Lecture 4.7
Parallel Computation Patterns
More on Parallel Scan

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

- To learn more about parallel scan
- Analysis of the work efficient kernel
- Exclusive scan
- Handling large input vectors
Work Analysis of the Work Efficient Kernel

- The work efficient kernel executes $\log(n)$ parallel iterations in the reduction step
  - The iterations do $n/2, n/4, \ldots, 1$ adds
  - Total adds: $(n-1) \rightarrow O(n)$ work

- It executes $\log(n)-1$ parallel iterations in the post reduction reverse step
  - The iterations do $2-1, 4-1, \ldots, n/2-1$ adds
  - Total adds: $(n-2) - (\log(n)-1) \rightarrow O(n)$ work

- Both phases perform up to no more than $2*(n-1)$ adds

- The total number of adds is no more than twice of that done in the efficient sequential algorithm
  - The benefit of parallelism can easily overcome the 2X work when there is sufficient hardware
Some Tradeoffs

- The work efficient scan kernel is normally more desirable
  - Better Energy efficiency
  - Less execution resource requirement
- However, the work inefficient kernel could be better for absolute performance due to its single-step nature if
  - There is sufficient execution resource
Definition: The exclusive scan operation takes a binary associative operator $\oplus$, and an array of $n$ elements $[x_0, x_1, \ldots, x_{n-1}]$ and returns the array $[0, x_0, (x_0 \oplus x_1), \ldots, (x_0 \oplus x_1 \oplus \ldots \oplus x_{n-2})]$.

Example: If $\oplus$ is addition, then the all-prefix-sums operation on the array $[3, 1, 7, 0, 4, 1, 6, 3]$, would return $[0, 3, 4, 11, 11, 15, 16, 22]$. 
Why Exclusive Scan

- To find the beginning address of allocated buffers

- Inclusive and exclusive scans can be easily derived from each other; it is a matter of convenience

Exclusive: [0 3 4 11 11 15 16 22]
Inclusive: [3 4 11 11 15 16 22 25]
A simple exclusive scan kernel

- Adapt an inclusive, work in-efficient scan kernel
- Block 0:
  - Thread 0 loads 0 into XY[0]
  - Other threads load X[threadIdx.x-1] into XY[threadIdx.x]
- All other blocks:
  - All thread load X[blockIdx.x*blockDim.x+threadIdx.x-1] into XY[threadIdx.x]
- Similar adaption for work efficient scan kernel but pay attention that each thread loads two elements
  - Only one zero should be loaded
  - All elements should be shifted by only one position

Read the Harris article for a more intellectually interesting approach to exclusive scan kernel implementation.
Handling large Input Vectors

- Build on the work efficient scan kernel
- Have each section of 2*blockDim.x elements assigned to a block
- Have each block write the sum of its section into a Sum[] array indexed by blockIdx.x
- Run the scan kernel on the Sum[] array
- Add the scanned Sum[] array values to the elements of corresponding sections

- Adaptation of work inefficient kernel is similar.
Overall Flow of Complete Scan

- Initial Array of Arbitrary Values
  - Scan Block 0
  - Scan Block 1
  - Scan Block 2
  - Scan Block 3
- Store Block Sum to Auxiliary Array
- Scan Block Sums
- Add Scanned Block Sum $i$ to All Values of Scanned Block $i + 1$
- Final Array of Scanned Values
To learn more, read Sections 9.4-9.5