Assignment 2

Akhil Jarodia
Student Number: 1010169899

Algorithm Choice and Justification

The algorithm implemented for the inclusive scan is a hierarchical, recursive approach leveraging CUDA's warp-level primitives and shared memory. Initially, the Kogge-Stone algorithm was considered due to its suitability for parallel prefix sum computations. However, it did not meet performance expectations, primarily because of increased synchronization overhead and inefficient use of shared memory, leading to suboptimal performance on large datasets.

Subsequently, a hierarchical scan was implemented on top of it. While this approach worked well for smaller input sizes (up to 10^6 elements), it encountered issues with larger datasets. Specifically, data was not correctly propagated to the final recursive layer, resulting in invalid outputs for large inputs.

After further investigation, the warp-level `__shfl_up_sync` intrinsic was utilized. This intrinsic enables efficient intra-warp communication without shared memory, allowing for a more scalable and performant implementation. By integrating `__shfl_up_sync` into the inclusive scan, the algorithm achieved better performance and correctness across all input sizes.

By performing computations within warps, the number of required __syncthreads() calls is minimized, reducing synchronization overhead and potential performance bottlenecks.

Design Decisions and Performance Implications

Use of Warp-Level Primitives

Implementing the inclusive scan at the warp level using the `__shfl_up_sync` for intra-warp communication. It allows threads within a warp to exchange data efficiently. This approach eliminates the need for shared memory and explicit synchronization within the warp, reducing latency and improving performance.

Shared Memory

Ensured that all allocated memory is freed. Shared memory is employed to store partial sums computed by each warp. By synchronizing threads at the block level, these partial sums are aggregated to compute the block-level inclusive scan.

Recursive Hierarchical Scan

The recursive nature of the algorithm enables handling of datasets larger than a single block otherwise the maximum length we could handle was 1024. After each block computes its inclusive scan, the block sums are recursively scanned to produce a global prefix sum. This hierarchical approach ensures scalability and maintains performance across varying input sizes.

Block and Thread Configuration

A block size of 1024 threads is selected to maximize GPU occupancy, which is the maximum size possible for our GPU. Larger block sizes can improve memory throughput.

Experiments and Observations

Kogge-Stone Algorithm Implementation

The initial implementation using the Kogge-Stone algorithm showed that, despite its suitability for parallel prefix sums, it suffered from performance issues. The algorithm's reliance on multiple synchronization points and extensive use of shared memory led to increased overhead.

Hierarchical Scan Challenges

Transitioning to a hierarchical scan improved performance for smaller datasets (up to 10^6 elements). However, for larger inputs, the algorithm failed to produce correct results. The issue was traced to data not being properly transferred to the final recursive layer. The lack of efficient inter-block communication mechanisms in the initial implementation led to incomplete propagation of partial sums.

Adoption of `__shfl_up_sync`

To address these challenges, the `__shfl_up_sync` intrinsic was incorporated into the warp-level scan. This change significantly improved intra-warp communication efficiency. The intrinsic allows threads to read values from other threads within the same warp without shared memory, reducing latency and synchronization overhead. With this modification, the inclusive scan produced correct results across all tested input sizes and showed improved performance.

Bottlenecks and Potential Optimizations

Global Memory Accesses

Global memory latency remains a bottleneck, particularly during the recursive scan of block sums. Potential optimizations include utilizing faster memory tiers.

Resource Utilization

Optimizing shared memory and register usage can enhance occupancy and performance.

Conclusion

The evolution of the inclusive scan implementation highlights the importance of selecting appropriate algorithms and leveraging hardware-specific features. Starting from the Kogge-Stone algorithm, moving through hierarchical scans, and finally integrating the `__shfl_up_sync` intrinsic, the final implementation achieves correctness and improved performance across all input sizes. Future optimizations should focus on reducing global memory latency, minimizing thread divergence, and enhancing resource utilization to further improve performance.