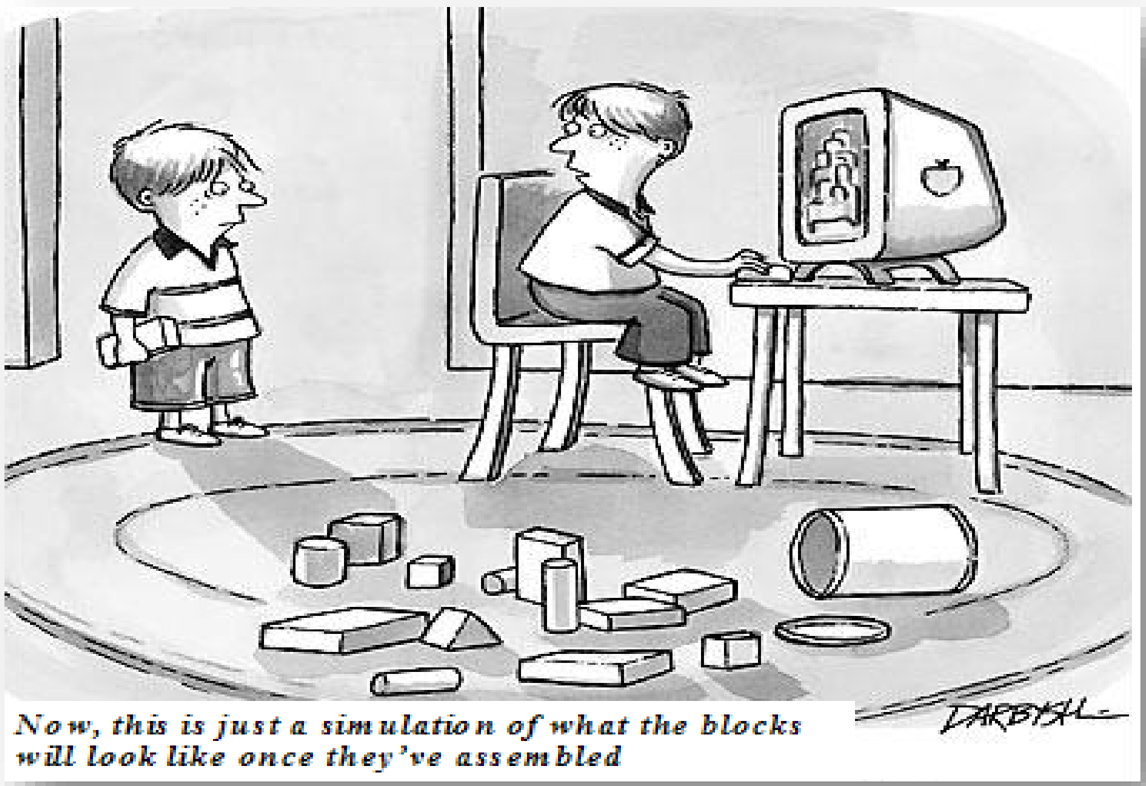
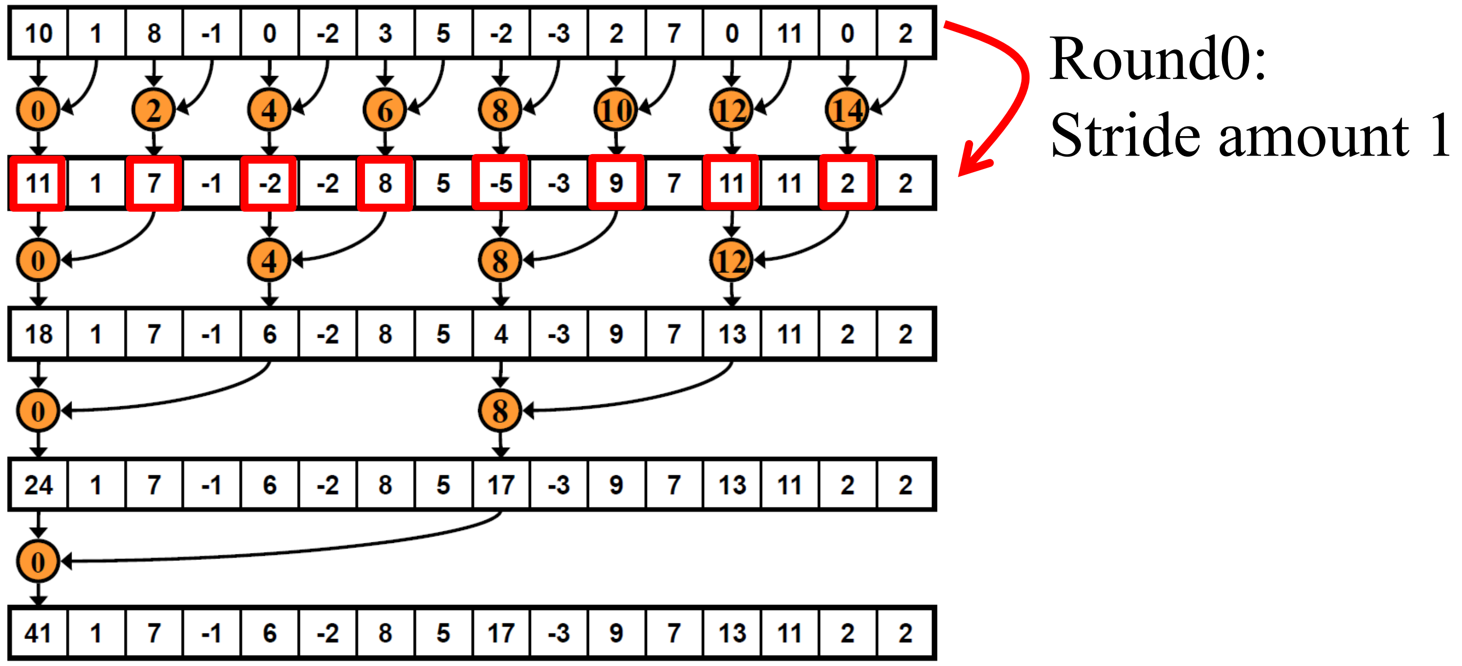
ECE569
Module 34


Now, this is just a simulation of what the blocks will look like once they've assembled
DARBYSH

- Reduction: Stride Pattern – Global and Shared Memory

Kernel: Global Memory – Stride Pattern –Round 0

```
__global__ void global_reduce_stride(float* d_out,float* d_in){
  int myId = threadIdx.x + blockDim.x * blockIdx.x;
  int tid  = threadIdx.x;
    // do reduction in global mem
```

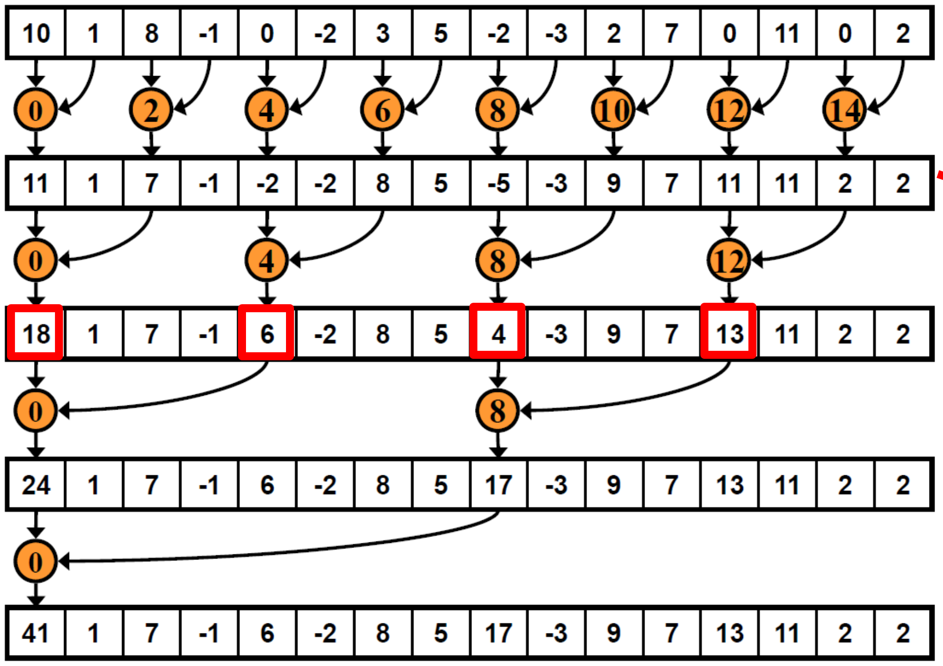


```
// thread 0 writes result for this block back to global mem


}
```

Kernel: Global Memory – Stride Pattern –Round 1

```
__global__ void global_reduce_stride(float* d_out,float* d_in){
  int myId = threadIdx.x + blockDim.x * blockIdx.x;
  int tid  = threadIdx.x;
    // do reduction in global mem
```



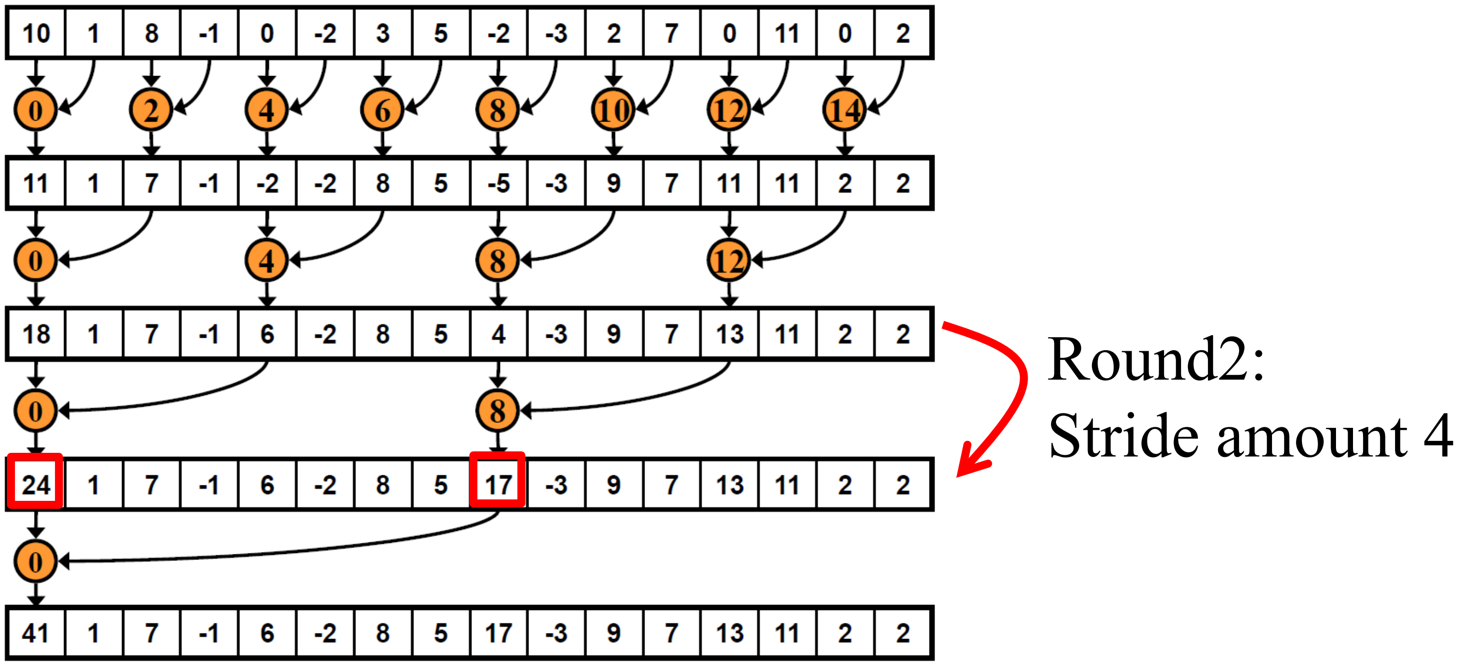
Round1:
Stride amount 2

```
// thread 0 writes result for this block back to global mem


}
```

Kernel: Global Memory – Stride Pattern –Round 2

```
__global__ void global_reduce_stride(float* d_out,float* d_in){
  int myId = threadIdx.x + blockDim.x * blockIdx.x;
  int tid  = threadIdx.x;
    // do reduction in global mem
```

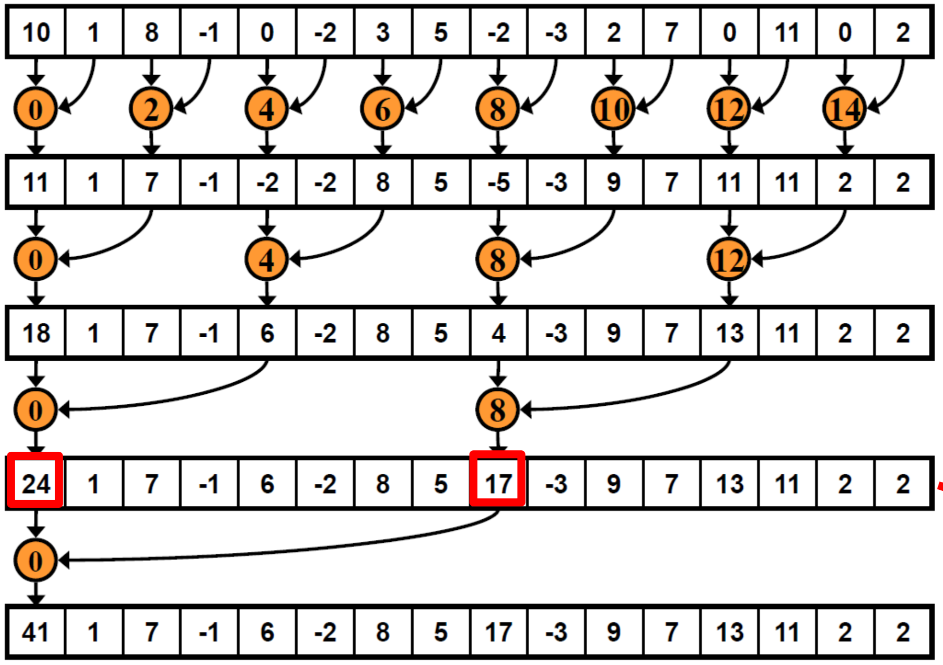


```
// thread 0 writes result for this block back to global mem


}
```

Kernel: Global Memory – Stride Pattern– Round 3

```
__global__ void global_reduce_stride(float* d_out,float* d_in){
  int myId = threadIdx.x + blockDim.x * blockIdx.x;
  int tid  = threadIdx.x;
    // do reduction in global mem
```



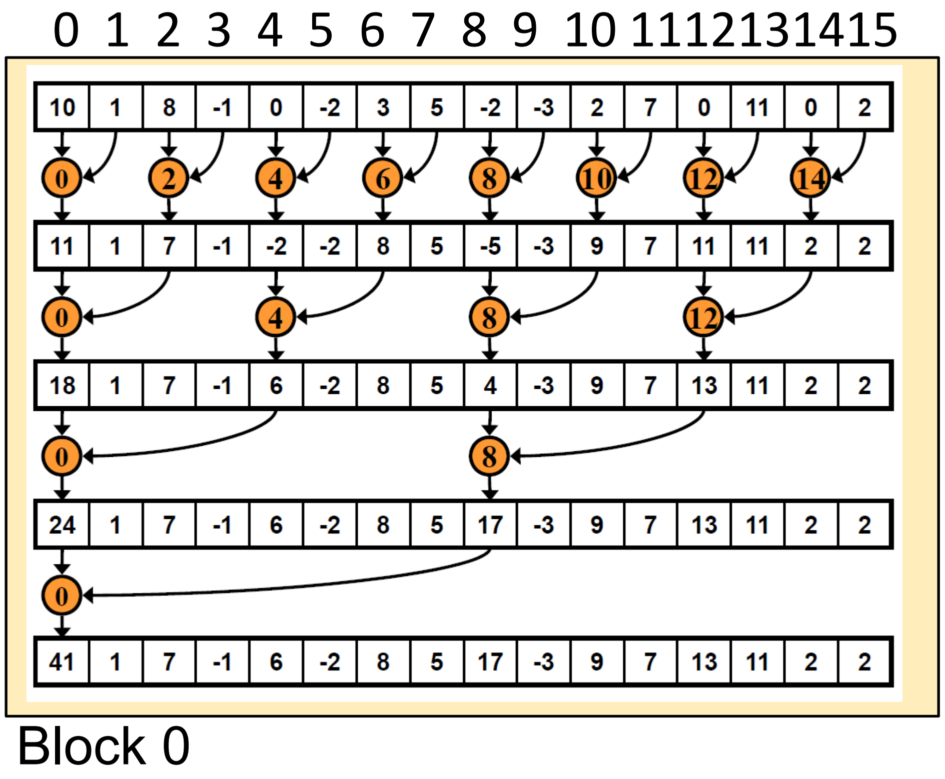
Round3:
Stride amount 8

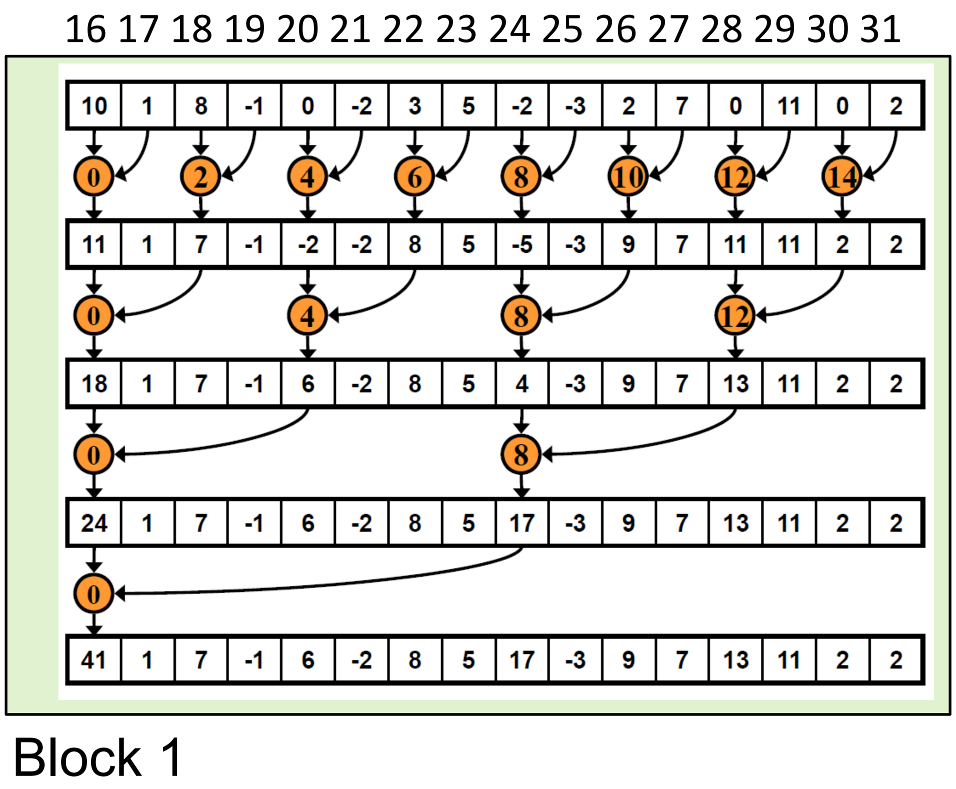
```
// thread 0 writes result for this block back to global mem


}
```

Kernel: Global Memory – Stride Pattern

```
__global__ void global_reduce_stride(float* d_out,float* d_in){
  int myId = threadIdx.x + blockDim.x * blockIdx.x;
  int tid  = threadIdx.x;
```





Kernel: Global Memory – Stride Pattern

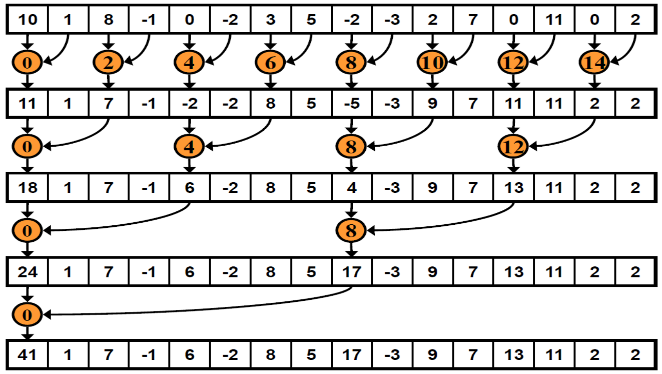
```
__global__ void global_reduce_stride(float* d_out,float* d_in){
  int myId = threadIdx.x + blockDim.x * blockIdx.x;
  int tid  = threadIdx.x;
  // do reduction in global mem
  for(int stride =______; stride <________; stride =________){

    if(___________________________) {



      d_in[__________] += d_in[__________________]; }
   }


// thread 0 writes result for this block back to global mem
 if (              ) {
        d_out[__________] = d_in[_________]; }
}
```



Kernel: Global Memory – Strided Access

```
__global__ void global_reduce_stride(float* d_out,float* d_in){
  int myId = threadIdx.x + blockDim.x * blockIdx.x;
  int tid  = threadIdx.x;
  // do reduction in global mem
  for(int stride = 1; stride < blockDim.x; stride *= 2)
  {
    __syncthreads();
    if(myId % (2*stride) == 0) {
      d_in[myId] += d_in[myId+stride]; }
   }

// thread 0 writes result for this block back to global mem
 if (tid == 0) {
        d_out[blockIdx.x] = d_in[myId]; }
}
```

Reduction - Tesla P100;compute v6.0;

n: 1<<20

Version	Time (ms)
serial	3.27400
global reduce stride – naïve	0.16450

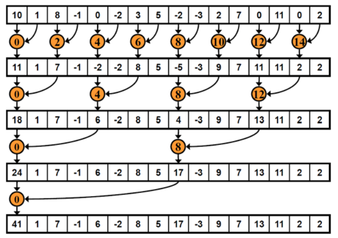
19.9X

Kernel: Shared Memory – Strided Access

```
__global__ void shared_reduce_stride(float* d_out, float* d_in){

 //shared_reduce_stride<<<blocks,threads,threads*sizeof(float)>>>
  int myId = threadIdx.x + blockDim.x * blockIdx.x;
  int tid  = threadIdx.x;
  // load shared mem from global mem
   ________________________________________
  // make sure entire block is loaded!
  // do reduction in shared memory
  for(int stride =      ; stride <        ; stride =        ){

    if(                         ) {
    _________________________________ }
   }
// thread 0 writes result for this block back to global mem
 if (           ) {
    _________________________________ }
}
```

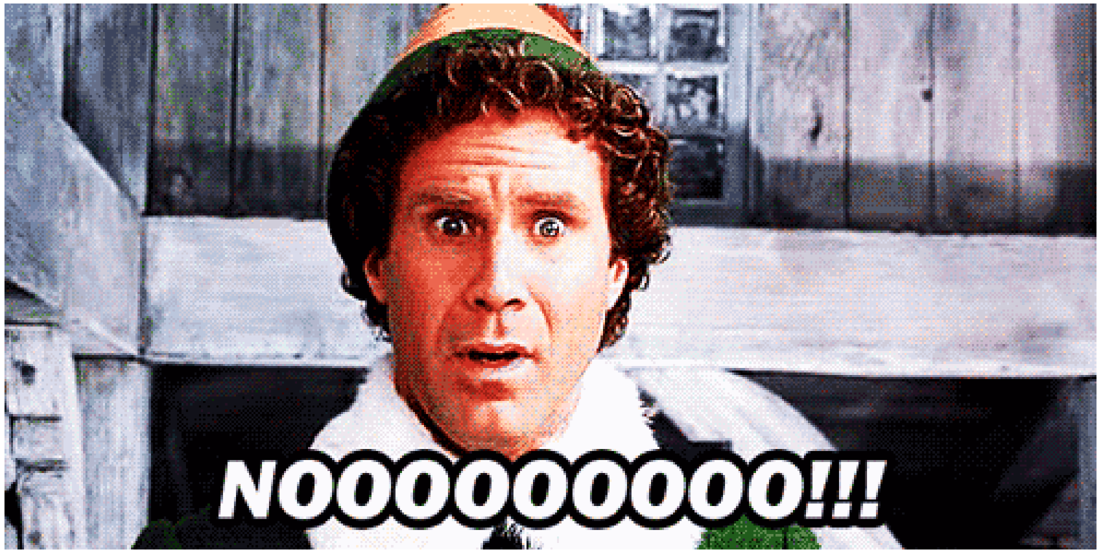


Reduction - Tesla P100;compute v6.0;

n: 1<<20

Version	Time (ms)
serial	3.27400
global reduce stride – naïve	0.16450
shared stride reduce	0.15835

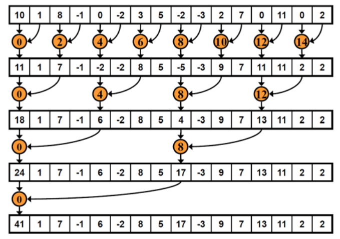
20.7X



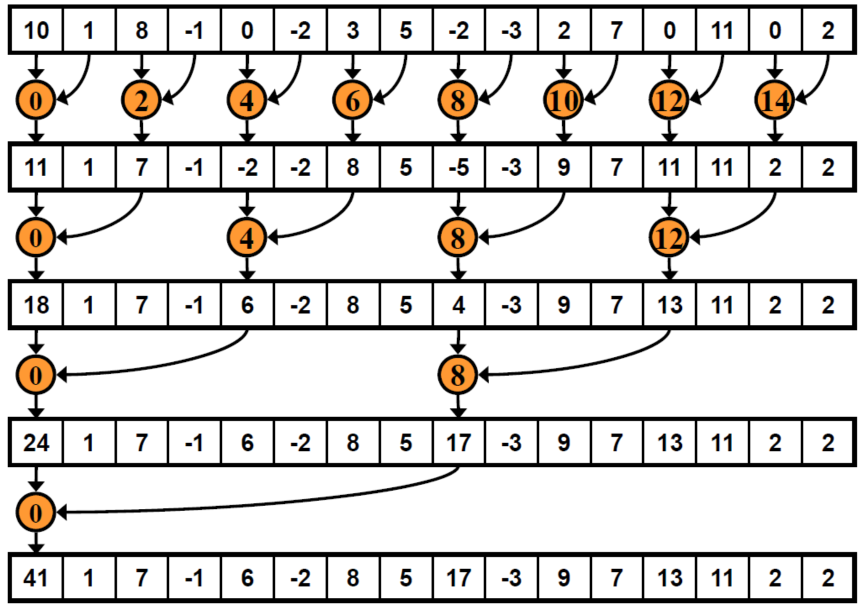
Unusual we should have had better performance!

Kernel: Shared Memory – Strided Access

```
__global__ void shared_reduce_stride(float* d_out, float* d_in){
extern __shared__ float sdata[];
  // shared_reduce<<<blocks,threads,threads*sizeof(float)>>>
  int myId = threadIdx.x + blockDim.x * blockIdx.x;
  int tid  = threadIdx.x;
  // load shared mem from global mem
  sdata[tid] = d_in[myId];
  // make sure entire block is loaded!
  // do reduction in shared memory
  for(int stride = 1; stride < blockDim.x; stride *= 2)  {
    __syncthreads();
    if(myId % (2*stride) == 0) {
      sdata[tid] += sdata[tid+stride]; }
   }
// thread 0 writes result for this block back to global mem
 if (tid == 0) {
        d_out[blockIdx.x] = sdata[tid]; }
}
```



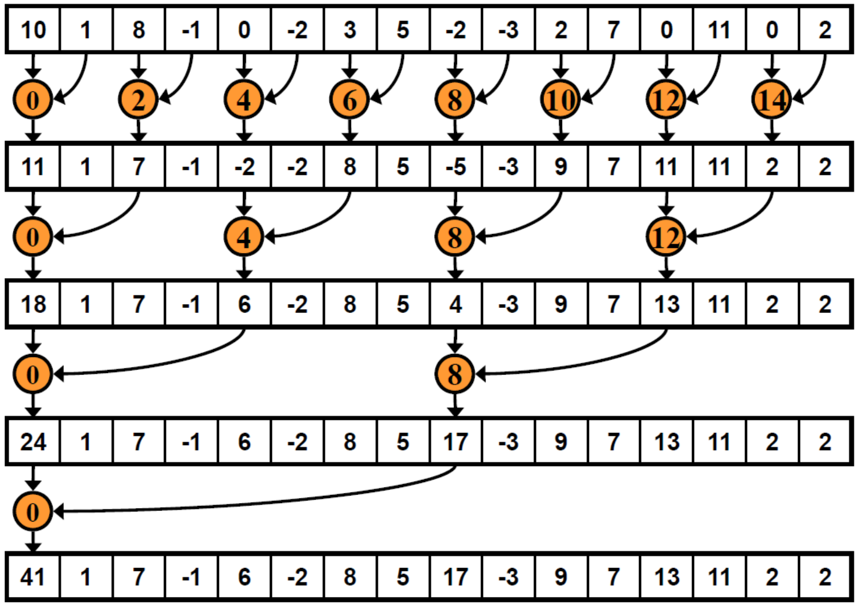
Observations on the Stride Pattern



- Global
  - Memory access pattern

- Each thread
  - responsible for an even-index location of the partial sum vector (location of responsibility)
- In each step
  - second input comes from an increasing distance away
- After each step
  - half of the threads are no longer needed

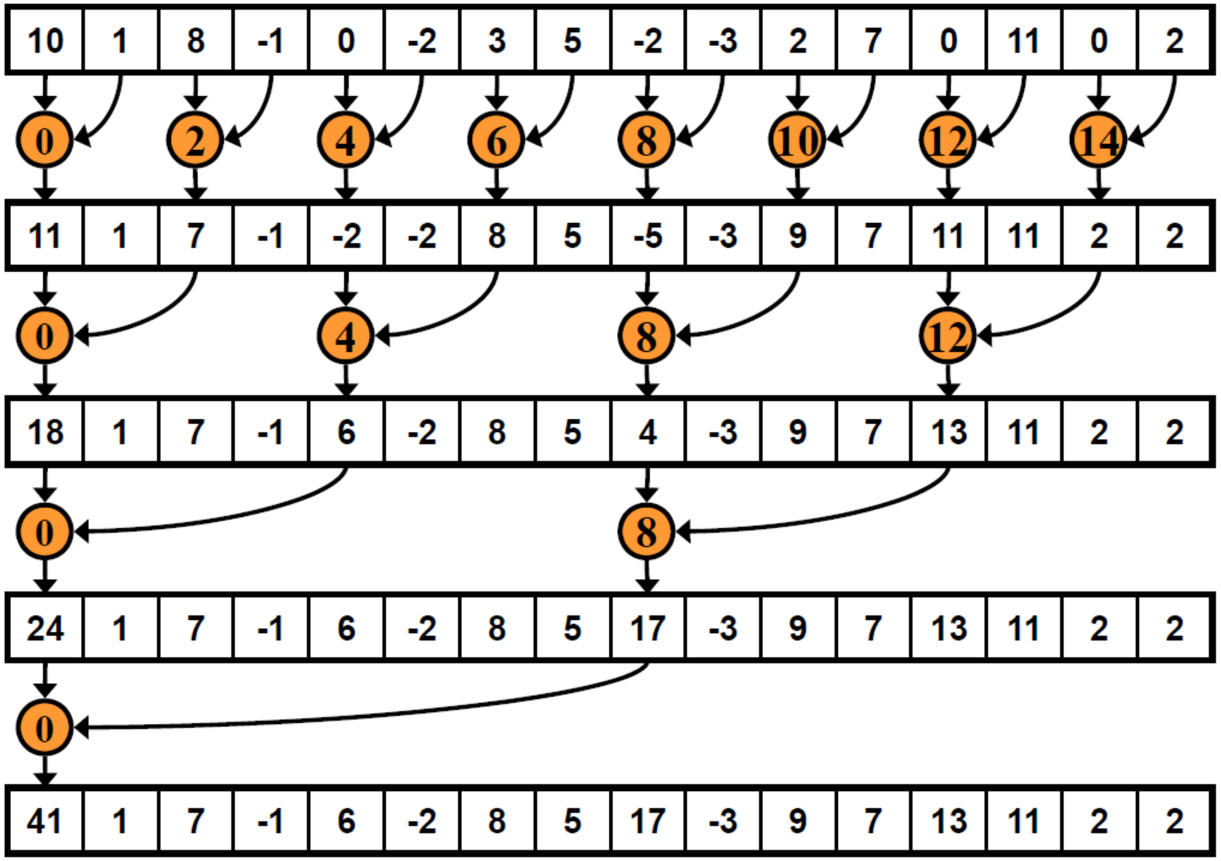
Observations on the Stride Pattern



- In each iteration, two paths will be sequentially traversed for each warp
  - Threads that perform addition and threads that do not

- Half or fewer of threads will be executing after the first step
  - All odd-index threads are disabled after first step

Kernel: Shared Memory – Stride Pattern



Next:

How to reorganize workload assignment to avoid divergence?