**Applications of parallel scan:**

1. Array Summation: Calculating the prefix sums of an array for use in algorithms like histogram computation and quicksort partitioning.

2. Stream Compaction: Removing invalid or null elements from a dataset while preserving the order of valid elements.

3. Radix Sort: Used in building key distributions for sorting algorithms by summing the counts of digits in each bin.

**Number of iterations in the reduction step:**

The reduction step performs a tree-like summation, where each thread adds data from two elements based on a stride that doubles in each iteration. For an array of size N :

- Iterations: log(N), at each step, the stride doubles ( 1, 2, 4, … , N/2 ) until the entire array is reduced, leading to log(N) iterations.

**Floating operations in the reduction step:**

Each thread performs one addition per stride.

- Operations: (N/2) + (N/4) + … + 1 = N - 1 additions.

- Explanation: The total number of additions performed across all threads at all strides sums to N - 1, as this is the number of pairwise additions needed to reduce N elements.

**Number of iterations in the post-reduction reverse step:**

The reverse step propagates partial sums back down the tree.

Iterations: log(N), similar to the reduction step, each thread updates values based on a stride that doubles in each iteration, requiring log(N) iterations.

**Floating operations in the post-reduction reverse step:**

Each thread performs one addition per stride in the reverse step.

- Operations: (N/2) + (N/4) + … + 1 = N - 1 additions.

- Explanation: Similar to the reduction step, the number of additions required to propagate results back down sums to N - 1.

**Global memory reads:**

The kernel reads the input array once and may also read intermediate results for each block's sum.

- Reads: N + B, where B is the number of blocks.

- Explanation: Each element is read once during the initial loading into shared memory, and block sums are read for further processing.

**Global memory writes:**

The kernel writes the final scan result for all elements and the auxiliary block sums.

- Writes: N + B, where B is the number of blocks.

- Explanation: Each output element is written once, and block sums are stored in the auxiliary array.

**Optimizations in the work-efficient scan kernel:**

1. Shared Memory Usage:

- The kernel minimizes global memory access by using shared memory for intermediate calculations.

- Reduces the latency associated with global memory operations.

2. Two-Phase Process:

- First, scans within blocks are computed, and block sums are stored.

- Second, block sums are combined and propagated efficiently.

3. Parallel Reduction:

- The reduction and reverse steps are optimized to avoid redundant computations and ensure balanced workloads among threads.

4. Fewer Global Memory Accesses:

- Global memory is accessed only for initial reads and final writes, reducing overall memory bandwidth usage.