Sadaf Sayeed Chowdhury

Report on Parallel Radix Partition

This project involves the implementation of a GPU-based parallel radix partition. The main goal was to partition an input array of keys into several partitions based on their radix values while ensuring that keys with the same radix value are grouped together. The implementation uses three kernels: histogram, prefix scan, and reorder, each designed to work in parallel for optimized performance on the GPU.

The 'histogram' kernel calculates a histogram of the input keys based on their radix values. Every kernel thread uses the bfe() function to extract a bit field from the key after processing an element from the input array. The partition index for the key is found using the extracted bit field. The thread then updates the corresponding histogram bin using atomicAdd() to ensure thread safety. The overhead of frequent global memory accesses is decreased by using shared memory to store the histogram for every block. The histogram is stored to global memory once every thread in the block has finished processing its data.

The 'prefixScan' kernel performs a prefix sum operation on the histogram to compute the offsets for each partition. Each thread processes one partition, and shared memory is used to store partial prefix sums. In order to minimize thread divergence and guarantee correct results, the kernel uses a parallel scan approach in which threads collaboratively calculate the prefix sum within each block. The offsets required for the next ordering step are provided by writing the calculated prefix sums to global memory after the scan is finished.

The 'Reorder' kernel reorders the keys based on the computed offsets from the prefix scan. One element from the input array is processed by each thread, and the target partition is determined by extracting the radix value. The offset for its key in the output array of the relevant partition is updated by the thread using atomic operations, guaranteeing that keys are arranged correctly and without race conditions. The keys in the final rearranged array are grouped based on their radix values which are determined by the prefix sum offsets and then written back to global memory.

To improve the program's performance, several optimizations were implemented. The histogram and prefix scan kernels made use of shared memory to minimize global memory accesses which greatly increased execution speed. By storing histograms in shared memory, processing the input array in parallel at the block level, and employing atomic operations to safely update the histogram and prefix sums, efficient memory access patterns were created. Correct results were guaranteed following updates to shared memory and before writing to global memory thanks to effective synchronization utilizing __syncthreads(). For the prefix scan kernel, the number of block was calculated using the number of partitions instead of rSize which also helped increase the performance.

The performance of the program improved significantly after optimizations were implemented. When tested with 1000000 elements and 512 partitions, the total running

time of all kernels before optimization was 1.67 ms, but after optimizing it went down to 1.19 ms which is a 29% reduction in execution time.