University of the Pacific

Lab Report

CUDA: Convolution

Gary R. Roberts, Jr.

g_roberts@u.pacific.edu

Edited: April 21, 2017

1 Abstract

The purpose of this lab is to compare the performance gains of different optimization strategies for convolution on GPGPUs. In order to explore this, an algorithm for several levels of optimization will be designed for GPGPUs using the initial serial algorithm as a start point.

2 Introduction

GPGPUs provide several thousands of threads for use in parallel computation. This is inherently powerful for problems that are highly concurrent. Convolution satisfies this requirement as a matrix manipulation problem but it requires a significant amount of memory accesses. Three different levels of memory access optimization will be implemented with an emphasis on shared memory.

3 Methodology

Three pairs of convolution kernels were created: $L0$, $L1$, and $L2$. In these kernels, filters are used to convolve the matrix in two passes: one for the horizontal neighbors and one for the vertical. This effectively reduces the problems from a $M * N$ to a $M + N$ problem. The same algorithm is used in the serial implementation. A block size of 256 results in the most effective segmentation for optimal multiprocessor occupancy.

3.1 L0

The first level of optimization uses global device memory for the convolution operations. The host passes the image to the device, the image is convolved and then the host copies the resulting matrix back into host memory.

3.2 L1

The second level of optimization uses cooperative loading into shared memory for the convolution operations. Aprons are used to store the bordering pixels of adjacent blocks. These aprons are loaded cooperatively depending on the filter:

1. **Vertical:** The top aprons are loaded by the corresponding threads one level above the apron in question. The bottom aprons are loaded by the corresponding threads one level below the apron in question. If the levels are extreme, the values are set to zero.

2. **Horizontal:** The left aprons are loaded by the corresponding threads one level to the left of the apron in question. The right aprons are loaded by the corresponding threads one level to the right of the apron in question. If the levels are extreme, the values are set to zero.

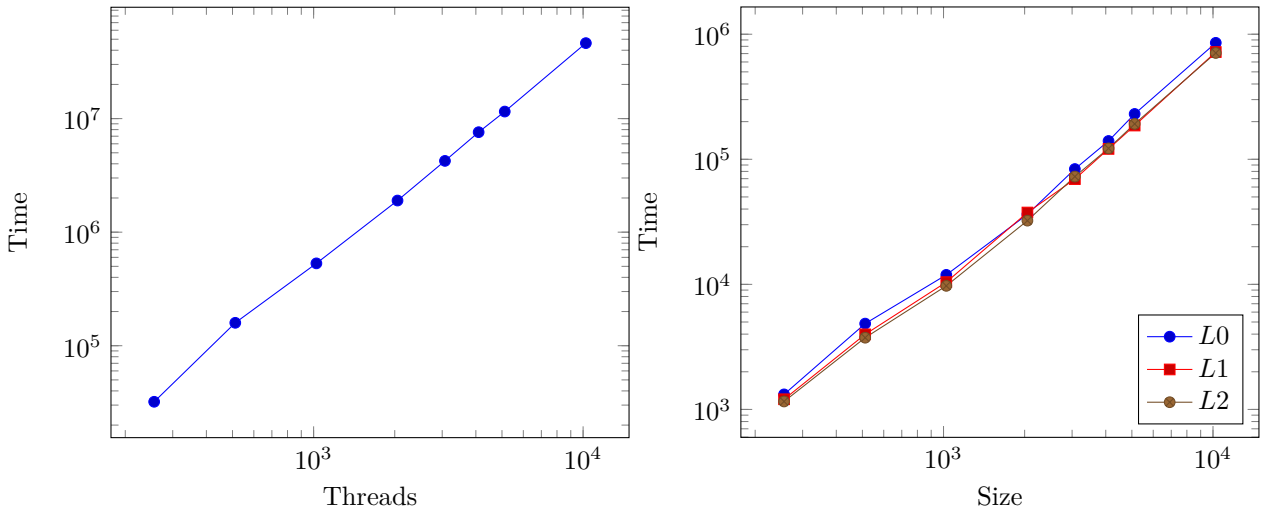
3.3 L2

The third level of optimization uses cooperative loading into shared memory for the convolution operations. Aprons are loaded using accesses to the global memory relying on the functionality of the L2 cache for speedy access.

4 Results

4.1 Timing

The resulting execution times for $L0$, seen in figure one, clearly offers several orders of magnitude of optimization. $L1$ and $L2$ offer some improvement from $L0$, with $L1$ performing better on larger images, and $L2$ outperforming $L1$ on smaller images.



(a) Serial Execution time vs. Image Size

(b) Execution Time vs. Optimization Levels

Figure 1: Execution Time vs. Image Size

4.2 Speed-Up

The values for each optimization level can be found in table one below. The speedup for $L0$ was computed using the serial version as the initial timing. Each successive speedup was computing using the preceding level.

The speedup for $L0$ ranged from 20% for smaller images to 50% for larger images. The speedup for $L1$ consistently provided a 9-30% boost. The speedup for $L2$ was greater than one for smaller images but less than one for larger images.

Size	$L0$	$L1$	$L2$
256	24.32	1.09	1.04
512	32.78	1.21	1.06
1024	44.59	1.14	1.07
2048	52.37	0.97	1.16
3072	50.76	1.29	0.95
4096	54.22	1.15	0.99
5120	50.05	1.23	0.97
10240	54.15	1.18	1.02

Table 1: Speed-Up For Each Level Dependent on Size

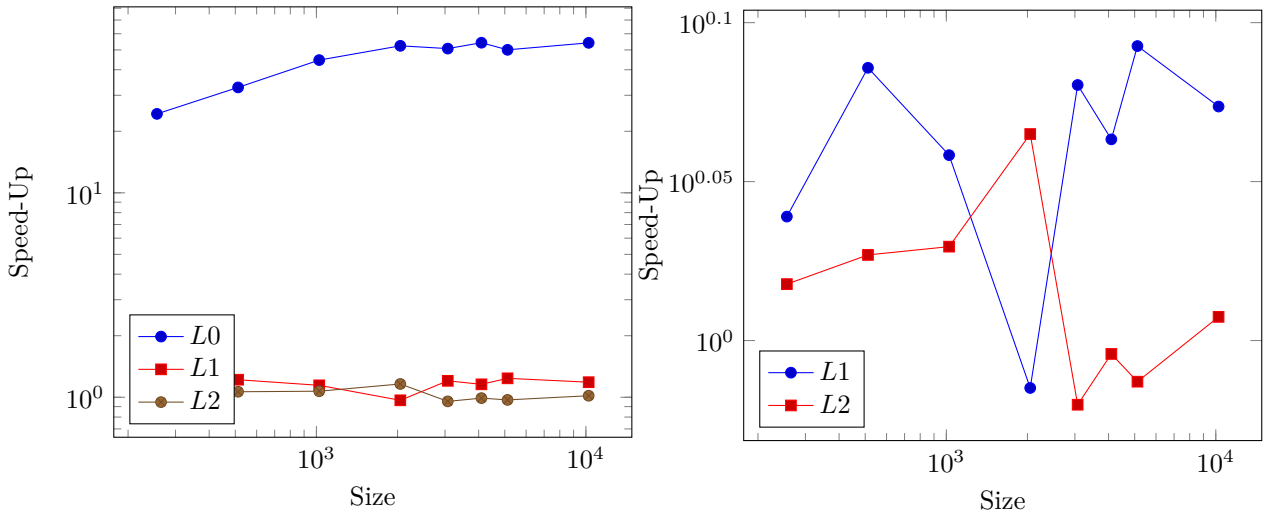


Figure 2: Speed-Up vs. Image Size per Level

5 Conclusion

$L0$ clearly offers a significant boost in performance over the serial implementation. By utilizing shared memory both the $L1$ and $L2$ versions offer a speedup increase of up to 30%. The $L1$ version achieves better performance with images of larger size wheras the $L2$ version performs greater with images of smaller sizes. This can be attributed to the size of the $L2$ cache. Images of greater size will be less likely to be present in the $L2$ due to its limited size. The $L1$ cache does not have this problem due to its manual apron loading using local shared memory and global ids.