Using CUDA to Optimize a Stereo Vision Algorithm

Takeshi Kevin Musgrave, Yasin Nazzar, James Clark - Electrical and Computer Engineering

Introduction

In 3D movies, window violations are visual defects caused by objects moving off-screen in one of the stereo images while remaining visible in the other [1].
The following is one method for detecting window violations:

- Create a depth map to find pixels appearing in front of the screen
- Group together pixels that have similar features, using a clustering algorithm
- Detect if any of these groups move too close to the edge of the image

This can be sped up by using Compute Unified Device Architecture (CUDA), a parallel computing platform for Nvidia's graphics processing units (GPUs).

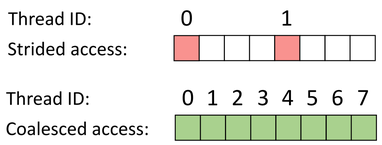
Accessing Memory in CUDA

Global memory is accessible to all threads. For maximum speed, data should be stored and retrieved in a contiguous (coalesced) manner. This is because CUDA accesses global memory in groups of 32, 64, or 128 bytes.

Shared memory is faster than global memory, and its memory accesses do not need to be coalesced. However, shared memory is divided into banks, and if two or more threads try to access one bank at the same time, a bank conflict occurs, which results in serialized performance. As well, shared memory is accessible only to threads within the same group or "block".

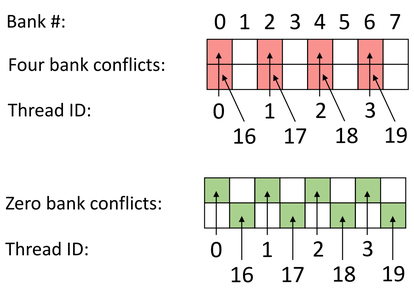
Global memory is used in every CUDA function. When shared memory is also used, the data must first be copied from global memory.

Global memory access



Each square represents a GPU memory location holding 4 bytes. In the strided example, threads access every fourth element. This means when the GPU loads 32 bytes of memory, only 8 of those bytes are actually used. In contrast, when access is coalesced all 32 bytes are used.

Shared memory access



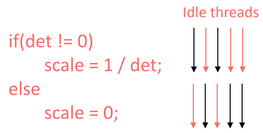
In the top example, threads 0 and 16 both try to access bank 0, so thread 16 has to wait for thread 0 to finish, before gaining access. In the bottom example, threads 0 and 16 can access memory at the same time because they are accessing different banks.

Optimizing the Code

Warps are groups of threads that walk through the code in lockstep. If the threads in a warp do not all follow the same path in the code (i.e. diverge), then some of those threads will be idle. Ensuring there is just one path for every warp will keep all threads busy and improve processing speed.

Other ways of improving the performance of a CUDA program include minimizing the number of memory copy operations between the CPU and GPU, and using 32-bit floating point numbers instead of double precision numbers or integers.

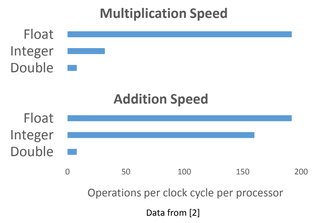
Thread divergence



No thread divergence

```
scale = (det!=0) / (det+1e-37);
```

Float vs Integer and Double



Depth Map Process

1. Determine video size and allocate GPU memory. This is done just once.

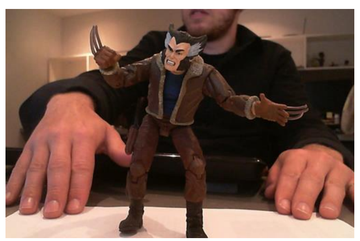
2. Load image into CPU memory

3. Copy image from CPU to GPU

4. Convert image to grayscale and perform edge detection

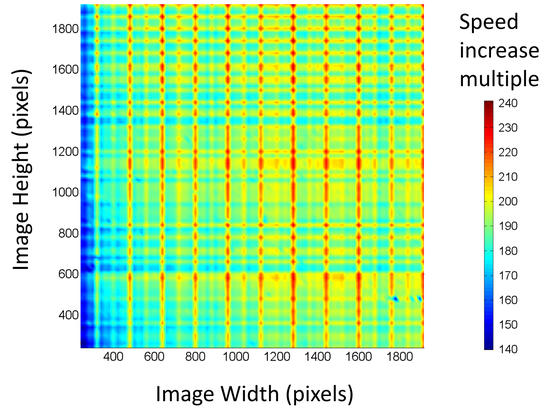
5. Test a range of depth values and find the best one for each pixel [3].

The dashed box indicates code that was timed.



The bottom image is an example of a depth map. Lighter shades of gray indicate areas that are closer to the viewer.

Image Depth Map Speed Increase



This heat map shows how many times faster the CUDA code creates depth maps compared to the original code. For a sense of time: the original code tests 121 depth levels on a 1080p image in 10 minutes, while the CUDA code does the same in under 3 seconds.

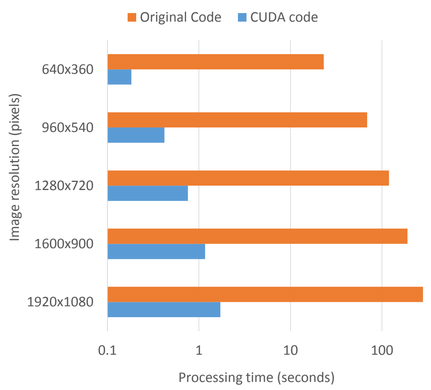
The red vertical lines in the heat map are resolutions at which the number of threads per thread-block is a multiple of the warp size, which is 32. The number of threads has such a large effect that 1280x720 is processed in less time than 1180x720.

A possible solution:
Images with non-ideal resolutions could be padded with extra pixels. For example, if the input is 1180x720, horizontal pixels could be added so that the CUDA functions operate on 1280x720. Before writing the output to disk, the program would trim the size back down to 1180x720.

Focus Estimation Speed Increase

In this focus estimation map, lighter shades of gray indicate areas of the image that are in greater focus.

The focus estimation map, depth map, and grayscale image are the inputs to the clustering algorithm. Thus, pixels that are at similar focus, depth, and color-intensity levels will be grouped together and tracked for window violations.



This graph shows that the CUDA code generates a focus estimation map 100+ times faster than the original code. The time scale is logarithmic.

References and Hardware

[1] Mendiburu, B. (2009). 3D movie making (1st ed.). Burlington, MA: Elsevier.
[2] Nvidia Corporation (2014). CUDA C Programming Guide [Online]. http://docs.nvidia.com/cuda
[3] Hosni et al. "Real-time local stereo matching using guided filter". ICME 2011

Hardware used:
Intel Xeon E5-1607 3.00 GHz CPU
Nvidia Quadro K4000 GPU