Payman Behnam
U1078370

Sobel-Optimization
Parallel Programing for Many-core Systems

In this assignment, I am implementing an optimized version of sobel algorithm. I make use of different approaches to reduces the time as much as possible.

What is Sobel:

Edge detection includes various mathematical approaches that tries to define points in a digital image while the image brightness changes sharply in corners. Sobel if one of the famous one that tries to detect all the edges based on convolution. It convolves the image with a small, separable, and integer-valued filter in the horizontal and vertical directions and is therefore relatively inexpensive in terms of computations.

Optimization techniques:

1- Baseline: sobel_gpu_base:
   In the base line, we make use of global memory without any optimization This is the baseline we used in previous assignment as well.
2- Register Reuse: sobel_gpu_regreuse:

In the sobel algorithm, we make use of two sums to compute the magnitude. *magnitude = (sum1*sum1) + (sum2*sum2).* sum1 and sum2 have several shared components. Hence, we can define some register once and use them whenever we want. To do so, I defined the Downl, CntyL, UpL, DownR, CntyR, UpR, DownCntx, UpCntx, and use them in both Sum1 and Sum2. Experimental results show that It doesn't reduce the time too much. Apparently, compiler does this technique implicitly.

3- Shared memory: sobel_gpu_shared:

In this optimization, I tried to make use of shared memory instead of using global memory. I copied the part of image into shared memory and tried to make use of that. However, the edges and corners should be treated differently. So, we have to make use of different *if-else statement* to handle out of bound accesses. This technique leads to time reduction.

Sobel needs the boundary pixels, so shared memory size is allocated in this way. To compute sobel operation for n*n, we make use of block size (n+2)(n+2) . Each block image is copied to shared memory. We make us of __syncthread() so that synchronization is done to each block. Figure 1 shows that if we have an image, we can divide in into four segment and put each segment along with halo regions around that in shared memory.

Accessing the shared memory (within blocks) rather than global memory takes less time; So the timing will be reduced.

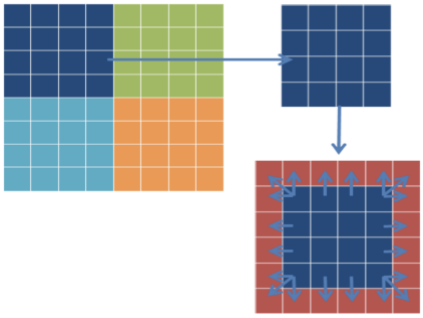

Figure 1-Sobel Operation using shared memory[1]

4- Sharedmemory + Controlflow simplification: sobel_gpu_shared2
To mitigate *"if then else statement"* in the shared version, we can compute sobel operations in another way. To describe this, I will explain an example.
Suppose that Previously(sobel_gpu_shared2), we had 32*32 threads but computation is done in 30*30 area. For copy from global memory into shared memory, some threads did one copy and some did more than one copy operation. Now, I want that each thread copies just one pixel to shared memory. In computation, we do some overlapping between different blocks. So, I don't need to consider a lot of *if statement*. The following figure shows what happens in terms of different blocks.

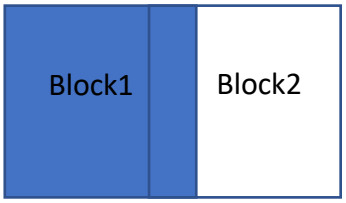

Figure 2- Sobel Operation using overlapping shared

Removing if-statement significantly and using sharing overlap will help to reduce the timing. Since there is no need for threads to check a lot of conditions all the time.
4. Approximate Computing: sobel_gpu_appx
Approximate computing is a technique while we don't need the precise results. In fact, an approximate of results is enough many applications. Many DSP applications can take advantages of these possibility. If we are back to the way that sum1 and sum 2 are computed, we can observe some shared components (4/6). However, we can modify different components (2/6) a little in a way that make all of the component similar (the weights of the components are still different) without affecting too much in the final result. In this way, we can increase the registers reuse, avoid some computations and increase performance while delivering acceptable results.
This will lead to more reuse of data and hence decrease the gpu time.

Experimental Results:

The timing results a for different methods described in the above along with number of thread in each block are presented in the table and following plot. Please pay attention that the way that we measure timing (i.e. considering overheads, warm-up etc.) may affect the final result. The results are the average of 30 tries. With comparison with base line system the amount of improvement is about 40.1%. The details of implementation are commented in the code. For the last two operations, I tried 16*16 as well. The timing results was 0.385 but the quality of the image was not acceptable.

Table 1-Num of Threads in blocks and timing results of different optimization techniques

Method	ThreadDim	GPU Time (ns)
Baseline	16*16	0.661
Baseline	32*32	0.651
Register Reuse	16*16	0.637
Register Reuse	32*32	0.651
Shared memory	16*16	0.650
Shared memory	30*30	0.645
Approximate Computing	32*32	0.571
Approximate Computing	16*16	0.520
Shared memory + controlflow simplification +registerreuse	32*32	0.441
Shared memory + controlflow simplification +registerreuse+approximatecomputing	32*32	0.398

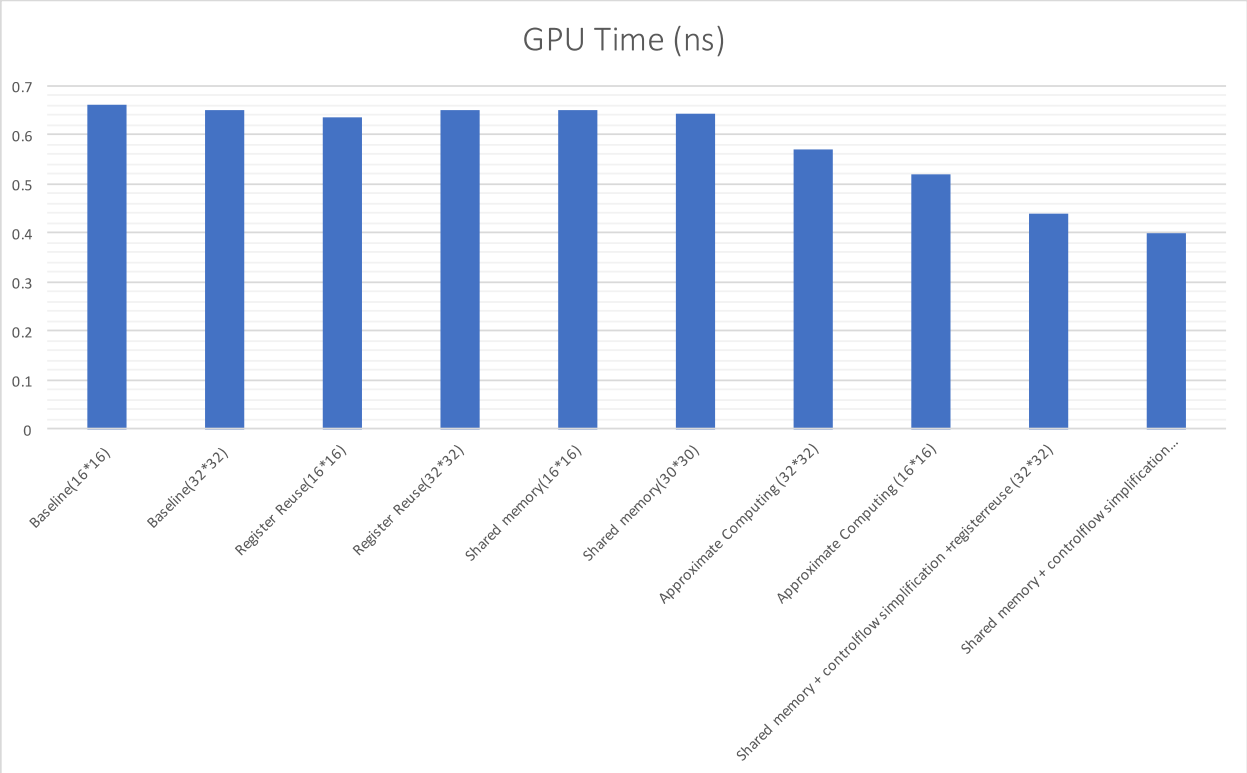

Figure 2- Timing Result of different optimization technique

References

[1]. Ji Hoon Jo and Sang Gu Lee, "Sobel mask operations using shared memory in CUDA environment," *2012 6th International Conference on New Trends in Information Science, Service Science and Data Mining (ISSDM2012)*, Taipei, 2012, pp. 289-292.