GPU Teaching Kit

Accelerated Computing

Module 10.2 – Parallel Computation Patterns (scan)

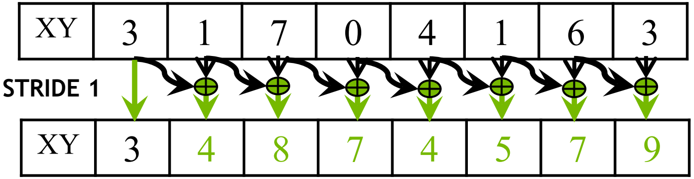
Lecture 10.2 - A Work-inefficient Scan Kernel

Objective

- To learn to write and analyze a high-performance scan kernel
  - Interleaved reduction trees
  - Thread index to data mapping
  - Barrier Synchronization
  - Work efficiency analysis

A Better Parallel Scan Algorithm

1. Read input from device global memory to shared memory
2. Iterate log(n) times; stride from 1 to n-1: double stride each iteration



XY	3	1	7	0	4	1	6	3
XY	3	4	8	7	4	5	7	9

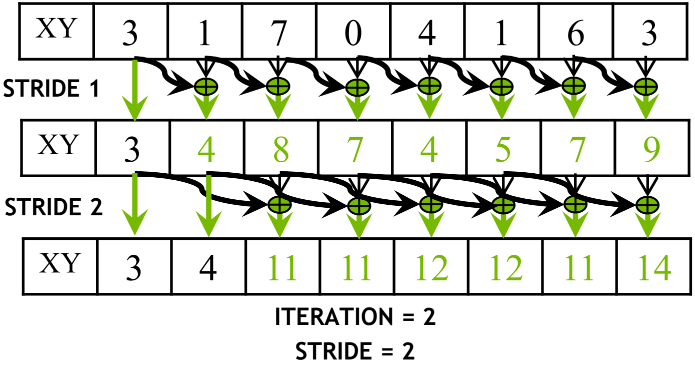
ITERATION = 1

STRIDE = 1

- Active threads *stride* to n-1 (n-stride threads)
- Thread *j* adds elements *j* and *j-stride* from shared memory and writes result into element j in shared memory
- Requires barrier synchronization, once before read and once before write

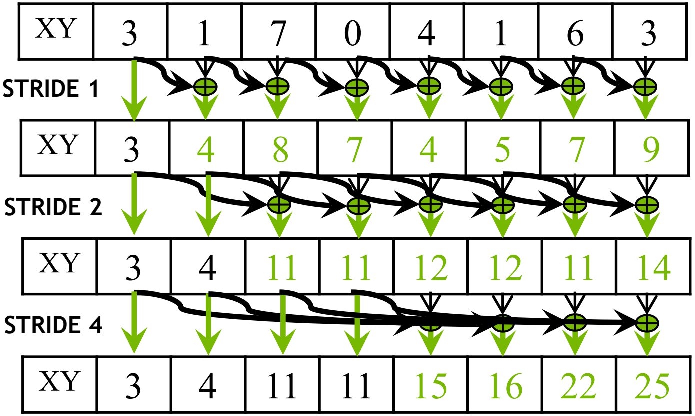
A Better Parallel Scan Algorithm

1. Read input from device to shared memory
2. Iterate log(n) times; stride from 1 to n-1: double stride each iteration.



A Better Parallel Scan Algorithm

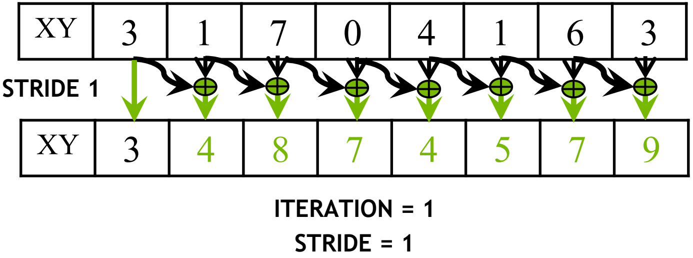
1. Read input from device to shared memory
2. Iterate log(n) times; stride from 1 to n-1: double stride each iteration
3. Write output from shared memory to device memory



ITERATION = 3
STRIDE = 4

Handling Dependencies

- During every iteration, each thread can overwrite the input of another thread
  - Barrier synchronization to ensure all inputs have been properly generated
  - All threads secure input operand that can be overwritten by another thread
  - Barrier synchronization is required to ensure that all threads have secured their inputs
  - All threads perform addition and write output



ITERATION = 1
STRIDE = 1

A Work-Inefficient Scan Kernel

```
__global__ void work_inefficient_scan_kernel(float *X, float *Y, int InputSize) {
    __shared__ float XY[SECTION_SIZE];
    int i = blockIdx.x * blockDim.x + threadIdx.x;
    if (i < InputSize) {XY[threadIdx.x] = X[i]; }
        // the code below performs iterative scan on XY
        for (unsigned int stride = 1; stride < blockDim.x; stride *= 2) {
        __syncthreads();
            float in1;
            if (stride <= threadIdx.x) {
                in1 = XY[threadIdx.x - stride]; }
            __syncthreads();
            XY[threadIdx.x] += in1;
        }
      __ syncthreads();
      If (stride <= threadIdx.x) {
          XY[threadIdx.x] += in1; }}
```

Work Efficiency Considerations

- This Scan executes log(n) parallel iterations
  - The iterations do (n-1), (n-2), (n-4),..(n- n/2) adds each
  - Total adds: n * log(n) - (n-1) → O(n*log(n)) work

- This scan algorithm is not work efficient
  - Sequential scan algorithm does *n* adds
  - A factor of log(n) can hurt: 10x for 1024 elements!

- A parallel algorithm can be slower than a sequential one when execution resources are saturated from low work efficiency

GPU Teaching Kit