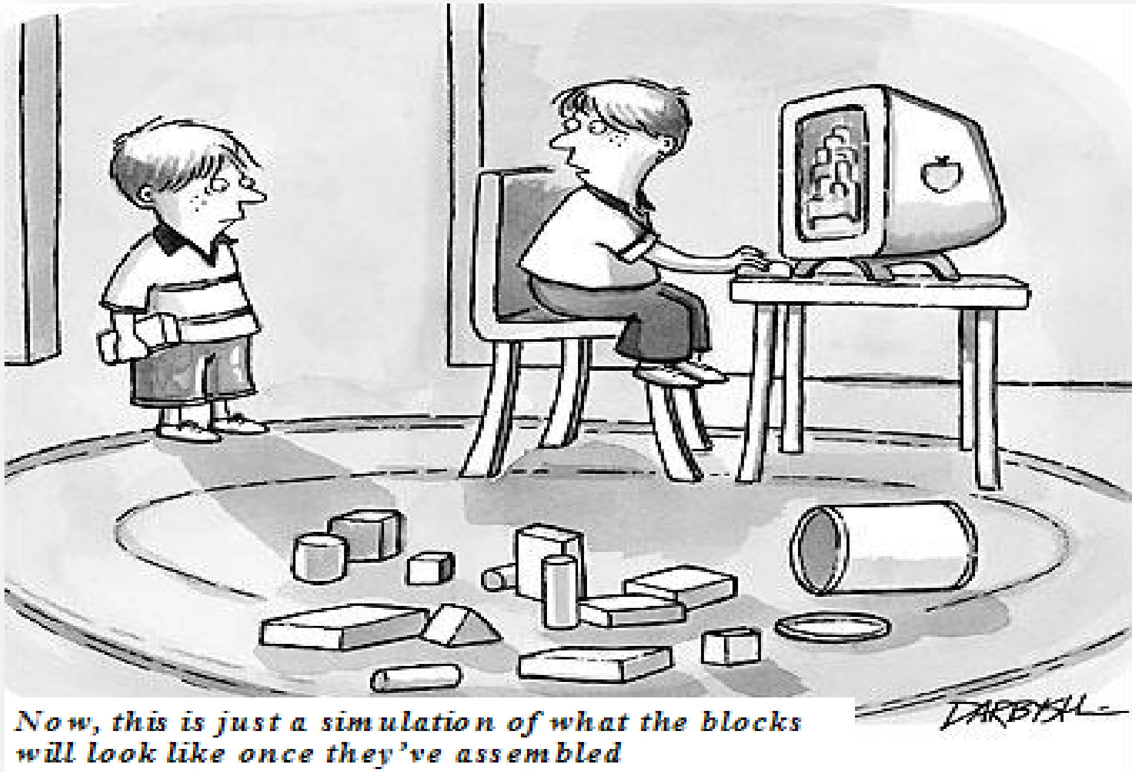
ECE569
Module 38



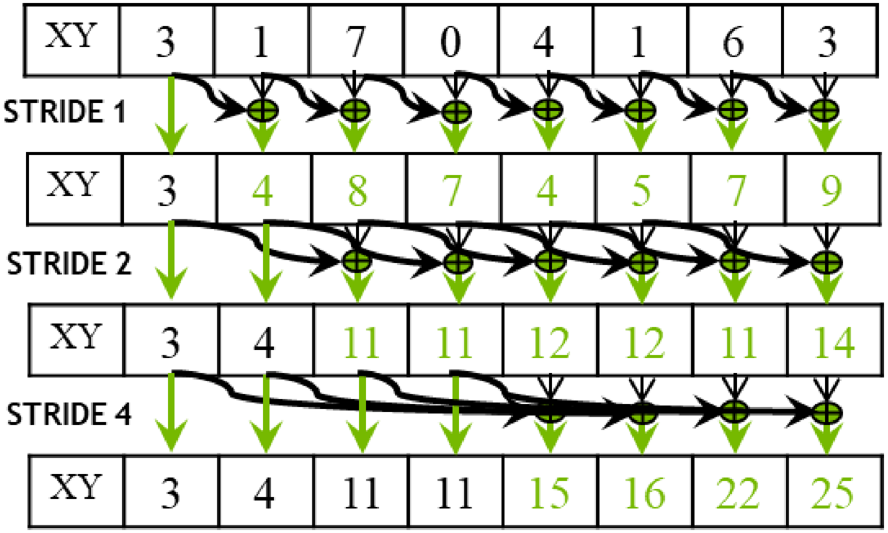
- Scan – Version 2 Implementation and Analysis

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{__________________];
}
```

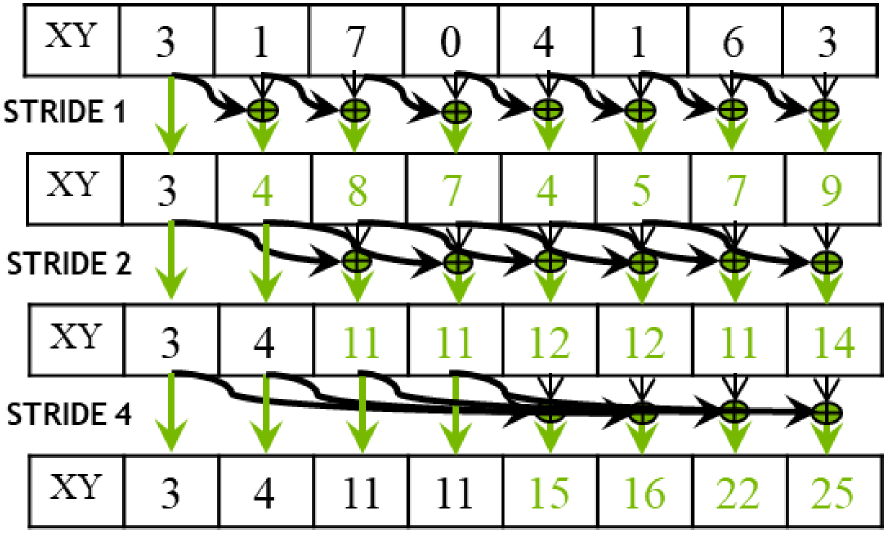
Analysis of parallel version-2

- **Running Sum of N elements**
  – What is the step complexity?



Analysis of parallel version-2

- **Running Sum of N elements**
  - What is the work complexity?



Evaluation

	Step Complexity	Work Complexity
Version 1 (Naïve)	**N**	**O(N^2)**
Version 2	**logN**	**NlogN**

Wait a minute: my serial code has better work complexity

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

O(N)!

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

Next: We can improve the work efficiency

- As we know, the fastest parallel way to produce sum values for a set of values is a reduction tree.
- With sufficient execution units, a reduction tree can generate the sum for N values in logN time units.
- Solution:
  - What if we do a reduction to find the sum (REDUCE)
  - And then use intermediate sums to use as critical piece of information to find the other running sums (CONSTRUCT)

Reading

- **Mark Harris, Parallel Prefix Sum with CUDA**
  - https://developer.nvidia.com/gpugems/GPUGems3/gpugems3_ch39.html