Ashkan Vedadi Gargary | **aveda002 | 862352576 | HW2**

1. **For the naive reduction kernel, how many steps execute without divergence? How many steps execute with divergence?**

- It generates a tree for its kernel function. The number of elements is 1024 and the block size is 512 (2 elements for each thread) and the number of steps is Log(1024) which is 10.
- Because all of the threads are used for first steps, only the first step executes without divergence (after that half of the threads will do nothing, also, in the naive algorithm the neighbor threads will not work after each step).

2. **For the optimized reduction kernel, how many steps execute without divergence? How many steps execute with divergence?**

- With optimized reduction kernels, we can see that the number of threads is less than a warp and therefore the first 5 steps (512, 256, 128, 64, 32) are executed without divergence and after that they will diverge.
- Because the size of the warp is only 32 threads and we need to wait until the thread size is less than equal to 32. (also, the optimize algorithm is like using neighbor threads in a single warp to reduce warp occupancy)

3. **Which kernel performed better? Use profiling statistics to support your claim.**

- Naive:

```
gpu_sim_cycle = 126806
gpu_sim_insn = 71024154
gpu_ipc =     560.1009
gpu_tot_sim_cycle = 126806
gpu_tot_sim_insn = 71024154
gpu_tot_ipc =     560.1009
gpu_tot_issued_cta = 0
gpu_stall_dramfull = 2686
gpu_stall_icnt2sh    = 11024
gpu_total_sim_rate=125706
```

- Optimized:

```
gpu_sim_cycle = 91144
gpu_sim_insn = 62524254
gpu_ipc =     685.9942
gpu_tot_sim_cycle = 91144
gpu_tot_sim_insn = 62524254
gpu_tot_ipc =     685.9942
gpu_tot_issued_cta = 0
gpu_stall_dramfull = 3373
gpu_stall_icnt2sh    = 35149
gpu_total_sim_rate=139252
```

Optimized version is better for sure. By looking at statistics we can see that the number of cycles in the optimized version is lower than the naive version. Even by looking at the gpu_sim_insn we can see the number of instructions in the last kernel in the optimized version is lower than the naive version.

4. **How does the warp occupancy distribution compare between the two Reduction implementations?**

- Warp Occupancy Distribution (naive):

  Stall:146546 W0_Idle:56748 W0_Scoreboard:315050 W1:369306 W2:187584 W3:0 W4:187584 W5:0 W6:0 W7:0 W8:187584 W9:0 W10:0 W11:0 W12:0 W13:0 W14:0 W15:0 W16:187584 W17:0 W18:0 W19:0 W20:0 W21:0 W22:0 W23:0 W24:0 W25:0 W26:0 W27:0 W28:0 W29:0 W30:0 W31:0 W32:2125882

- Warp Occupancy Distribution (optimized):

  Stall:79961 W0_Idle:107377 W0_Scoreboard:434180 W1:13678 W2:7816 W3:0 W4:7816 W5:0 W6:0 W7:0 W8:7816 W9:0 W10:0 W11:0 W12:0 W13:0 W14:0 W15:0 W16:7816 W17:0 W18:0 W19:0 W20:0 W21:0 W22:0 W23:0 W24:0 W25:0 W26:0 W27:0 W28:0 W29:0 W30:0 W31:0 W32:2039906

- The warp number of warp in the optimized version is lower than the naive (you can consider w1,w2,w4,w8,w16,w32). It is true because the optimizer version has fewer warp occupancy.

5. **Why do GPGPUs suffer from warp divergence?**

- GPGPU is good when the amount of data processing is larger than the needs of data scheduling. In fact, SIMD is used for doing the same structure for multiple data (like binary operations). But, our code will be more serialized in comparison with normal use of GPGPU and therefore it has lower performance because of warp divergence (in other words, GPGPU is a warp based system which is a challenge).
- However, in general, Because of SIMD instruction GPGPUs have better performance than CPUs.