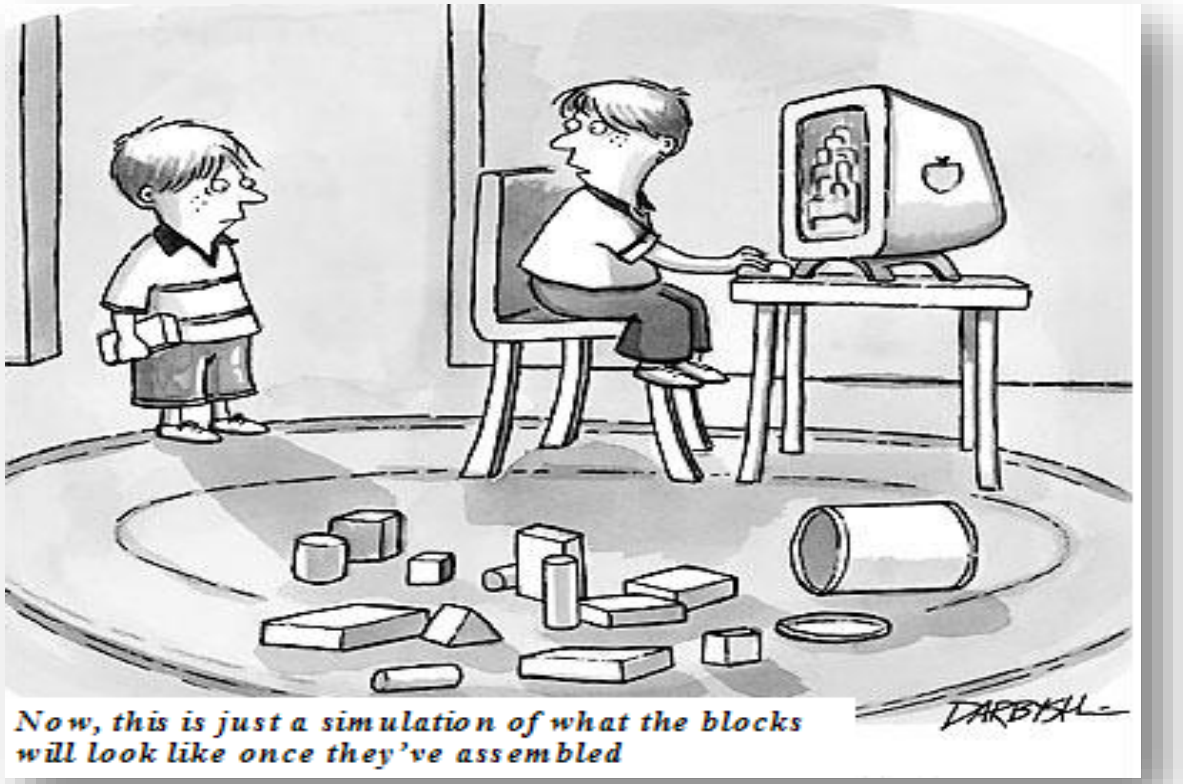
ECE569
Module 37



- Scan

Scan

- **Many parallel applications**
  - Radix sort, Quicksort, String comparison, Data compression
  - Resource management, Lexical analysis, Stream compaction,
  - Tree operations

- **Just like reduce:**
  - Binary associative operator (sum, min, max, etc)
  - Identify element
  - Input array

Scan: Running Sum

- **Input:**

[3 1 7 0 4 1 6 3]

- **Output**

[3 4 11 11 15 16 22 25]

Scan: Running Sum

- Input: [3 1 7 0 4 1 6 3]
- Output: [3 4 11 11 15 16 22 25]

Given a sequence $[x_0, x_1, x_2, \ldots]$
Calculate output $[y_0, y_1, y_2, \ldots]$
Such that
$y_0 = x_0$
$y_1 = x_0 + x_1$
$y_2 = x_0 + x_1 + x_2$
recursive definition: $y_i = y_{i-1} + x_i$

```
out[0] = x[0];
for(i=1;i<n;i++)
    out[i] = out[i-1] + x[i];
```

- Work complexity =
- Step complexity =

Scan

- Scan is a very useful primitive for parallelization
  - Scan is highly parallelizable and
  - Many applications with iteration dependence can be expressed as a scan problem
    - An application that is poor fit for GPU becomes very suitable when formulated as scan.

Naïve parallel scan (Version 1)

Input: [x_0, x_1, x_2, ...], Calculate output: [y_0, y_1, y_2, ...], where:

$y_0 = x_0$

$y_1 = x_0 + x_1$

$y_2 = x_0 + x_1 + x_2$

recursive definition: $y_i = y_{i-1} + x_i$

- Assign one thread to calculate each y element
- Have every thread to add up all x elements needed for the y element

- What is the step complexity?

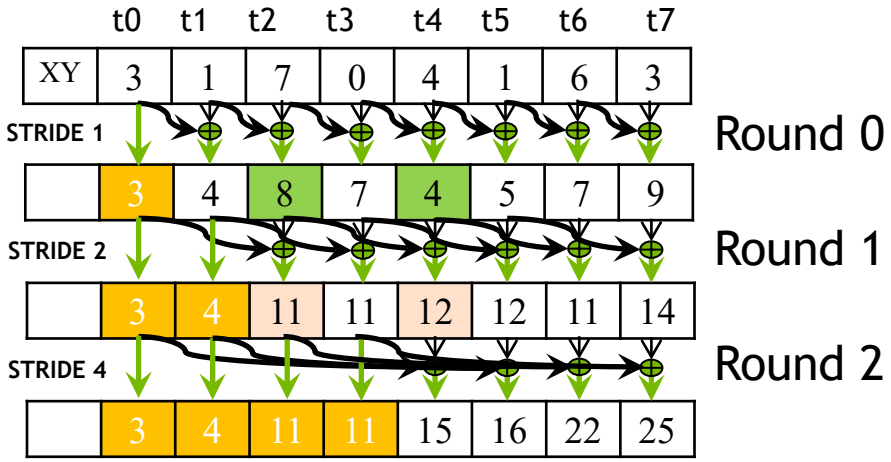
- What is the work complexity?

Scan: Running Sum – Version 2

t0	t1	t2	t3	t4	t5	t6	t7
3	**1**	**7**	**0**	**4**	**1**	**6**	**3**

Notes on Version 2

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



Version 2 Implementation

```
__global__ void my_kernel(float *X, float *Y, int InputSize) {
  __shared__ float XY[SECTION_SIZE];
int i = = blockIdx.x * blockDim.x + threadIdx.x;
int tid = threadIdx.x;
// Read from global to shared memory
If (_____________________________)
   XY[_________________] = X[_________________]
// Write the for loop
for (int stride=_________;stride___________;stride=________){




}
// Write back to global memory
If (_____________________________)
  Y[_____________] = XY{_____________________];
}
```