PARALLEL REDUCTION

Parallel reduction

- common optimization technique in parallel computing, often used to efficiently compute the sum, minimum, maximum, or other associative operations of an array of values.
- The basic idea behind parallel reduction is to divide the input array into smaller chunks, process those chunks in parallel, and then combine the results to obtain the final result.
- This is done iteratively until you have a single value that represents the reduction result.

Parallel reduction

- Divide Input Data: Divide the input array into blocks of threads. Each thread block will be responsible for reducing a portion of the input data.
- Load Data: Each thread loads its respective data into shared memory, which is faster to access compared to global memory. Shared memory is a small, low-latency memory space shared among threads within a thread block.
- Perform Parallel Reduction: Perform a parallel reduction within each thread block using an algorithm like tree-based reduction or warp-based reduction. These algorithms take advantage of the parallelism offered by the GPU's architecture to efficiently reduce the data.

Parallel reduction

- Combine Block Results: After each thread block completes its reduction, one thread in each block writes the block's reduced result to a separate global memory array.

- Perform Final Reduction: Finally, you perform another reduction step on the global memory array that contains the results from each block. This can be done using the same parallel reduction technique, recursively, until you obtain a single value.

Parallel reduction

```
__global__ void parallelSum(float* input, float* output, int N) {
    __shared__ float sharedData[THREADS_PER_BLOCK];

    int tid = threadIdx.x;
    int blockId = blockIdx.x;
    int globalId = blockDim.x * blockId + tid;
```

Parallel reduction

```
if (globalId < N) {
        sharedData[tid] = input[globalId];
   } else {
        sharedData[tid] = 0.0f;
   }
   __syncthreads();

   // Perform parallel reduction within the block
```

Parallel reduction

```
for (int stride = blockDim.x / 2; stride > 0; stride >>= 1)
 {
       if (tid < stride) {
           sharedData[tid] += sharedData[tid + stride];
       }
       __syncthreads();
   }

   // Write the block's reduced result to global memory
   if (tid == 0) {
       output[blockId] = sharedData[0];
   }
}
```