Lecture No.4: Tree-based Reduction

Muhammad Osama Mahmoud, TA

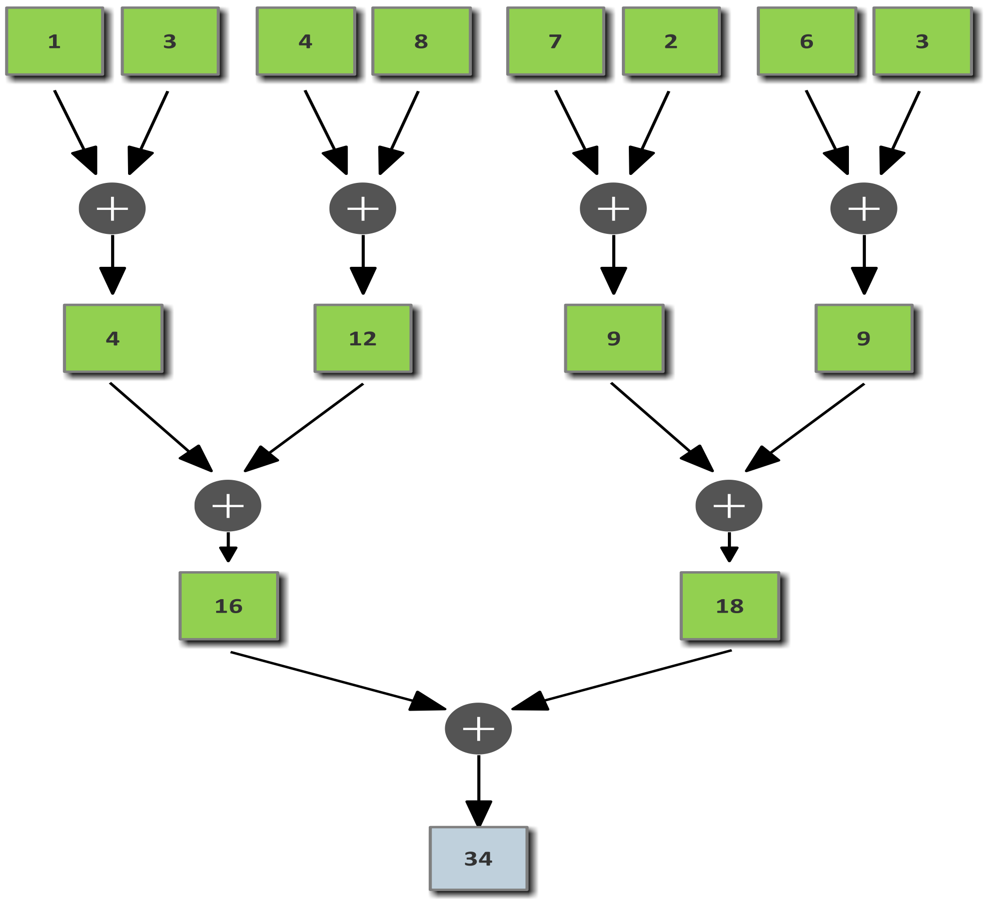
Computer and Systems Eng. Dept. - Faculty of Engineering – University of Minia

What is Reduction ?

- Reduce a set of input data values into a single value using some reduction operation
  - Sum
  - Min
  - Max
  - Product
- May be applied to dot-product as we see later
- The sequential version has a complexity of O(N), recursive over N input data and perform the reduction operation N times

Tree-based reduction algorithm

- Assume a sum operation



Basic Parallel Reduction

- Assume the sum operation
- Uses the interleaved addressing
- Each thread adds two consecutive elements within the shared memory
- Recursively halve the number of threads after each step
- Repeat the reduction operation on the resulting data
- Add the final partial results on the host side

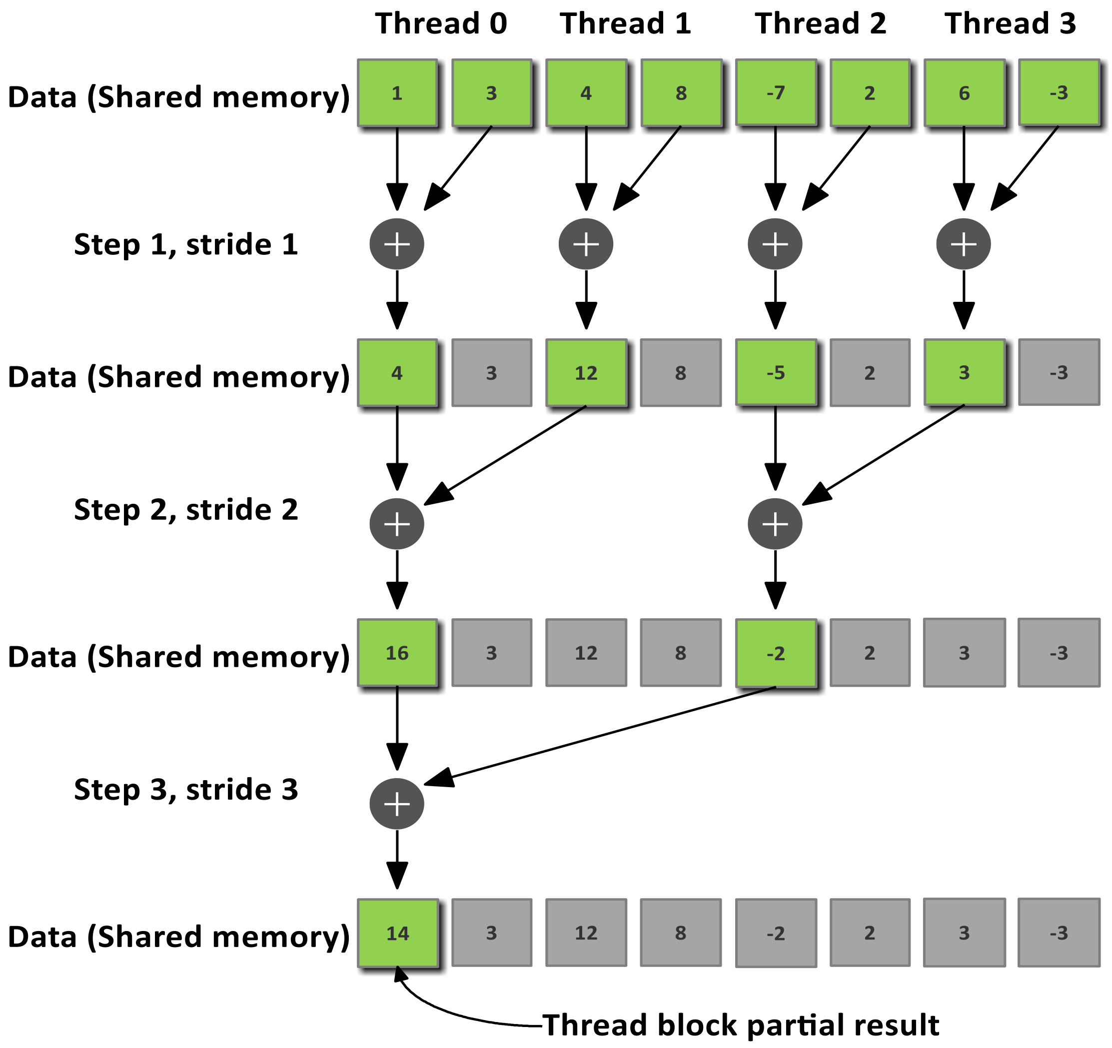
Basic Reduction Kernel

```
__global__ void reduce_v1(int * inp_data, int * outp_data) {
    extern  __shared__  int  sh_data[ ]; //dynamically locate shared memory at kernel launch
    unsigned int  tx   = threadIdx.x;
    unsigned int  idx = blockIdx.x * blockDim.x  +  threadIdx.x;
    sh_data[tx]  =  inp_data[idx];
    __syncthreads();
    // do reduction in shared memory
    for(unsigned int stride = 1; stride < blockDim.x; stride <<= 1) {
        int index = 2 * stride * tx;
        if ((index)  <  blockDim.x) {
            sh_data[index] += sh_data[index + stride];
        }
        __syncthreads();
    }
    // make thread 0 write result for each block to global memory
    if (tx == 0) outp_data[blockIdx.x] = sh_data[tx];
}
```

Basic Reduction Kernel Execution

- Input data of 8 values
- Single thread block
- Block = 8 threads
- Shared memory =

  8 threads * sizeof(int)



Basic Reduction Kernel Execution (Cont.)

- Eliminates the problem of thread-divergence while reducing the number of threads at each stride
- The interleaved addressing has a disadvantage of shared memory bank conflicts
- A bank conflict arises when a halfwarp tries to load/store from/to the same memory bank (the access will be serialized)
- Solution, use the sequential addressing instead to avoid any bank conflicts (each thread in halfwarp access consecutive 32-bit words)
- A barrier synchronization required to make sure each value in shared memory is updated , that's why __syncthreads() is used

Reduction Kernel with sequential addressing

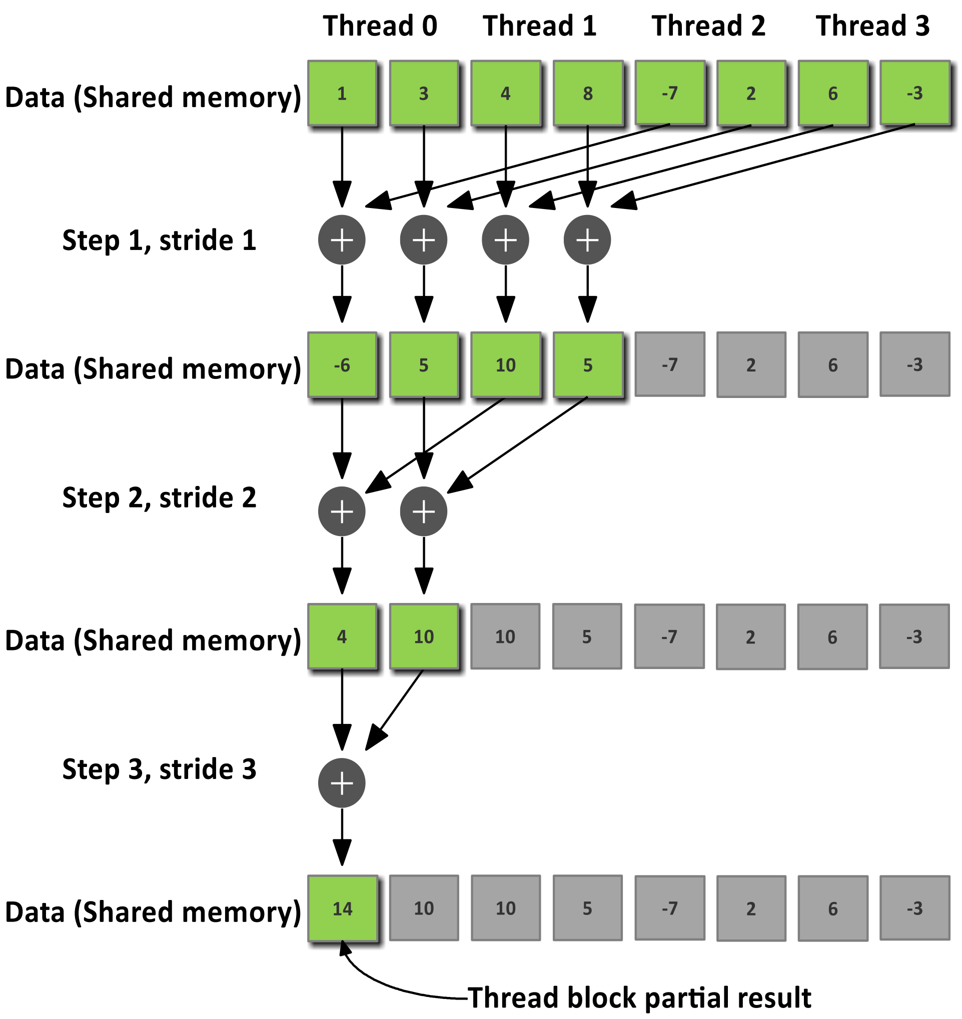
- Just replace the reduction step in the basic kernel

```
for (unsigned int stride = 1; stride < blockDim.x; stride
<<= 1) {
        int index = 2 * stride * tx;
        if ((index)  <  blockDim.x) {
                sh_data[index] += sh_data[index + stride];
        }
        __syncthreads();
}
```

With a reversed loop and threadID-based indexing:

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

Reduction Kernel with sequential addressing Execution



References

[1] Wen-mei W. Hwu, “Heterogeneous Parallel Programming”. Online course, 2014. Available: https://class.coursera.org/hetero-002

[2] M. Harris, “Optimizing Parallel Reduction in CUDA”, Oct. 2007.