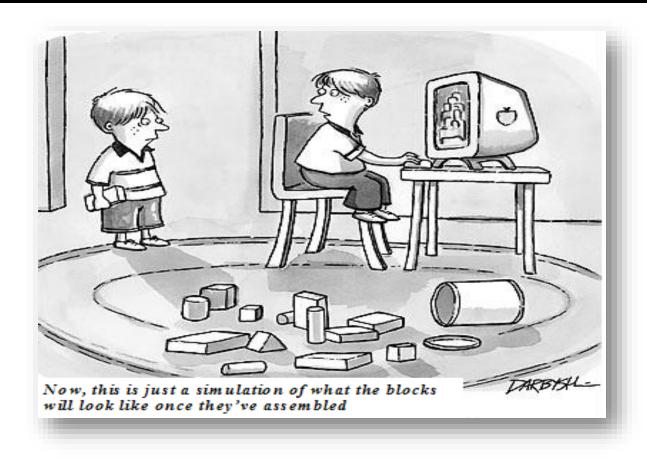
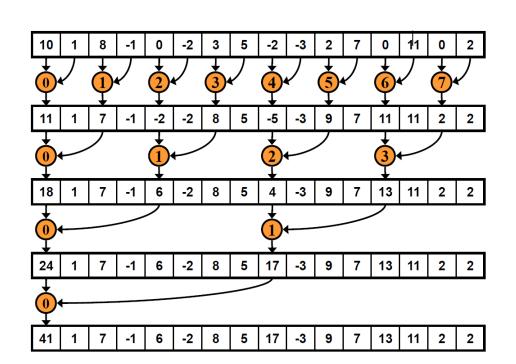
ECE569 Module 36



Reduction – Shared Memory and Bank Conflicts Resolved

1

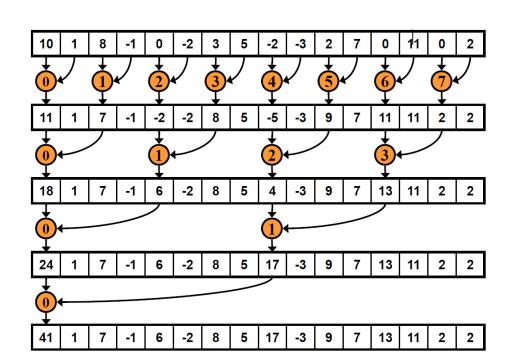
Kernel: Shared Memory – Bank Conflicts in Round 0: 2x stride (1024 elements)



Thread 0
Thread 1
Thread 2
Thread 3
Thread 4
Thread 5
Thread 6
Thread 7
Thread 8
Thread 9
Thread 10
Thread 11
Thread 12
Thread 13
Thread 14
Thread 15
Thread 16
Thread 17
Thread 18
Thread 19
Thread 20
Thread 21
Thread 22
Thread 23
Thread 24
Thread 25
Thread 26
Thread 27
Thread 28
Thread 29
Thread 30
Thread 31

Bank 0 Bank 1
Dalin
Bank 2
Bank 3
Bank 4
Bank 5
Bank 6
Bank 7
Bank 8
Bank 9
Bank 10
Bank 11
Bank 12
Bank 13
Bank 14
Bank 15
Bank 16
Bank 17
Bank 18
Bank 19
Bank 20
Bank 21
Bank 22
Bank 23
Bank 24
Bank 25
Bank 26
Bank 27
Bank 28
Bank 29
Bank 30
Bank 31

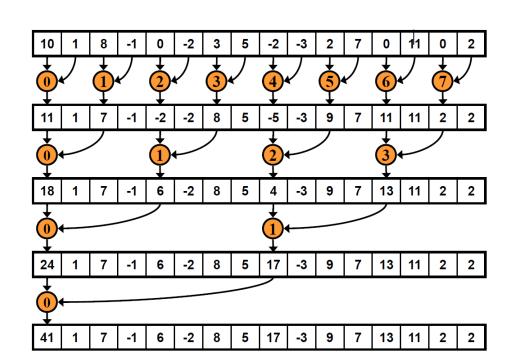
Kernel: Shared Memory – Bank Conflicts in Round 1: 4x stride (512 elements)



Thread 0
Thread 1
Thread 2
Thread 3
Thread 4
Thread 5
Thread 6
Thread 7
Thread 8
Thread 9
Thread 10
Thread 11
Thread 12
Thread 13
Thread 14
Thread 15
Thread 16
Thread 17
Thread 18
Thread 19
Thread 20
Thread 21
Thread 22
Thread 23
Thread 24
Thread 25
Thread 26
Thread 27
Thread 28
Thread 29
Thread 30
Thread 31

Bank 0 Bank 1 Bank 2 Bank 3 Bank 4 Bank 5 Bank 6 Bank 7 Bank 8 Bank 9 Bank 10 Bank 11 Bank 12 Bank 13 Bank 14 Bank 15 Bank 16 Bank 17 Bank 18 Bank 19 Bank 20 Bank 21 Bank 22 Bank 23 Bank 24 Bank 25 Bank 26 Bank 27 Bank 28 Bank 29 Bank 30 Bank 31

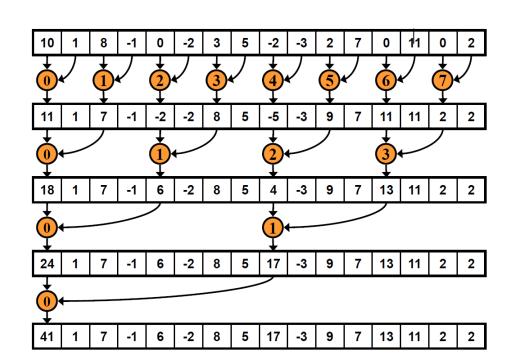
Kernel: Shared Memory – Bank Conflicts in Round 2: 8x stride (256 elements)



Thread 0
Thread 1
Thread 2
Thread 3
Thread 4
Thread 5
Thread 6
Thread 7
Thread 8
Thread 9
Thread 10
Thread 11
Thread 12
Thread 13
Thread 14
Thread 15
Thread 16
Thread 17
Thread 18
Thread 19
Thread 20
Thread 21
Thread 22
Thread 23
Thread 24
Thread 25
Thread 26
Thread 27
Thread 28
Thread 29
Thread 30
Thread 31

Bank 0
Bank 1
Bank 2
Bank 3
Bank 4
Bank 5
Bank 6
Bank 7
Bank 8
Bank 9
Bank 10
Bank 11
Bank 12
Bank 13
Bank 14
Bank 15
Bank 16
Bank 17
Bank 18
Bank 19
Bank 20
Bank 21
Bank 22
Bank 23
Bank 24
Bank 25
Bank 26
Bank 27
Bank 28
Bank 29
Bank 30
Bank 31

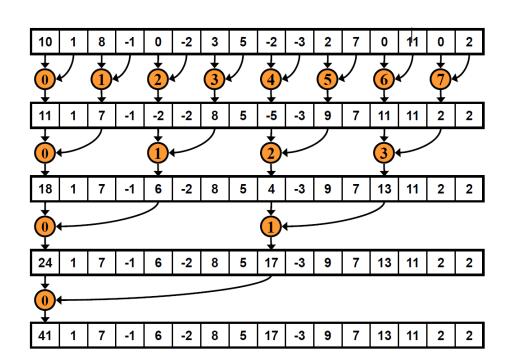
Kernel: Shared Memory – Bank Conflicts in Round 3: 16x (128 elements)



Thread 0
Thread 1
Thread 2
Thread 3
Thread 4
Thread 5
Thread 6
Thread 7
Thread 8
Thread 9
Thread 10
Thread 11
Thread 12
Thread 13
Thread 14
Thread 15
Thread 16
Thread 17
Thread 18
Thread 19
Thread 20
Thread 21
Thread 22
Thread 23
Thread 24
Thread 25
Thread 26
Thread 27
Thread 28
Thread 29
Thread 30
Thread 31

Bank 0 Bank 1 Bank 2 Bank 3 Bank 4 Bank 5 Bank 6 Bank 7 Bank 8 Bank 9 Bank 10 Bank 11 Bank 12 Bank 13 Bank 14 Bank 15 Bank 16 Bank 17 Bank 18 Bank 16 Bank 20	
Bank 2 Bank 3 Bank 4 Bank 5 Bank 6 Bank 7 Bank 8 Bank 9 Bank 10 Bank 11 Bank 12 Bank 13 Bank 14 Bank 15 Bank 16 Bank 17 Bank 18 Bank 20 Bank 20 Bank 21 Bank 20 Bank 21 Bank 22 Bank 23 Bank 24 Bank 25 Bank 26 Bank 27 Bank 28 Bank 29 Bank 29 Bank 29 Bank 30	The state of the s
Bank 3 Bank 4 Bank 5 Bank 6 Bank 7 Bank 8 Bank 9 Bank 10 Bank 11 Bank 12 Bank 13 Bank 14 Bank 15 Bank 16 Bank 17 Bank 16 Bank 20 Bank 20 Bank 21 Bank 20 Bank 21 Bank 22 Bank 23 Bank 24 Bank 25 Bank 26 Bank 27 Bank 28 Bank 29 Bank 29 Bank 29 Bank 30	Bank 1
Bank 4 Bank 5 Bank 6 Bank 7 Bank 8 Bank 9 Bank 10 Bank 11 Bank 12 Bank 13 Bank 14 Bank 15 Bank 16 Bank 17 Bank 18 Bank 20 Bank 20 Bank 21 Bank 20 Bank 21 Bank 22 Bank 23 Bank 24 Bank 25 Bank 24 Bank 25 Bank 26 Bank 27 Bank 28 Bank 29 Bank 29 Bank 29	Bank 2
Bank 5 Bank 6 Bank 7 Bank 8 Bank 9 Bank 10 Bank 11 Bank 12 Bank 13 Bank 14 Bank 15 Bank 16 Bank 17 Bank 18 Bank 19 Bank 20 Bank 21 Bank 21 Bank 22 Bank 23 Bank 24 Bank 25 Bank 25 Bank 26 Bank 27 Bank 28 Bank 29 Bank 29 Bank 30	Bank 3
Bank 6 Bank 7 Bank 8 Bank 9 Bank 10 Bank 11 Bank 12 Bank 13 Bank 14 Bank 15 Bank 16 Bank 17 Bank 18 Bank 19 Bank 20 Bank 20 Bank 21 Bank 22 Bank 23 Bank 24 Bank 25 Bank 25 Bank 26 Bank 27 Bank 28 Bank 29 Bank 29 Bank 30	Bank 4
Bank 7 Bank 8 Bank 9 Bank 10 Bank 11 Bank 12 Bank 13 Bank 14 Bank 15 Bank 16 Bank 17 Bank 18 Bank 19 Bank 20 Bank 21 Bank 21 Bank 22 Bank 23 Bank 24 Bank 25 Bank 26 Bank 27 Bank 28 Bank 29 Bank 29 Bank 30	Bank 5
Bank 8 Bank 9 Bank 10 Bank 11 Bank 12 Bank 13 Bank 14 Bank 15 Bank 16 Bank 17 Bank 18 Bank 19 Bank 20 Bank 21 Bank 21 Bank 22 Bank 23 Bank 24 Bank 25 Bank 25 Bank 26 Bank 27 Bank 28 Bank 29 Bank 30	Bank 6
Bank 9 Bank 10 Bank 11 Bank 12 Bank 13 Bank 14 Bank 15 Bank 16 Bank 17 Bank 18 Bank 19 Bank 20 Bank 21 Bank 21 Bank 22 Bank 23 Bank 24 Bank 25 Bank 25 Bank 26 Bank 27 Bank 28 Bank 29 Bank 30	Bank 7
Bank 10 Bank 11 Bank 12 Bank 13 Bank 14 Bank 15 Bank 16 Bank 17 Bank 18 Bank 19 Bank 20 Bank 21 Bank 21 Bank 25 Bank 24 Bank 25 Bank 26 Bank 27 Bank 28 Bank 29 Bank 30	Bank 8
Bank 11 Bank 12 Bank 13 Bank 14 Bank 15 Bank 16 Bank 17 Bank 18 Bank 19 Bank 20 Bank 21 Bank 21 Bank 25 Bank 24 Bank 25 Bank 26 Bank 27 Bank 28 Bank 29 Bank 30	Bank 9
Bank 12 Bank 13 Bank 14 Bank 15 Bank 16 Bank 17 Bank 18 Bank 19 Bank 20 Bank 21 Bank 22 Bank 23 Bank 24 Bank 25 Bank 26 Bank 27 Bank 28 Bank 29 Bank 30	Bank 10
Bank 13 Bank 14 Bank 15 Bank 16 Bank 17 Bank 18 Bank 19 Bank 20 Bank 21 Bank 21 Bank 22 Bank 23 Bank 24 Bank 25 Bank 26 Bank 27 Bank 28 Bank 29 Bank 30	
Bank 14 Bank 15 Bank 16 Bank 17 Bank 18 Bank 19 Bank 20 Bank 21 Bank 22 Bank 23 Bank 24 Bank 25 Bank 26 Bank 27 Bank 28 Bank 29 Bank 30	Bank 12
Bank 15 Bank 16 Bank 17 Bank 18 Bank 19 Bank 20 Bank 21 Bank 22 Bank 23 Bank 24 Bank 25 Bank 26 Bank 27 Bank 28 Bank 29 Bank 30	Bank 13
Bank 16 Bank 17 Bank 18 Bank 19 Bank 20 Bank 21 Bank 22 Bank 23 Bank 24 Bank 25 Bank 26 Bank 27 Bank 28 Bank 29 Bank 30	Bank 14
Bank 17 Bank 18 Bank 19 Bank 20 Bank 21 Bank 22 Bank 23 Bank 24 Bank 25 Bank 26 Bank 27 Bank 28 Bank 29 Bank 30	Bank 15
Bank 18 Bank 19 Bank 20 Bank 21 Bank 22 Bank 23 Bank 24 Bank 25 Bank 26 Bank 27 Bank 28 Bank 29 Bank 30	Bank 16
Bank 19 Bank 20 Bank 21 Bank 22 Bank 23 Bank 24 Bank 25 Bank 26 Bank 27 Bank 28 Bank 29 Bank 30	Bank 17
Bank 20 Bank 21 Bank 22 Bank 23 Bank 24 Bank 25 Bank 26 Bank 27 Bank 28 Bank 29 Bank 30	Bank 18
Bank 21 Bank 22 Bank 23 Bank 24 Bank 25 Bank 26 Bank 27 Bank 28 Bank 29 Bank 30	
Bank 22 Bank 23 Bank 24 Bank 25 Bank 26 Bank 27 Bank 28 Bank 29 Bank 30	Bank 20
Bank 23 Bank 24 Bank 25 Bank 26 Bank 27 Bank 28 Bank 29 Bank 30	Bank 21
Bank 24 Bank 25 Bank 26 Bank 27 Bank 28 Bank 29 Bank 30	Bank 22
Bank 25 Bank 26 Bank 27 Bank 28 Bank 29 Bank 30	Bank 23
Bank 26 Bank 27 Bank 28 Bank 29 Bank 30	Bank 24
Bank 27 Bank 28 Bank 29 Bank 30	Bank 25
Bank 28 Bank 29 Bank 30	
Bank 29 Bank 30	Bank 27
Bank 30	Bank 28
	Bank 29
Bank 31	Bank 30
	Bank 31

Kernel: Shared Memory – Bank Conflicts in Round 4: 32x (64 elements)

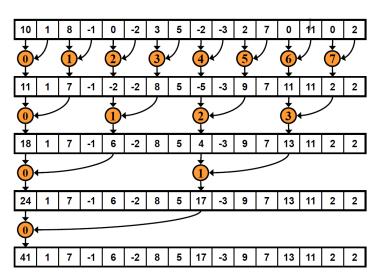


Thread 0	
Thread 1	
Thread 2	
Thread 3	
Thread 4	
Thread 5	
Thread 6	
Thread 7	
Thread 8	
Thread 9	
Thread 10	
Thread 11	
Thread 12	
Thread 13	
Thread 14	
Thread 15	
Thread 16	
Thread 17	
Thread 18	
Thread 19	
Thread 20	
Thread 21	
Thread 22	
Thread 23	
Thread 24	
Thread 25	
Thread 26	
Thread 27	
Thread 28	
Thread 29	
Thread 30	
Thread 31	

Bank 0 Bank 1 Bank 2 Bank 3 Bank 4 Bank 5 Bank 6 Bank 7 Bank 8 Bank 9 Bank 10 Bank 11 Bank 12 Bank 13 Bank 14 Bank 15 Bank 16 Bank 17 Bank 18 Bank 19 Bank 20 Bank 21 Bank 22 Bank 23 Bank 24 Bank 25 Bank 26 Bank 27 Bank 28 Bank 29 Bank 30 Bank 31

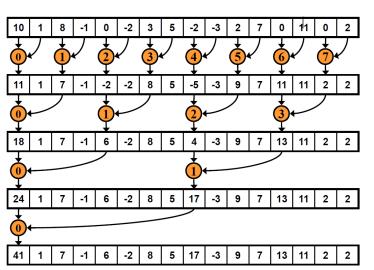
Kernel: Shared Memory – Resolving Bank Conflicts

- Need a new thread to data mapping function that will allow sequence of 32 threads access subsequent addresses
 - But workload per thread is two elements.



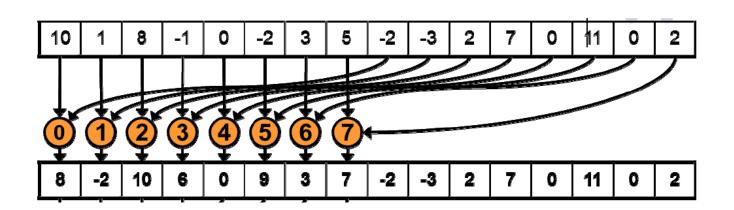
Kernel: Shared Memory – Resolving Bank Conflicts

- Need a new thread to data mapping function that will allow sequence of 32 threads access subsequent addresses
 - But workload per thread is two elements.
- Reduction is associative:
 - we can change index and stride



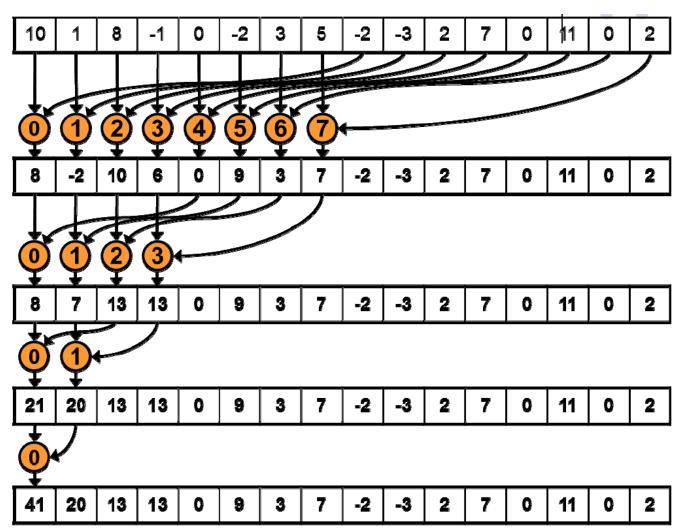
Kernel: Shared Memory – Resolving Bank Conflicts

- Need a new thread to data mapping function that will allow sequence of 32 threads access subsequent addresses
 - But workload per thread is two elements.
- Reduction is associative:
 - we can change index and stride
- What if we assign sequence of 32 elements as first inputs for each thread and feed second input data that is half of the thread block size distance away in round 0?



Next: Shared Memory – Reverse Stride Pattern

Reverse the access pattern and reduce the stride by half in each round



Kernel: Shared Memory – Reverse –Pattern

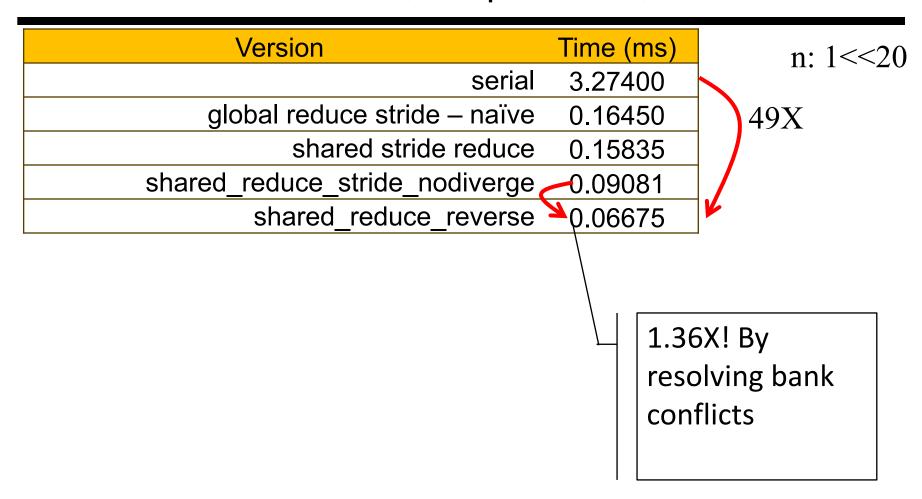
```
__global__ void shared_reduce_reverse(float* d_out, float* d_in) {
    extern __shared__ float sdata[];
    // shared_reduce<<<br/>blocks, threads, threads*sizeof(float)>>>
    int myId = threadIdx.x + blockDim.x * blockIdx.x;
    int tid = threadIdx.x;
    // load shared mem from global mem
    sdata[tid] = d_in[myId];
    // make sure entire block is loaded!
    // do reduction in shared memory
```

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

Kernel: Shared Memory – Reverse Pattern

```
global void shared reduce reverse(float* d out, float* d in) {
extern shared float sdata[];
// shared reduce<<<blooks,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];
syncthreads();
// make sure entire block is loaded!
                                                 20 13 13 0 9 3 7 -2 -3 2 7 0 11 0 2
 // do reduction in shared memory
for (unsigned int s = blockDim.x / 2; s > 0; s >>= 1) {
       if (tid < s) {
           sdata[tid] += sdata[tid + s]; }
     syncthreads();
if (tid == 0) {d out[blockIdx.x] = sdata[tid]; }
```

Reduction - Tesla P100; compute v6.0;



Observations on the Reverse Stride Pattern

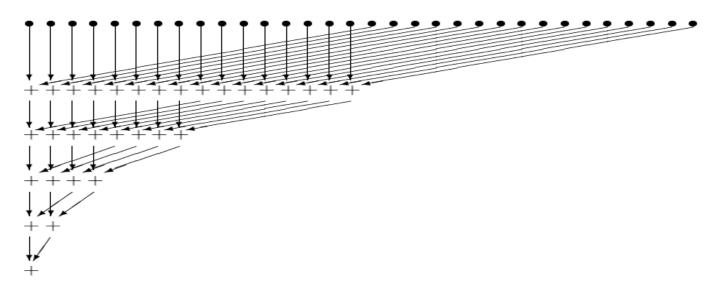
```
for (unsigned int s = blockDim.x / 2; s > 0; s >>= 1) {
       if (tid < s) {
           sdata[tid] += sdata[tid + s]; }
```

- Note that half of the threads are idle on first loop iteration!
 - This is wasteful...
- What can we do?

First Reduction

- Use only half the blocks
- Do the first reduction during the load from the global memory
 - Replace single load with two loads
 - Kernel launch will be:

```
shared_reverse_firstreduction <<<ble>blocks/2, threads,<br/>threads * sizeof(float)>>>
```



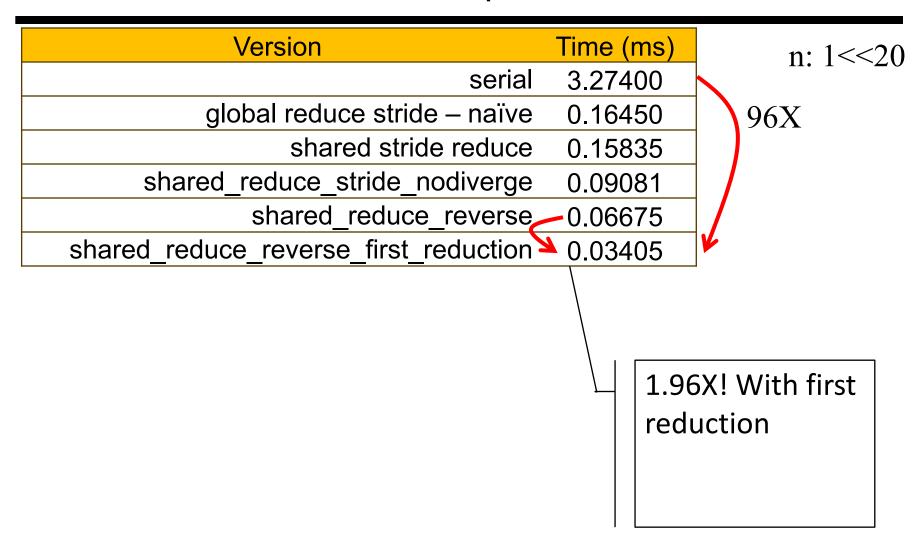
Kernel: Shared, Reverse, First reduction

```
global void shared reverse firstreduction (float * d out,
float * d in) {
  extern shared float sdata[];
  // shared reduce << < blocks, threads, threads * size of (float) >>>
  int myId = threadIdx.x + blockDim.x * blockIdx.x;
  int tid = threadIdx.x;
  // load shared mem from global mem
  sdata[tid] = d in[myId];
                                                             13 13 0 9 3 7 -2 -3 2 7 0 11 0 2
  syncthreads();
                                                           20 | 13 | 13 | 0 | 9 | 3 | 7 | -2 | -3 | 2 | 7 | 0 | 11 | 0 | 2
  // make sure entire block is loaded!
                                                          41 20 13 13 0 9 3 7 -2 -3 2 7 0 11 0 2
  // do reduction in shared memory
  for (unsigned int s = blockDim.x / 2; s > 0; s >>= 1) {
         if (tid < s) {
              sdata[tid] += sdata[tid + s]; }
      syncthreads();
 if (tid == 0) {d out[blockIdx.x] = sdata[tid]; }
                Revise for first reduction
```

Kernel: Shared, Reverse, First reduction

```
global void shared reverse firstreduction (float * d out,
float * d in) {
 extern shared float sdata[];
  // shared reduce<<<blooks, threads, threads*sizeof(float)>>>
  int myId = threadIdx.x + blockDim.x * 2 * blockIdx.x;
  int tid = threadIdx.x;
                                                     blockIdx.x is
  // load shared mem from global mem
                                                     half of the
   sdata[tid] = d in[myId]+d in[myId+blockDim.x];
                                                     original range!
  syncthreads();
  // make sure entire block is loaded!
  // do reduction in shared memory
  for (unsigned int s = blockDim.x / 2; s > 0; s >>= 1) {
        if (tid < s) {
            sdata[tid] += sdata[tid + s]; }
      syncthreads();
if (tid == 0) {d out[blockIdx.x] = sdata[tid]; }
```

Reduction - Tesla P100; compute v6.0;

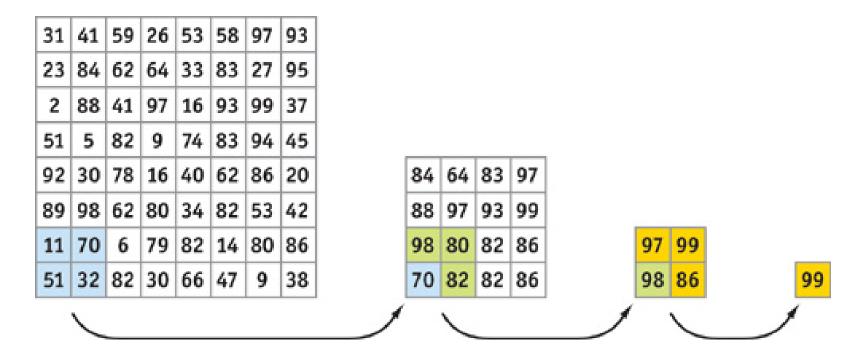


Parallel Sum Reduction Summary

- Parallel implementation
 - Recursively halve # of threads, add two values per thread in each step
 - Takes log(n) steps for n elements, requires n/2 threads
- In-place reduction using shared memory
 - The original vector is in device global memory
 - The shared memory is used to hold a partial sum vector
 - Each step brings the partial sum vector closer to the sum
 - The final sum will be in element 0 of the partial sum vector
 - Reduces global memory traffic due to partial sum values
 - Thread block size limits n to be less than or equal to 2,048

2D Max Reduction Example

 Read four elements from four quadrants of the input buffer, such that the output size is halved in both dimensions at each step.



Next

Scan