CS217 : LAB ASSIGNMENT 2

Name: Akash Suresh Bilgi | SID: 862395080 | Email: abilg003@ucr.edu

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

A1:

If Block Size = 512
Steps = log 2 (512) = which is 9 Steps.

First step uses all threads, others will stop using the threads as and when the width **halves every time**.
Hence 9-1 step= 8 with divergence

1 Step without divergence
8 Steps with divergence

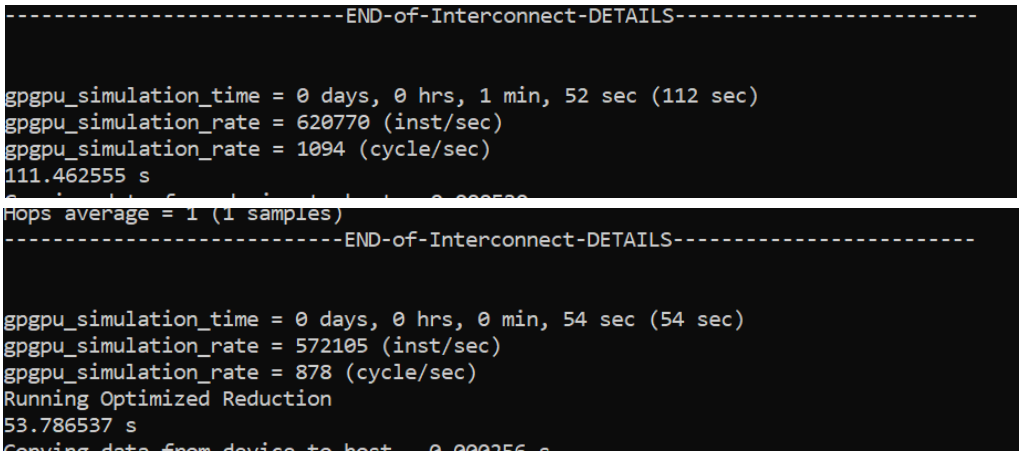
2. For the optimized reduction kernel, how many steps execute without divergence? How many steps execute with divergence?

A2:

Since I'm using the warpReduce function to unroll the **last 6 iterations** of the inner loop. (start<32)
9-1(without divergence)-6(warpReduce) = 2
7 Steps run without divergence
2 Steps run with divergence

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

A3.



```
GPGPU-Sim uArch: GPU detected kernel '_Z14naiveReductionPfS_j' finished on shader 12.
kernel_name = _Z14naiveReductionPfS_j
kernel_launch_uid = 1
gpu_sim_cycle = 122622
gpu_sim_insn = 69526251
gpu_ipc =     566.9965
gpu_tot_sim_cycle = 122622
gpu_tot_sim_insn = 69526251
gpu_tot_ipc =     566.9965
gpu_tot_issued_cta = 0
gpu_stall_dramfull = 2621
gpu_stall_icnt2sh    = 11853
gpu_total_sim_rate=620770
```

```
GPGPU-Sim uArch: GPU detected kernel '_Z18optimizedReductionPfS_j' finished on shader 8.
kernel_name = _Z18optimizedReductionPfS_j
kernel_launch_uid = 1
gpu_sim_cycle = 47445
gpu_sim_insn = 30893717
gpu_ipc =     651.1480
gpu_tot_sim_cycle = 47445
gpu_tot_sim_insn = 30893717
gpu_tot_ipc =     651.1480
gpu_tot_issued_cta = 0
gpu_stall_dramfull = 9746
gpu_stall_icnt2sh    = 199470
gpu_total_sim_rate=572105
```

The optimized reduction kernel has **low values in gpu_sim_cycle, gpu_sim_insn, gpu_tot_sim_insn and gpu_tot_sim_cycle and gpu_total_sim_rate**
And has high values in gpu_ipc, gpu_tot_ipc, gpu_stall_dramfull, and gpu_stall_icnt2sh

There is a huge run time difference between **Naive(1m52s)** and **Optimized Reduction (54s)**
Due to the major difference in run time, **the optimized reduction has better performance**.

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

Naive:

```
Warp Occupancy Distribution:
Stall:134228    W0_Idle:59071   W0_Scoreboard:255290    W1:370283       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      W17:0W18:0
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   W32:2063424
```

Optimized:

```
Warp Occupancy Distribution:
Stall:99069     W0_Idle:50884   W0_Scoreboard:244368    W1:5862 W2:0    W3:0    W4:0    W5:0    W6:0    W7:0    W8:0
W9:0    W10:0   W11:0   W12:0   W13:0   W14:0   W15:0   W16:0   W17:0   W18:0   W19:0   W20:0   W21:0   W22:0W23:0
W24:0   W25:0   W26:0   W27:0   W28:0   W29:0   W30:0   W31:0   W32:1011195
```

Optimized Reduce has less Idle time and has stalled less.
In **optimized reduce we see it has used 1 warp and mostly 32 warps** =>

w1:5862 and w32:1011195 and 0 for other count of warps

indicating the **optimized reduce has better warp utilization** (significantly less divergence compare to the naive)

Whereas naive has used w1,w2,w4,w8,w26,w32 (high divergence)

5. Why do GPGPUs suffer from warp divergence?

   If the block size is not divisible by 32, some of the threads in the last warp don't do anything, all threads in a warp do the same thing at the same time and different warps are executed independently (except for explicit synchronization) by run-time scheduler it's possible each warp executes until it needs to wait for data (from device memory or a previous operation) then another warp has a turn, hence this causes warp divergence.

   In This particular case of SIMD, we are shifting from O(n) serialized code to O log2(n) code which leaves a lot of space for the threads to sit idle and do nothing for a long period of time and cause divergence.