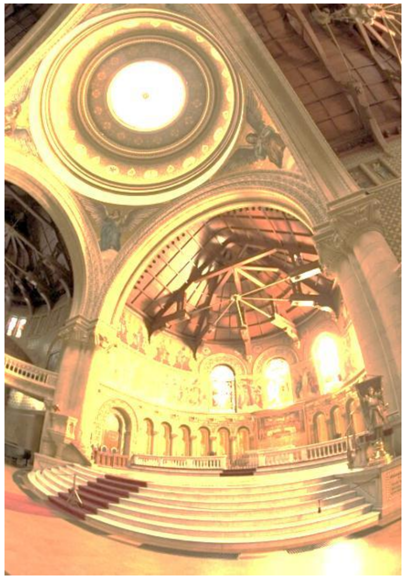
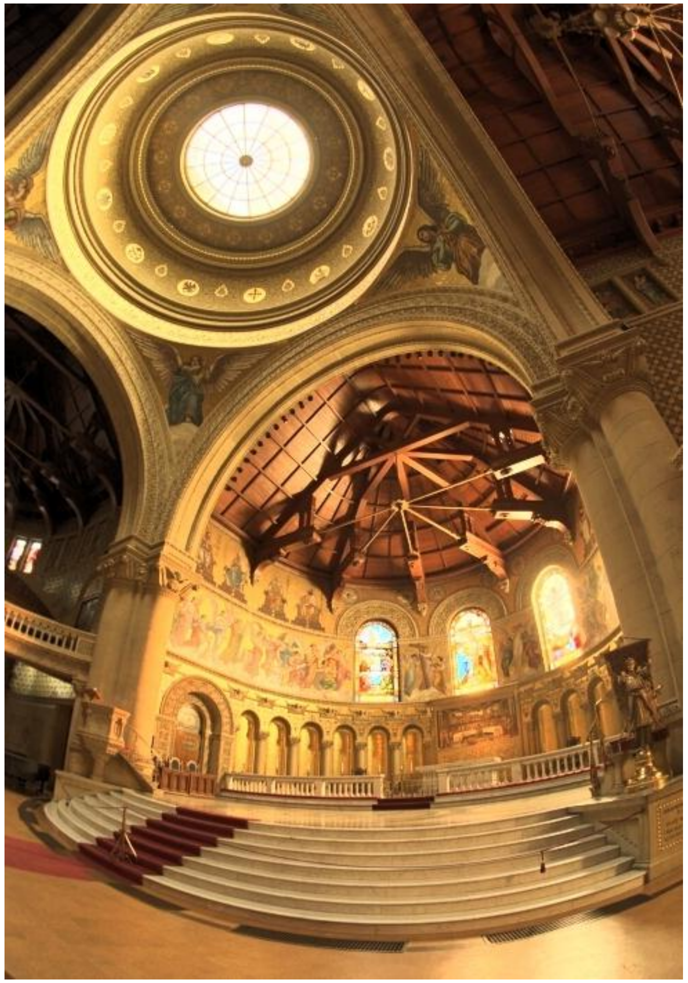
- Goal: find  in image



HW3: Automatic Red Eye Removal Using Template Matching (optional – we might want to replace this one)

- Convolve *original image* with an *eye template* to produce a *response image*
- Eye pixels will be the brightest pixels in the *response image*
- Sort pixels in the *response image* according to brightness to get all the eye pixels in the *original image*
- Reduce the redness of eye pixels
- See http://www.cs.illinois.edu/class/sp11/cs543/ (Lecture 7) for details
- Use experience from previous assignment, convolution should be easy, students can focus on sorting
- Best image processing application I could think of that relies directly on sorting
- Parallel primitives covered:
  - Map, Scatter/Gather, Stencil, Sort
- Performance evaluation questions:
  - Similar to previous assignment, but also how can we implement sort efficiently?
  - Lots of options for implementing an efficient sort
- Estimated difficulty: Moderate

HW4: High Dynamic Range Tone Mapping using Ward's Histogram Adjustment Operator

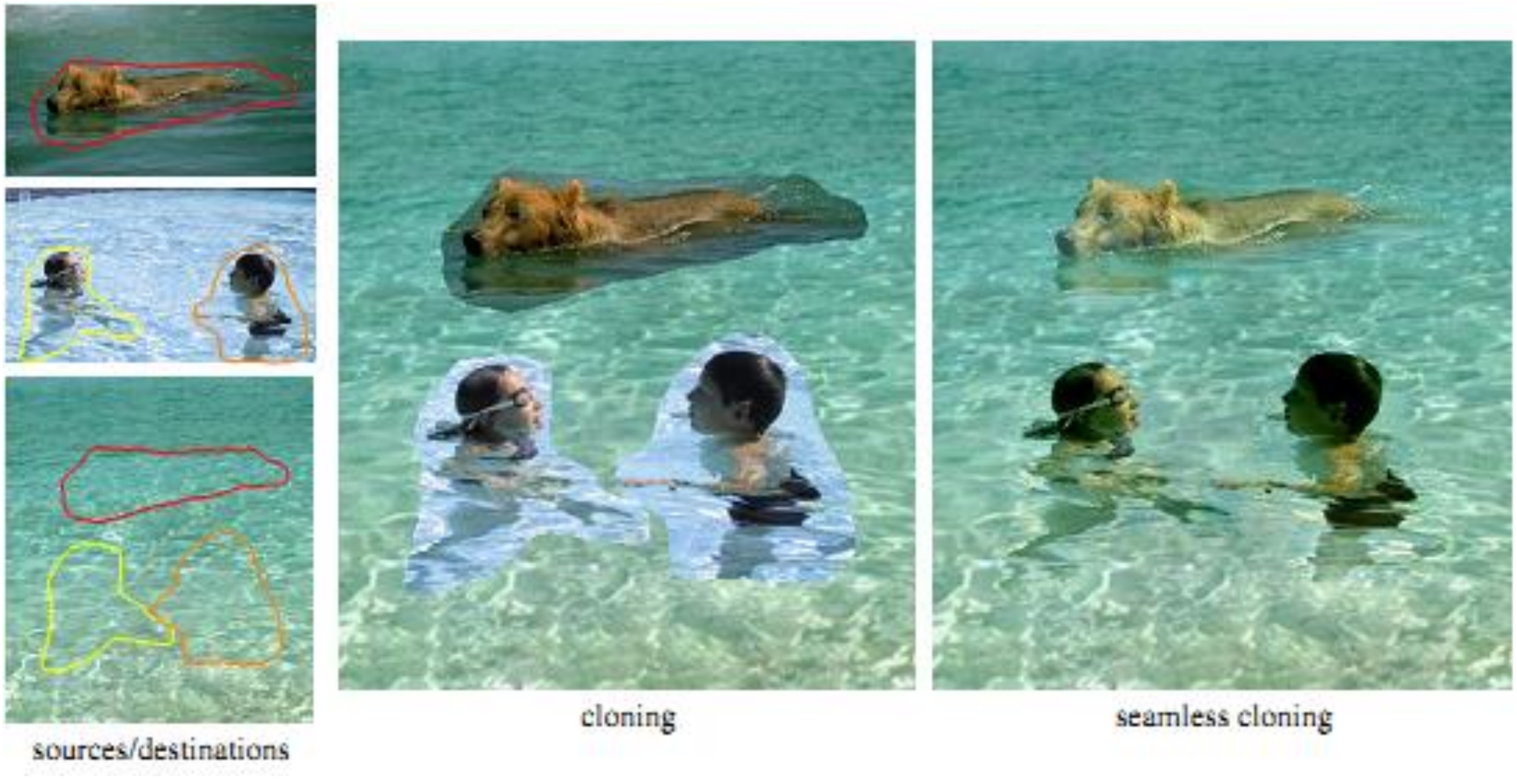
HW4: High Dynamic Range Tone Mapping using Ward's Histogram Adjustment Operator

- Compute global minimum and maximum luminance for image (reduce)
- Compute luminance histogram for image (scatter/gather using atomic operations)
- Adjust histogram using Ward's algorithm (iterated map and reduce)
- Compute the cumulative distribution function for the histogram by prefix-summing the histogram (scan)
- Use the cumulative distribution function as the luminance remapping curve (map)
- Parallel primitives covered:
  - Map, Scatter/Gather, Reduce, Scan
- See original paper and followup GPU paper for details:
  - http://radsite.lbl.gov/radiance/papers/lbnl39882/tonemap.pdf
  - http://developer.amd.com/gpu_assets/GPUHistogramGeneration_I3D07.pdf
- Similar homework assignment in MIT Computational Photography course
- Decided against Reinhart-style histogram equalization (SIGGRAPH 2002) because it seemed more complicated and less amenable to parallelization
- Estimated Difficulty: Moderate

HW5: Implementing Different Histogram Algorithms and Evaluating Their Performance

- Implement improved histogram algorithms for the previous assignment and evaluate the relative performance of each:
  - Sort the input, bin the sorted input, perform segmented prefix sum to accumulate the number of values in each bin, compact to compute histogram
  - Use shared memory where possible, use a hierarchical decomposition strategy where number of bins exceeds size of shared memory
- See the following papers for details:
  - http://developer.amd.com/gpu_assets/GPUHistogramGeneration_I3D07.pdf
  - http://developer.download.nvidia.com/compute/cuda/1.1-Beta/x86_website/projects/histogram64/doc/histogram.pdf
  - http://parse.ele.tue.nl/system/attachments/10/original/GPGPU_High%20Performance%20Predictable%20Histogramming%20on%20GPUs.pdf
  - http://innovativeparallel.org/Presentations/INPAR%20Modestly%20Faster%20GPU%20Histograms%20PDF.PDF
- Parallel primitives covered:
  - Map, Scatter/Gather, Segmented Scan, Compact, Sort
- If students are required to implement sorting in this assignment, then we might want to replace HW3 with HW8
- Quite open-ended. We could make this into a contest to see who can get the best performance.
- Estimated Difficulty: Hard

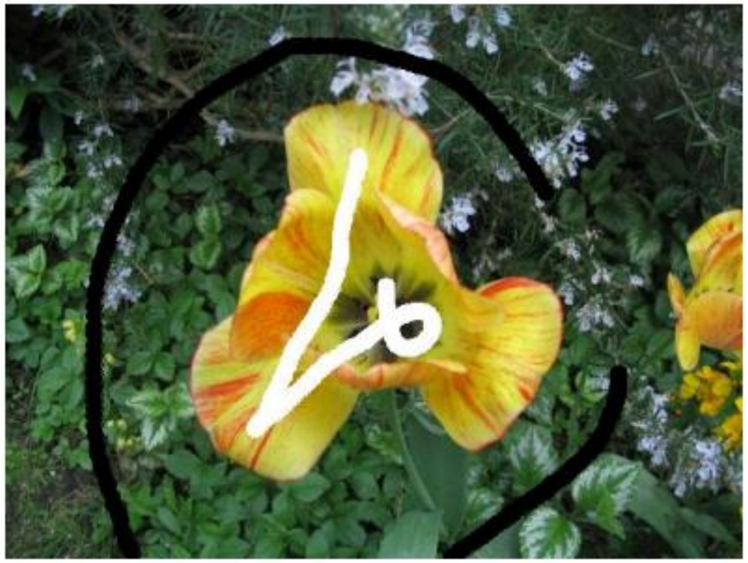
HW6: Seamless Image Compositing using Poisson Blending

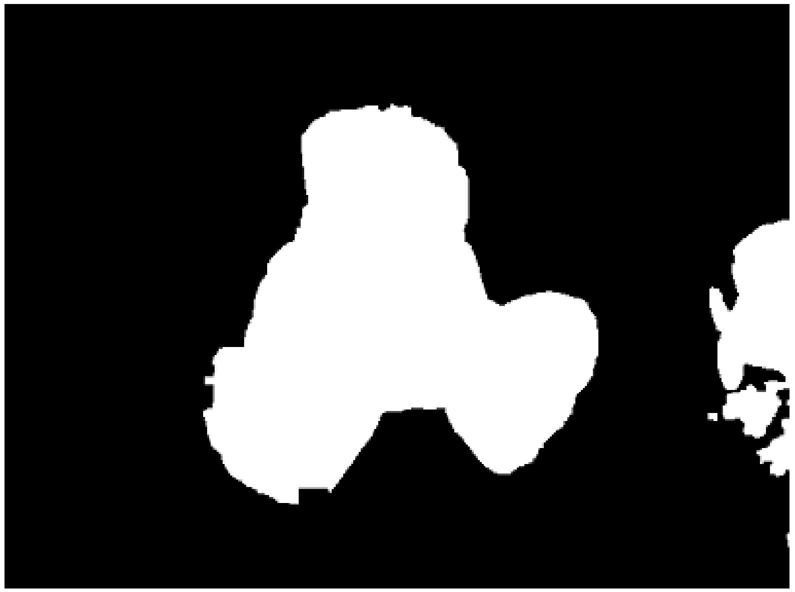


HW6: Seamless Image Compositing using Poisson Blending

- Given a *source image* and a *target image*, compute an *output image*
- The *output image* must respect a system of linear equations that relates the *gradients* of the *source image*, the *intensities* of the *target image*, and the *intensities* of the *output* image
- Construct a system of linear equations and solve using the Jacobi Method
- See the following papers for details on Poisson Blending:
  - http://www.cs.brown.edu/courses/csci1950-g/asgn/proj2/resources/PoissonImageEditing.pdf
  - http://www.cs.jhu.edu/~misha/Fall07/Notes/Perez.pdf
  - http://www.cs.jhu.edu/~misha/ReadingSeminar/Papers/Jeschke09b.pdf
- See the following paper for details on solving the Poisson equation using the Jacobi Method:
  - http://http.developer.nvidia.com/GPUGems/gpugems_ch38.html
- Parallel primitives covered:
  - Map, Scatter/Gather, Stencil, Reduce
- Performance evaluation questions:
  - Can we use shared memory to solve the Poisson equation more efficiently?
  - Are there other numerical methods that we can use to solve the Poisson equation more efficiently?
  - What are the convergence properties of these methods?
- Similar homework assignment in Brown, UIUC, and MIT Computational Photography courses, CMU Advanced Parallel Graphics course
- Estimated difficulty: Hard

HW7: Selecting Irregular Image Regions using Graph Cuts


Input


Result

HW7: Selecting Irregular Image Regions using Graph Cuts

- Set up a weighted undirected graph of nodes (pixels) and edges (connecting neighboring pixels)
- Set edge weights according to gradient magnitudes in the image
- Execute the push-relabel algorithm on each node in parallel until all nodes converged
  - Naïve implementation requires global atomic operations
- See Timo Stich's GTC 2009 talk, "Graph Cuts with CUDA" for more details
- Parallel primitives covered:
  - Map, Scatter/Gather, Stencil, Reduce
- Similar homework assignment in UIUC Computer Vision course, Brown and MIT Computational Photography courses, CMU Advanced Parallel Graphics course
- Estimated difficulty: Hard

HW8: Improving the Performance of Push-Relabel (optional – we need to remove HW3 and replace it with this)

- Use shared memory, change degree of parallelism to avoid global atomic operations
- Keep list of active image tiles using stream compaction
- See the following papers for details:
  - http://cvit.iiit.ac.in/papers/rtGCuts_2008.pdf
  - http://people.seas.harvard.edu/~mroberts/data/a_work-efficient_gpu_algorithm_for_level_set_segmentation_hpg_paper.pdf
- Parallel primitives covered:
  - Map, Scatter/Gather, Stencil, Reduce, Scan, Compact
- Quite open-ended. We could make this into a contest to see who can get the best performance
- Estimated difficulty: Hard

Summary

Topics	HW1: Conversion from Color to Greyscale	HW2: Gaussian Blur and Smart Blur using the Bilateral Filter	HW3: Automatic Red Eye Removal using Template Matching	HW4: High Dynamic Range Tonemapping using Ward's Histogram Adjustment Algorithm	HW5: Implementing Different Histogram Algorithms and Evaluating their Performance	HW6: Seamless Image Compositing using Poisson Blending	HW7: Selecting Irregular Image Regions using Graph Cuts	HW8: Improving the Performance of Push-Relabel
Primitives								
Map	X	X	X	X		X	X	X
Scatter/Gather		X	X	X	X	X	X	X
Stencil		X	X			X	X	X
Reduce				X		X	X	X
Scan				X	X			X
Segmented Scan					X			
Compact					X			X
Sort			X		X			

Notes

- Decided against "swishy picture" effect driven by a Stable Fluids simulation
  - This kind of effect is certainly important, but I feel that the other homework assignments represent image processing operations of broader interest
  - The most interesting part of most fluid simulations (from parallel programming perspective) is solving the Poisson equation to enforce incompressibility, thereby enforcing global constraints on the fluid
  - We are covering this pattern in the Poisson Blending assignment

Notes

- Decided against fisheye warp effect
  - This kind of warping is certainly important, but I feel that the other homework assignments represent image processing operations of broader interest
- Decided against seam carving
  - In some sense, the graph cuts assignment is a generalization of seam carving
  - It seems challenging to expose enough parallelism in the seam carving dynamic programming algorithm
    - If carving vertically, can only process one row at a time
    - If carving horizontally, can only process one column at a time
    - Only 1024 threads of independent work to do for a 1024 X 1024 image