TDT4200: Parallel computing

Assignment 7: Edge detection in CUDA

Name: Martin Rebne Farstad

The CUDA files for the tasks are named task1b.cu, task1e.cu and task2b.cu. To build them, run either 'make task1b', 'make task1e' or 'make task2b'.

Task 1

(d & e) The tables and figures below show the result from running the serial version, the basic GPU version and the shared memory version on a lab computer. They really display the power of using a GPU for this calculation.

Iterations	*Runtime (seconds)*	*Speedup vs. Serial*
1	1.574	1.06
1024	2.498	93.0
2048	3.420	135
4096	5.284	175

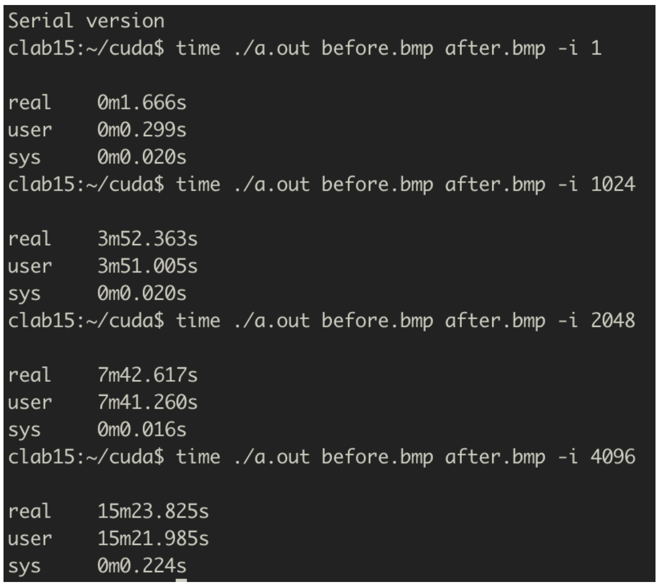
Table 1: Basic GPU version

Iterations	*Runtime (seconds)*	*Speedup vs. Serial*	*Speedup vs. Basic GPU*
1	1.558	1.07	1.01
1024	3.045	76.3	0.82
2048	4.452	104	0.77
4096	7.354	126	0.72

Table 2: Shared memory version

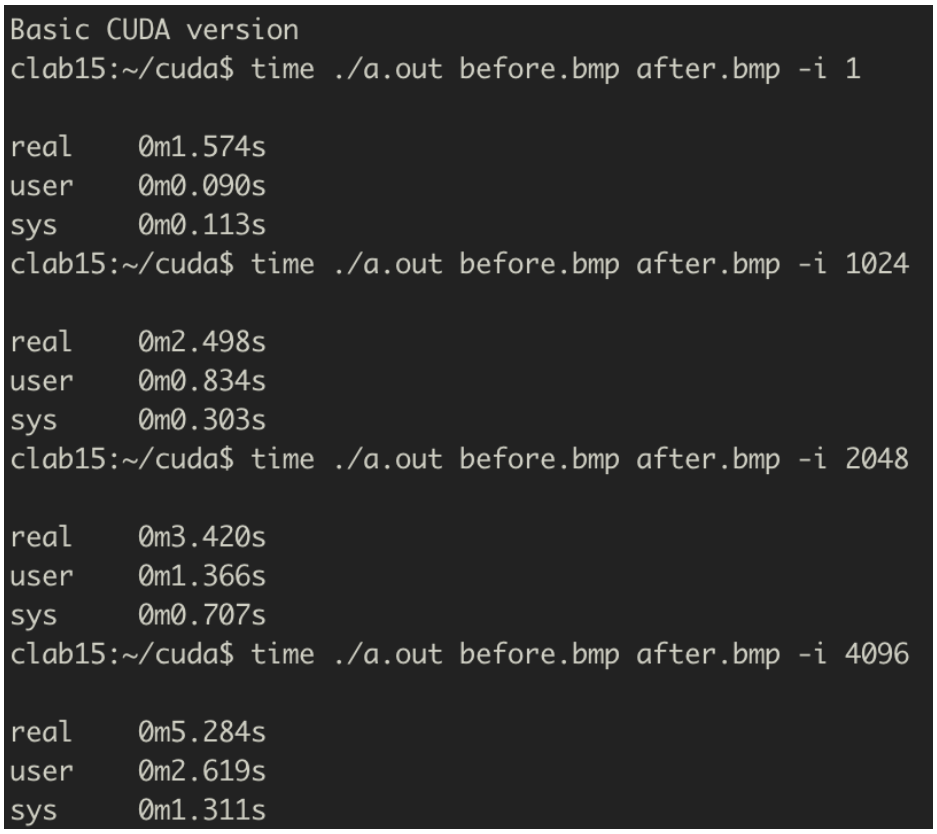
The table above shows that the basic GPU version actually performs better than the shared memory version. This is mostly because of additional reads to global memory and divergence because of additional conditional branches. It needs to access global memory when copying data into the shared memory and when accessing data on the halo.

I made another version of task 1e as well which uses only shared memory. This version performs better than both the basic GPU version and the previously created version for task 1e. However, this version only loads a single halo border and should therefore only work on 3 dimension filters. Because of this and the fact that my implementation of task2b is based on the previous implemented version of task1e, I will not base the timings on this version. This additional kernel will be provided in task1e.cu, but will be commented out.



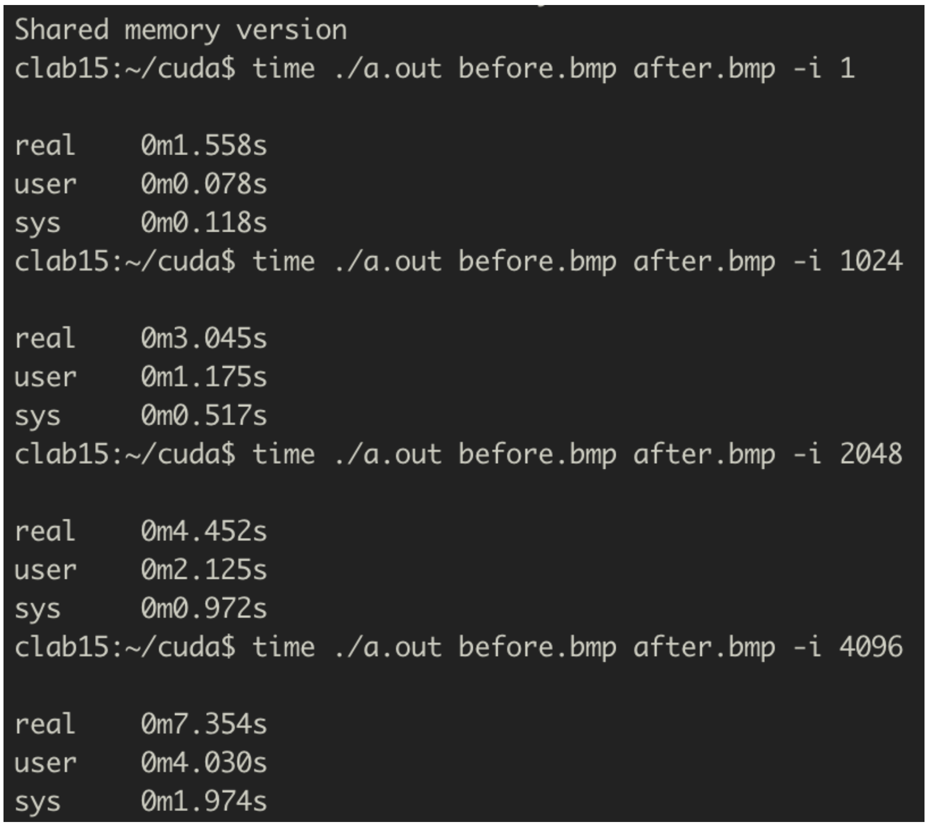
```
Serial version
clab15:~/cuda$ time ./a.out before.bmp after.bmp -i 1

real    0m1.666s
user    0m0.299s
sys     0m0.020s
clab15:~/cuda$ time ./a.out before.bmp after.bmp -i 1024

real    3m52.363s
user    3m51.005s
sys     0m0.020s
clab15:~/cuda$ time ./a.out before.bmp after.bmp -i 2048

real    7m42.617s
user    7m41.260s
sys     0m0.016s
clab15:~/cuda$ time ./a.out before.bmp after.bmp -i 4096

real    15m23.825s
user    15m21.985s
sys     0m0.224s
```

Figure 1: Serial version



```
Basic CUDA version
clab15:~/cuda$ time ./a.out before.bmp after.bmp -i 1

real    0m1.574s
user    0m0.090s
sys     0m0.113s
clab15:~/cuda$ time ./a.out before.bmp after.bmp -i 1024

real    0m2.498s
user    0m0.834s
sys     0m0.303s
clab15:~/cuda$ time ./a.out before.bmp after.bmp -i 2048

real    0m3.420s
user    0m1.366s
sys     0m0.707s
clab15:~/cuda$ time ./a.out before.bmp after.bmp -i 4096

real    0m5.284s
user    0m2.619s
sys     0m1.311s
```

Figure 2: Basic GPU version



```
Shared memory version
clab15:~/cuda$ time ./a.out before.bmp after.bmp -i 1

real    0m1.558s
user    0m0.078s
sys     0m0.118s
clab15:~/cuda$ time ./a.out before.bmp after.bmp -i 1024

real    0m3.045s
user    0m1.175s
sys     0m0.517s
clab15:~/cuda$ time ./a.out before.bmp after.bmp -i 2048

real    0m4.452s
user    0m2.125s
sys     0m0.972s
clab15:~/cuda$ time ./a.out before.bmp after.bmp -i 4096

real    0m7.354s
user    0m4.030s
sys     0m1.974s
```

Figure 3: Shared memory version

Task 2

(a) Synchronization between blocks inside the kernel has not been possible in previous CUDA versions, so to be able to know that each block is done with their computation and are ready for the next iteration we need to synchronize via the CPU. Introduced in CUDA 9, cooperative groups enables the possibility of synchronizing between blocks without leaving the kernel. By using cooperative groups, we can make sure that the data a block needs from other blocks is available for the next iteration. This is the data present on the "halo" border around the block, as presented in Assignment 3. This data will need to be updated and shared for each iteration.

A problem with using cooperative groups is limited shared memory which results in a limitation of the amount of blocks in the grid. This becomes a problem for us, since each thread handles a single pixel. This can be fixed by having the threads handle multiple pixels, which also results in a performance decrease.

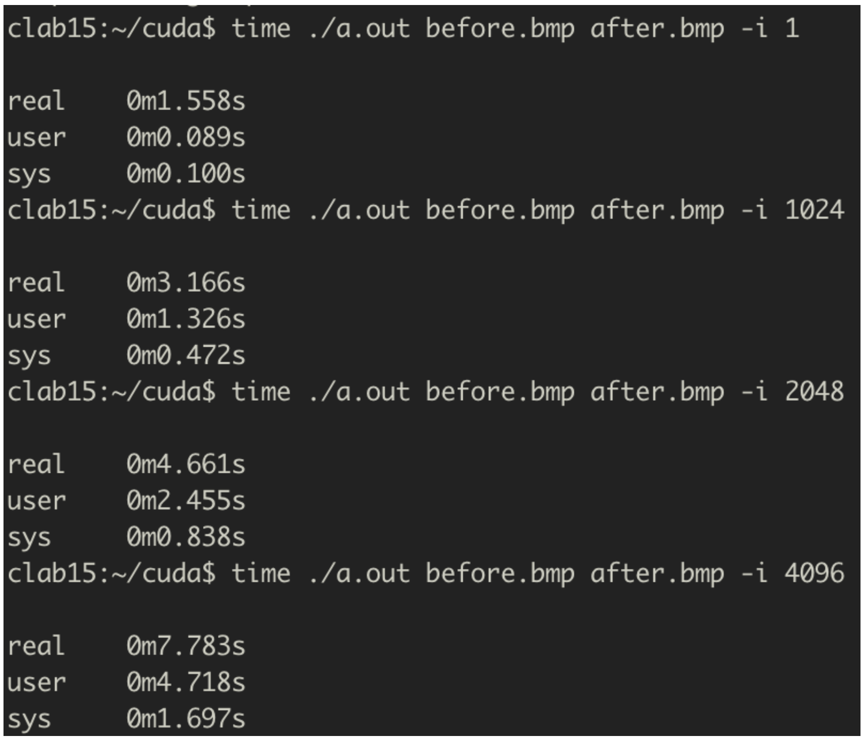
(b) Cooperative groups was introduced in CUDA 9 and might not be available on the default architecture selected by the compiler. I added the arguments -arch=sm_61 -rdc=true to nvicc. This targets the Pascal architecture with compute capability 6.1, which is the architecture present on the lab computers.

My implementation uses synchronization between blocks, and can therefore handle multiple iterations without leaving the kernel. In this implementation I have used cudaOccupancyMaxActiveBlocksPerMultiprocessor to retrieve the amount of simultaneously active blocks per SM. I have adjusted the block size to optimize shared memory usage for performance. I have used the lab computers which have 28 SMs in the GPU. This results in less amount of blocks per grid than my previous solutions, so I needed each thread to handle more than one pixel.

The hints for this task tells to use shared memory for both in and out buffers, but this was difficult due to the fact that the threads handle more than one pixel. Because multiple blocks occupy the shared memory within each iteration, my implementation reads all data from global memory instead of keeping it in the shared memory between iterations. Additionaly, instead of writing only the halo data to global memory, my implementation writes all block data back to global memory each iteration which is also not optimal.

Iterations	*Runtime (seconds)*	*Speedup vs. Serial*	*Speedup vs. Basic GPU*	*Speedup vs. Shared memory*
1	1.558	1.07	1.01	1.00
1024	3.166	73.4	0.79	0.96
2048	4.661	99.3	0.73	0.96
4096	7.783	119	0.68	0.94

Table 3: Cooperative groups version



```
clab15:~/cuda$ time ./a.out before.bmp after.bmp -i 1

real    0m1.558s
user    0m0.089s
sys     0m0.100s
clab15:~/cuda$ time ./a.out before.bmp after.bmp -i 1024

real    0m3.166s
user    0m1.326s
sys     0m0.472s
clab15:~/cuda$ time ./a.out before.bmp after.bmp -i 2048

real    0m4.661s
user    0m2.455s
sys     0m0.838s
clab15:~/cuda$ time ./a.out before.bmp after.bmp -i 4096

real    0m7.783s
user    0m4.718s
sys     0m1.697s
```

Figure 4: Cooperative group version