Where's a Pattern?

CUDA C/C++ Pattern Recognition

Kenny Ballou, Abe Oros, Jesse Riggs,
Sasa Rkman, Kristin Van Andel

December 15, 2012

Introduction

What is CUDA C/C++?

Introduction

Project Goals

- Utilize CUDA C/C++ to identify a pattern in an image
  1. Identify a simple, grey-scale box in an image, "white-out" non-matching pixels
  2. Identify a simple, color-scale box in an image, "white-out" non-matching pixels
  3. Identify a simple, grey-scale box in an image, outline border of box
  4. Identify a 16 x 16 pixel pattern in an image (grey-scale) and "white-out"
  5. Identify a 16 x 16 pixel pattern in an image (grey-scale) and outline
  6. Identify "Waldo" in a "Where's Waldo?" image

Approach

- Start Simple
- Add Complexity
- Add *More* Complexity

Methodology

- Implement a way to Read and Write Images (Pixel Maps)
- Implement a CUDA C/C++ program (for each Milestone)
- Implement C versions for performance comparisons

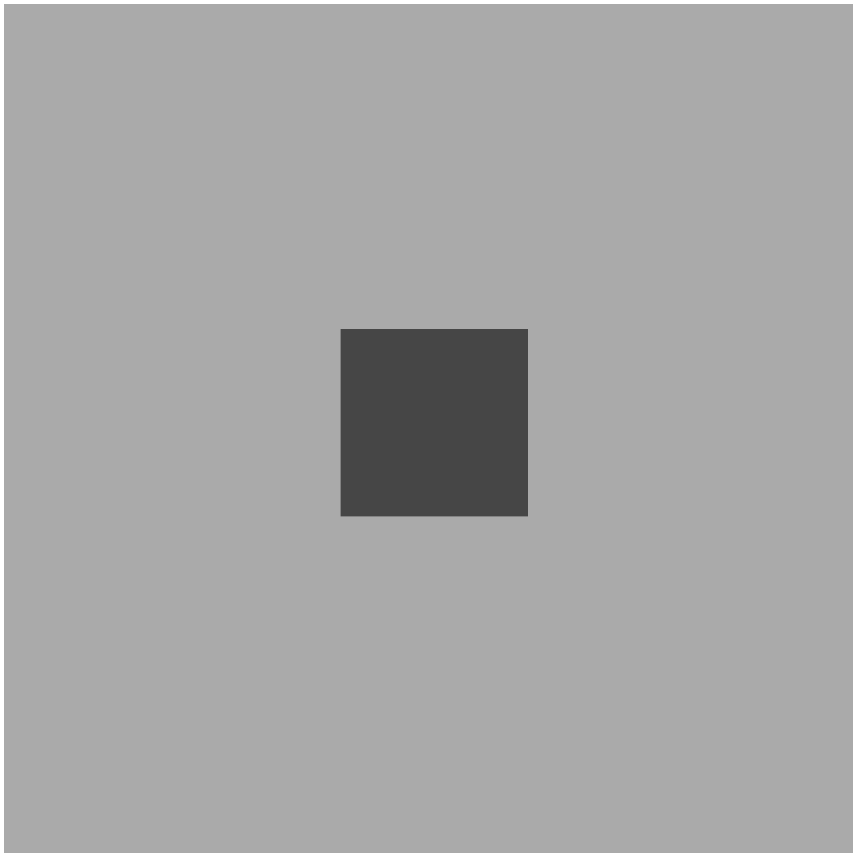
Sample Kernel Code:

```
1  __global__ void kernel (unsigned char *img)
2  {
3      int tid = threadIdx.x + blockIdx.x * blockDim.x;
4      while (tid < N)
5      {
6          if (img[tid] != B)
7              img[tid] = 255;
8          tid += blockDim.x * gridDim.x;
9      }
10 }
```

Listing 1: Grey-Scale

Milestone 1:

Input



Milestone 1:

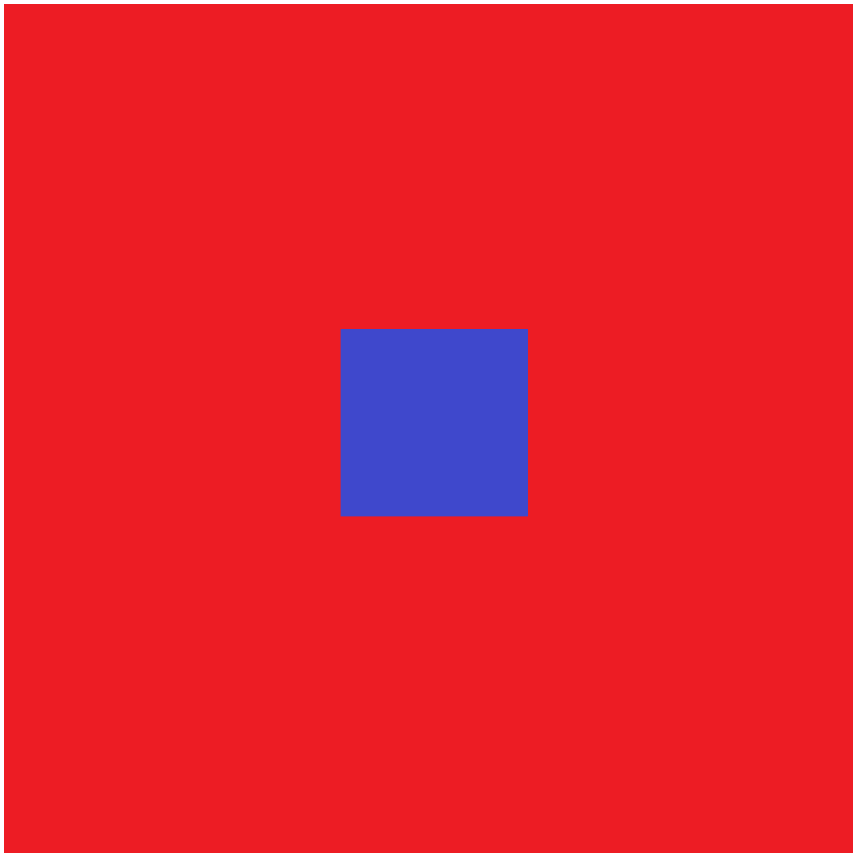
- Example of (Grey-Scale) Image Filtering
- Grey-Scale Image Input/Output
- No Comparisons Between Pixels
- Establish Basic Block/ Thread Indexing Scheme

Milestone 1:

Result

Milestone 2:

Input



Milestone 2:

Simple Color Box

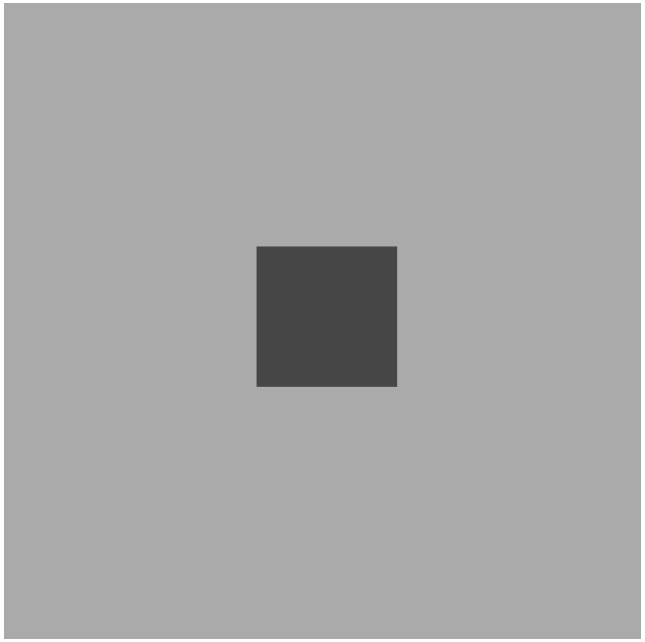
- Color Image Filtering
- Indexing Complexity given RGBA pixel arrays
- Color Image Input/ Output

Milestone 2:

Result

Milestone 3:

Input



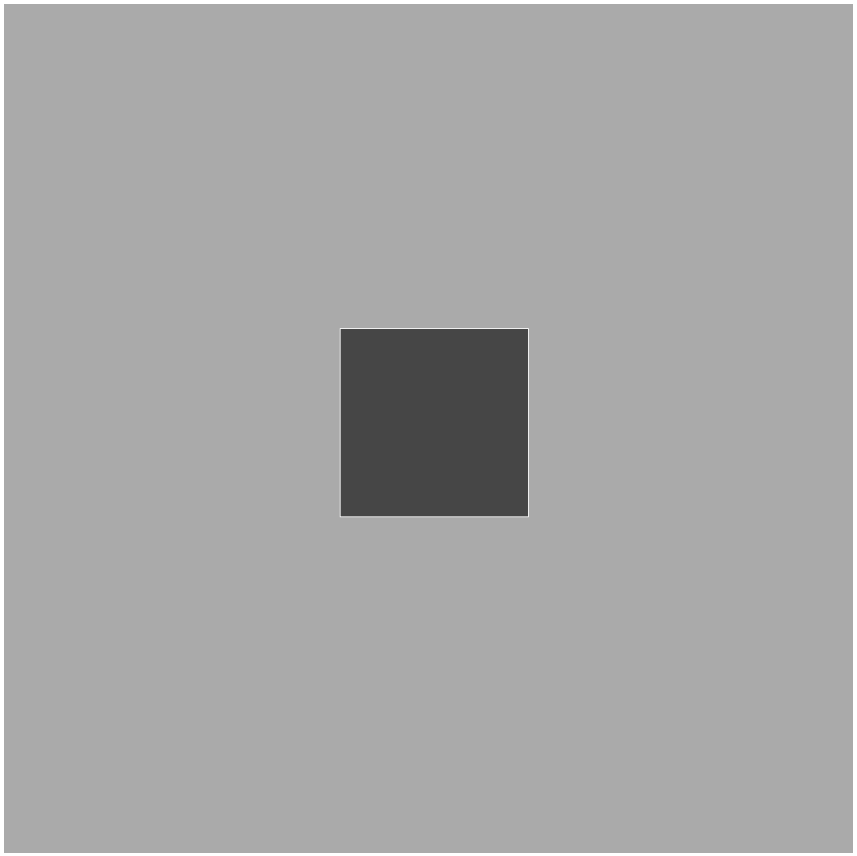
Milestone 3:

Grey Box Edge Detection

- Simple Edge Detection
  - Single Pixel Width Outline
- Grey-Scale for simplicity of indexing
- Comparisons Between Pixels
- Synchronization Issues

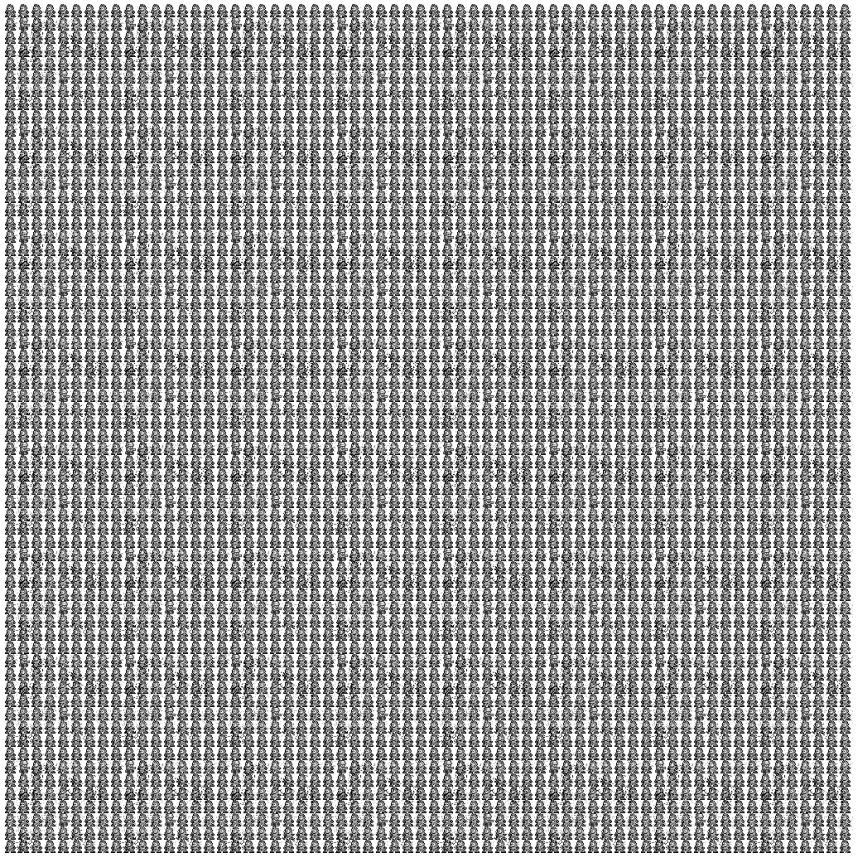
Milestone 3:

Result



Milestone 4:

Input



Milestone 4:

Grey-Scale 16 x 16 Pixel Pattern Detection

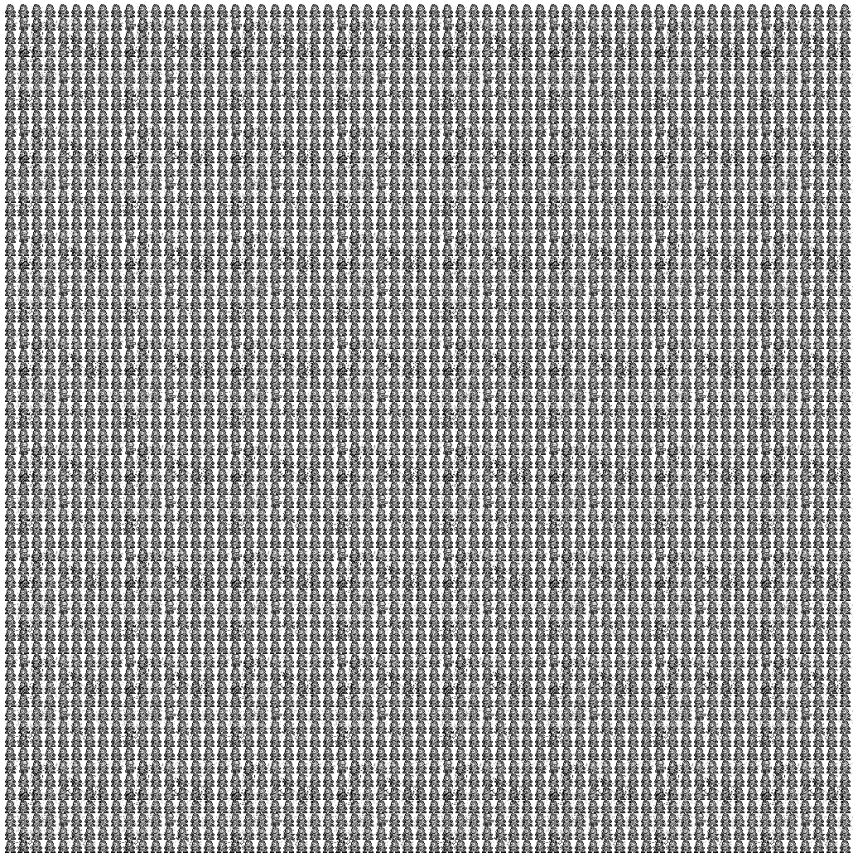
- 3 Kernels, each filtering further down to the pattern
  1. Identify the start of the pattern
  2. From the starting pixel, identify the rest of the pattern
  3. "White-out" any pixel that was not identified previously
- Grey-Scale Pattern Detection
- Application of previous milestones: filtering non-flagged pixels
- *More* Synchronization Issues

Milestone 4:

Result

Milestone 5:

Input



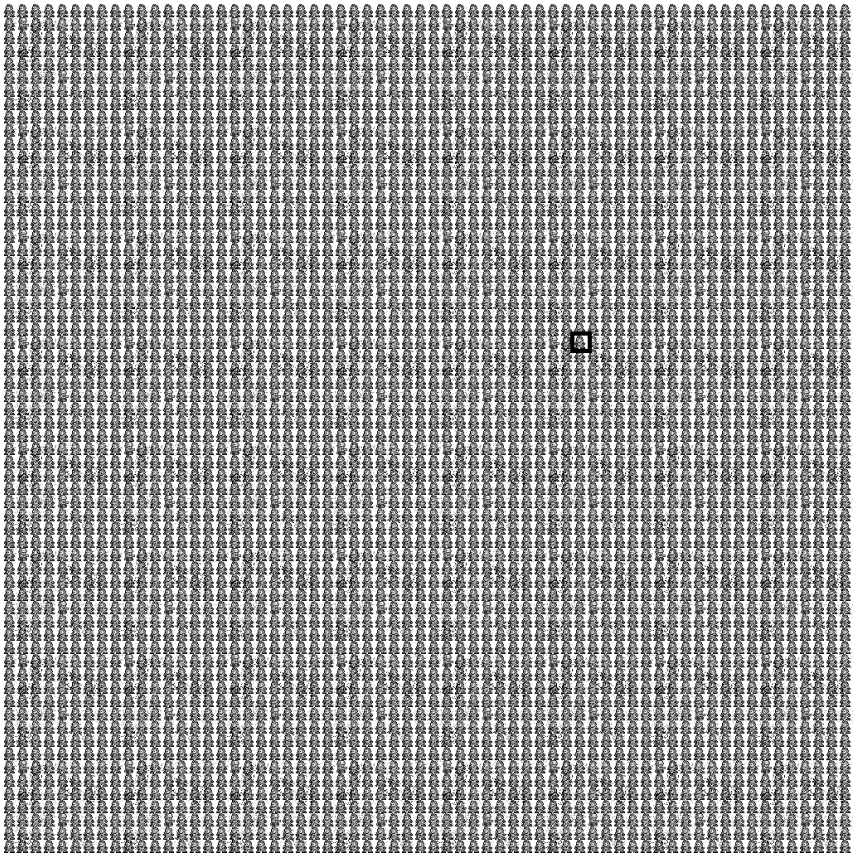
Milestone 5:

Grey-Scale 16 x 16 Pixel Pattern Detection, Outlined

- 3 Kernels, each filtering further down to the pattern
  1. Identify the start of the pattern
  2. From the starting pixel, identify the rest of the pattern
  3. Draw an outline around the identified patterns
- Varying degree of thickness of outline
- Indexing Complexity given attempting to outline pattern

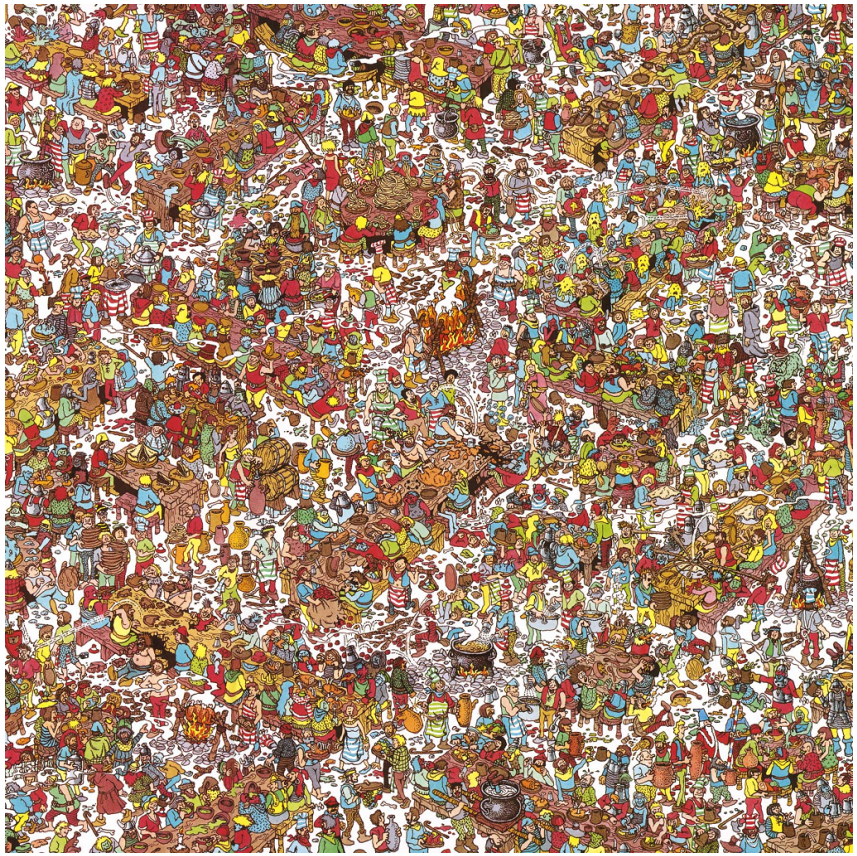
Milestone 5:

Input



Milestone 6:

Result



Milestone 6:

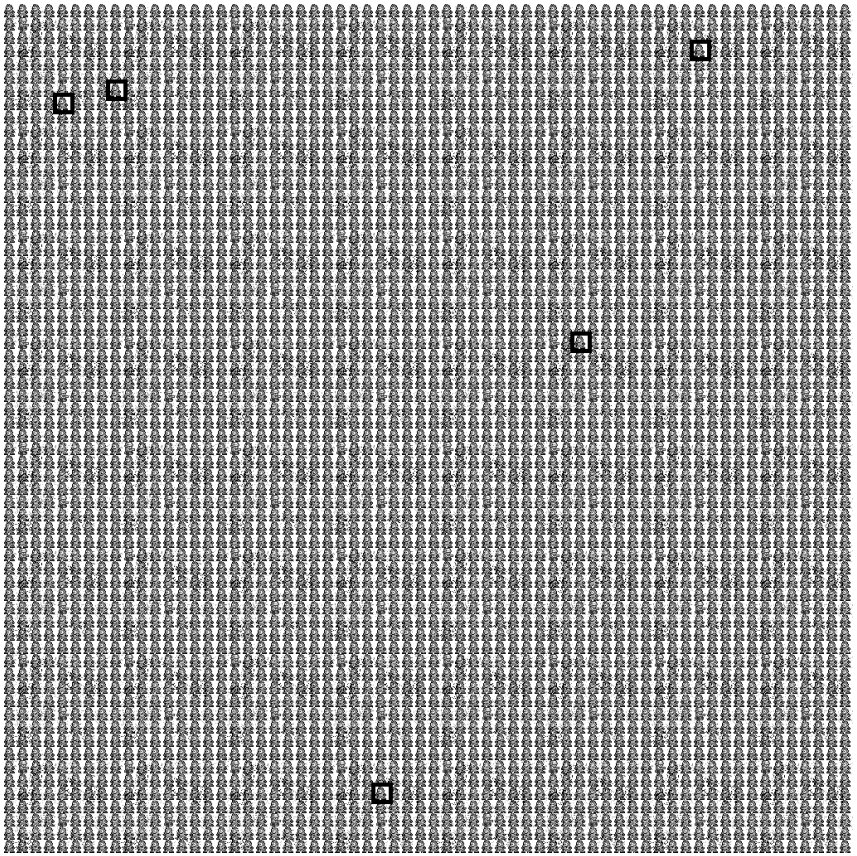
16 x 16 "Waldo" Pattern Detection

Very similar to our color pattern algorithm:

- 3 Kernels, each filtering further down to the pattern
  1. Identify the start of the pattern
  2. From the starting pixel, identify the rest of the pattern
  3. Draw an outline around Waldo's face
- Color Pattern Detection, outlined
- (statically) Found Waldo!

Milestone 6:

Result



Demonstration

Results

Performance Comparison

Average Running Times (seconds):

Milestone	**CUDA C/ C++**	**C**
Grey-Scale	0.157	0.052
Color-Scale	0.257	0.133
Outline Grey	0.201	0.065
Grey Pattern	0.303	0.953
Outline Pattern	0.355	1.227
Outline (Color) Pattern	0.620	1.278

Discussion/ Conclusion

- Our approach:
  - Promising for simple things (rules for better approaches)
  - Not so promising when the patterns or rules become complex
- Other Approaches
  - Neural Networks/ Machine Learning
    - Better suited toward Pattern Recognition Problem
- CUDA C/C++ can substantially decrease running times

Questions?

Thank You for listening and we invite your questions!