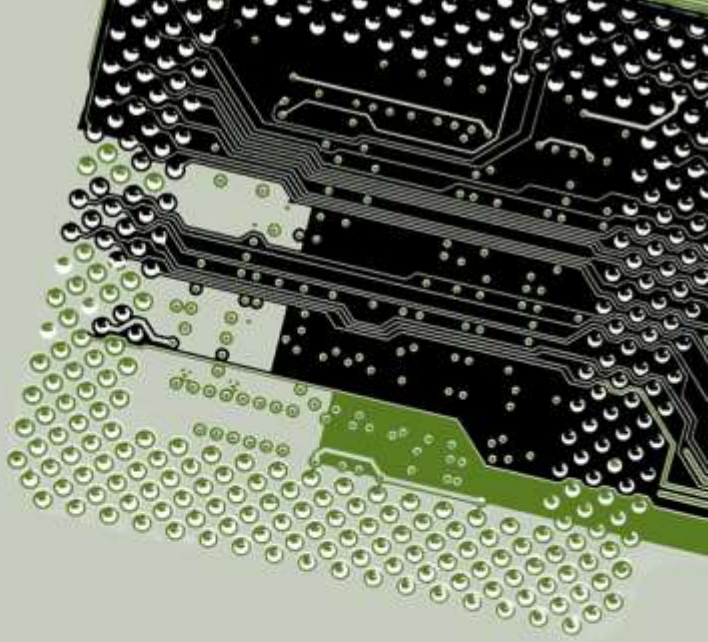
Heterogeneous Parallel Programming



Lecture 4.5

Parallel Computation Patterns

A Work-inefficient Scan Kernel

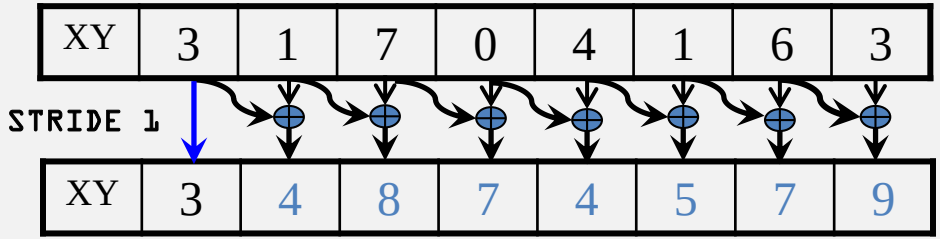
Wen-mei Hwu - University of Illinois at Urbana-Champaign

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



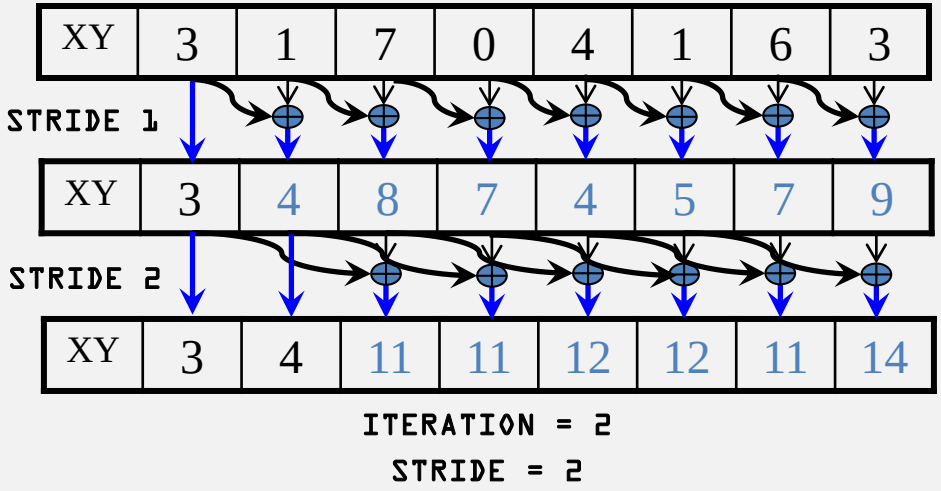
ITERATION = 1

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

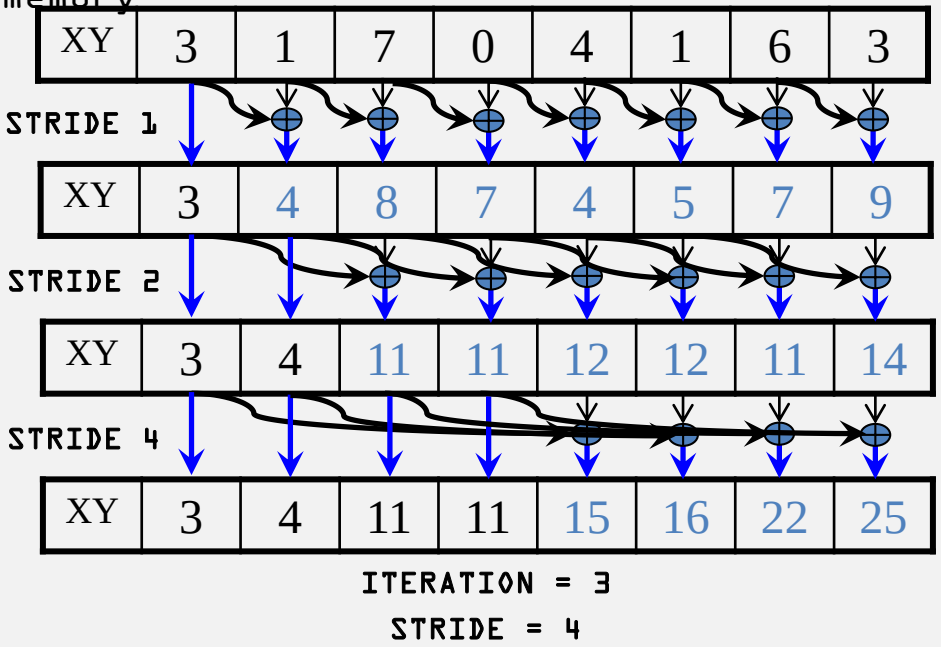
A Better Parallel Scan Algorithm

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



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 to ensure that all threads have secured their inputs
  - All threads perform Addition and write output

A Work-Inefficient Scan Kernel

```
1. __global__ void
   work_inefficient_scan_kernel(float *X, float
   *Y, int InputSize) {
2.       __shared__ float XY[SECTION_SIZE];
3.       int i = blockIdx.x*blockDim.x +
threadIdx.x;
4.       if (i < InputSize) {XY[threadIdx.x] =
X[i];}
          // the code below performs iterative
scan on XY
5.       for (unsigned int stride = 1; stride <=
threadIdx.x; stride *= 2) {
6.             __syncthreads();
7.             float in1 = XY[threadIdx.x-
stride];
8.             __syncthreads();
9.             XY[threadIdx.x] += in1;
10.      }
```

Work Efficiency Considerations

- This Scan executes log(n) parallel iterations
  - The steps 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

To learn more, read
Sections 9.2-9.3