Burak Cem Balcı 60100
Doruk Taneli 60066

CUDA Parallel Noise Remover

We completed all versions of the project, and they all give the correct answer, although performance is not optimal.

How to recreate the project

1. Extract files to workspace in kuacc
2. Run >“module load cuda/9.1”
3. Set block size by setting the #define SQRT_BLOCK_SIZE to the square root of desired value in v1, v2, and v3 of the code.
4. Run >”make” to compile serial, v1, v2, and v3 of the code simultaneously
5. Submit the job script with >”sbatch submit.sh”
6. Observe the results from the .out file

Part 1

We divided the computations in the serial part into 3 functions named compute_1, compute_2, and reduction. In the main function we defined device variables for most of the variables in the serial code. We allocated device memory for the variables and then copied from the host. Then we created a grid and a thread variable to pass in the kernel functions. We replaced the parts reduction, compute 1 and compute 2 with the kernels by keeping the iteration loop the same

In the computation kernels we constantly access global memory variables which makes it naïve. In the kernels compute_1 and compute_2 we replaced the for loops with indices defined according to threadId, blockDim, and blockID. We left the computations as they were with an additional if statement that checks for bounds.

For the reduction part we followed the instructions in https://developer.download.nvidia.com/assets/cuda/files/reduction.pdf.

Burak Cem Balcı 60100
Doruk Taneli 60066

Part 2

The main purpose of part 2 was to overcome the performance problems by eliminating the global memory references. To solve this problem we employed local variables that are defined in the kernels. With the use of temporary variables in registers the performance improved slightly.

Part 3

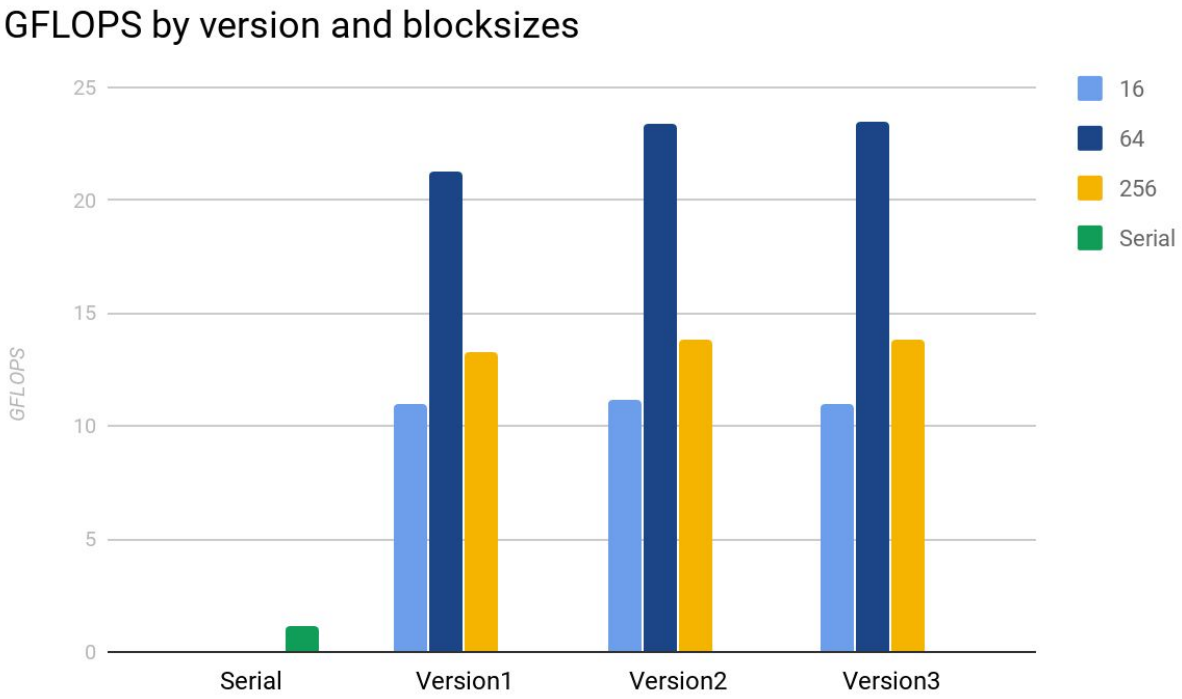
In this part we introduced image as an additional two-dimensional block into the shared memory in compute_2. We distributed the computation to threads within the same thread blocks. After the computations, we rewrote the shared values back to the local device memory, and then copied it back to host using cudaMemcpy.

Results

Average of sum of pixels was consistently 116.080063 across all block sizes and versions. It differed from the serial version by 0.00580, which was 116.074257. We believe that amount of difference is acceptable.

Below are the GFLOPS we got from our tests as a graph and the numerical values. We achieved a huge difference between the serial version and the parallel versions. We got the best results when the blocksize was 64, a 16 by 16 2D block. We achieved around a 20x increase in GFLOP between the serial and parallel versions. Comparing the parallel versions, The increase in speed is very small, but it is still existent. Each version performed slightly better than the previous version.

Burak Cem Balcı 60100
Doruk Taneli 60066



	16	64	256	Serial
Serial				1.112832
Version1	11.015689	21.235992	13.278566	
Version2	11.156005	23.401165	13.800064	
Version3	11.02281	23.50013	13.79359	