De La Salle University, Manila
Computer Technology Department

Term 2, A.Y. 2022-2023

CEPARCO

GPU CUDA and Other Feature (Data Analysis): DOT PRODUCT PROGRAM USING SHARED MEMORY

GROUP 1:

Members
Cai, Edison
Susada, Stephanie Joy

June 05, 2023

Data Analysis

C++ Program	Average Execution Time (30 runs)
2^{20}	3.490 ms
2^{22}	13.282 ms
2^{24}	51.712 ms

CUDA w/o shared memory using grid-stride loop with prefetching+mem advise	256 Threads per block	512 Threads per block	1024 Threads per block
2^{20}	921.742 ms	923.635 ms	913.810 ms
2^{22}	3560 ms	3580 ms	3570 ms
2^{24}	14189 ms	14163 ms	14091 ms

CUDA with shared memory using grid-stride loop with prefetching+mem advise	256 Threads per block	512 Threads per block	1024 Threads per block
2^{20}	4.178 ms	2.138 ms	1.086 ms
2^{22}	16.595ms	8.087 ms	4.220 ms
2^{24}	64.587 ms	32.286 ms	15.986 ms

First, it can be observe from the obtained data that for small array sizes (like 2^{20}), the C++ program performs significantly better than both CUDA implementations, the CUDA without shared memory method has the highest execution time, while the CUDA with shared memory method shows an improvement but is still not better than the C++ program. Based on the 2nd table which is the CUDA without shared memory table, it can be concluded that despite of the changes in threads per block, the average execution time are still close to each other and almost have the same values which means that the thread per block in CUDA without shared memory doesn't do anything to achieve some changes in the execution time or doesn't improve the execution time of the program. Meanwhile, it can be seen that CUDA with shared memory has a faster execution time compared to the one that doesn't implement shared memory with an average of 99% decrease in the execution time. It can also be observed that as the thread per block increases the execution time decreases. As the array size remains constant, increasing the number of threads per block leads to better performance. This can be attributed to better

utilization of the GPU's resources, improved memory access patterns, and increased parallelism. The execution time decreases significantly as the number of threads per block increases, resulting in faster computations.

Overall, the analysis demonstrates that the performance of the CUDA implementation may be greatly enhanced by using shared memory, a grid-stride loop, prefetching, and memory advice. Larger arrays can be executed more quickly by increasing the number of threads per block and streamlining memory access patterns, which improves the GPU's resource usage and parallelism. The CUDA implementation can more efficiently use parallelism and improve memory access patterns with shared memory, which significantly speeds up computation.