ASSIGNMENT - 2

Name - Srushti Hembade
Student Id - 862395839
Net id - shemb001

Question 1: For the naive reduction kernel, how many steps execute without divergence? How many steps execute with divergence?
→ Given block size = 512 and Number of steps = log2(512) = 9
As per the results, 9 Steps are needed through execution.

There is no divergence because the partial sum is computed in the first step using all of the threads. Following the first eight stages, half of the threads will be idle while the remaining eight threads operate in separate directions.

Question 2: For the optimized reduction kernel, how many steps execute without divergence? How many steps execute with divergence?
→ Less effective threads than a warp are present in the first four phases; there is no divergence in this sequence. The latter five phases, on the other hand, are executed divergently.

Question 3: Which kernel performed better? Use profiling statistics to support your claim.
→ Please look for the explanation and screenshots for the same below -
Naive Reduction -

```
kernel_name = _Z14naiveReductionPfS_j
kernel_launch_uid = 1
gpu_sim_cycle = 126295
gpu_sim_insn = 67524378
gpu_ipc =     534.6559
gpu_tot_sim_cycle = 126295
gpu_tot_sim_insn = 67524378
gpu_tot_ipc =     534.6559
gpu_tot_issued_cta = 0
gpu_stall_dramfull = 2476
gpu_stall_icnt2sh    = 14887
gpu_total_sim_rate=662003
```

Reduction-Optimized -

```
kernel_name = _Z18optimizedReductionPfS_j
kernel_launch_uid = 1
gpu_sim_cycle = 124743
gpu_sim_insn = 94040158
gpu_ipc =     753.8712
gpu_tot_sim_cycle = 124743
gpu_tot_sim_insn = 94040158
gpu_tot_ipc =     753.8712
gpu_tot_issued_cta = 0
gpu_stall_dramfull = 1822
gpu_stall_icnt2sh    = 18210
gpu_total_sim_rate=734688
```

Comparing the naïve reduction kernel to optimized reduction, the former has higher values in some attributes (such as GPU_sim_cycle and GPU_tot_sim_cycle) and lower values in other attributes. For this reason, an optimized reduction kernel outperforms a naive reduction kernel.

Question 4: How does the warp occupancy distribution compare between the two Reduction implementations?

→ Please look for the explanation and screenshots for the same below -

Naive Reduction -

```
Warp Occupancy Distribution:
Stall:120719    W0_Idle:57121   W0_Scoreboard:238280    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   W32:1985264
```

Reduction-optimized -

```
Warp Occupancy Distribution:
Stall:146272    W0_Idle:47820   W0_Scoreboard:248626    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   W32:2993528
```

More stalled warps occur in the warp occupancy distributions of optimized reduction kernels than in the warp occupancy distributions of naive reduction kernels, indicating the higher efficiency of the latter.

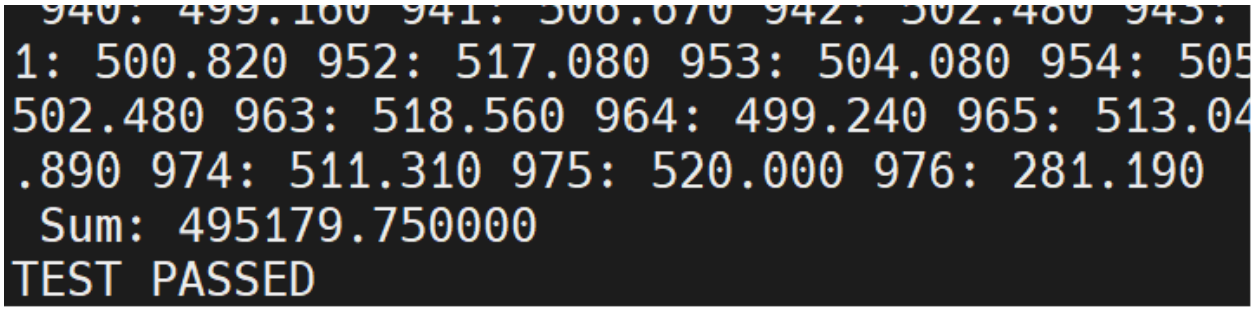
Question 5: Why do GPGPUs suffer from warp divergence?

→ GPGPUs outperform CPUs in SIMD because they spend a significantly larger amount of time on data processing and scheduling than on data transmission and scheduling. The same algorithm is used to process data in SMID. Serialization can occur in a number of ways, resulting in decreased GPGPU performance and warp divergence.

—----------

OUTPUT screenshot from the bender local execution-

Naive Reduction -



Reduction-optimized -