Lecture 4.6

Parallel Computation Patterns
A Work-Efficient Parallel Scan Kernel

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

- To learn to write a work-efficient scan kernel
  - Two-phased balanced tree traversal
  - Aggressive reuse of computation results
  - Reducing control divergence with more complex thread index to data index mapping
Improving Efficiency

• **Balanced Trees**
  • Form a balanced binary tree on the input data and sweep it to and from the root
  • Tree is not an actual data structure, but a concept to determine what each thread does at each step

• **For scan:**
  • Traverse down from leaves to root building partial sums at internal nodes in the tree
    • Root holds sum of all leaves
  • Traverse back up the tree building the output from the partial sums
Parallel Scan – Reduction Phase

Time

\[ \sum_{x_0}^{x_1} \]

\[ \sum_{x_2}^{x_3} \]

\[ \sum_{x_4}^{x_5} \]

\[ \sum_{x_6}^{x_7} \]

In-place calculation

Value after reduce
Reduction Phase Kernel Code

// XY[2*BLOCK_SIZE] is in shared memory

for (int stride = 1; stride <= BLOCK_SIZE; stride *= 2) {
    int index = (threadIdx.x+1)*stride*2 - 1;
    if(index < 2*BLOCK_SIZE)
        XY[index] += XY[index-stride];
    __syncthreads();
}

threadIdx.x+1 = 1, 2, 3, 4, ...
strides = 1, 2, 4, 8, ...
index = 1, 3, 5, 7, ...
Parallel Scan – Post Reduction Reverse Phase

Move (add) a critical value to a central location where it is needed.
Parallel Scan - Post Reduction Reverse Phase
Putting it together
Post Reduction Reverse Phase Kernel Code

for (int stride = BLOCK_SIZE/2; stride > 0; 
    stride /= 2) {
    __syncthreads();
    int index = (threadIdx.x+1)*stride*2 - 1;
    if(index+stride < 2*BLOCK_SIZE) {
        XY[index + stride] += XY[index];
    }
}

__syncthreads();
if (i < InputSize) Y[i] = XY[threadIdx.x];

First iteration for 16-element section
threadIdx.x = 0
stride = BLOCK_SIZE/2 = 8/2 = 4
index = 8-1 = 7
To learn more, read Sections 9.4-9.5