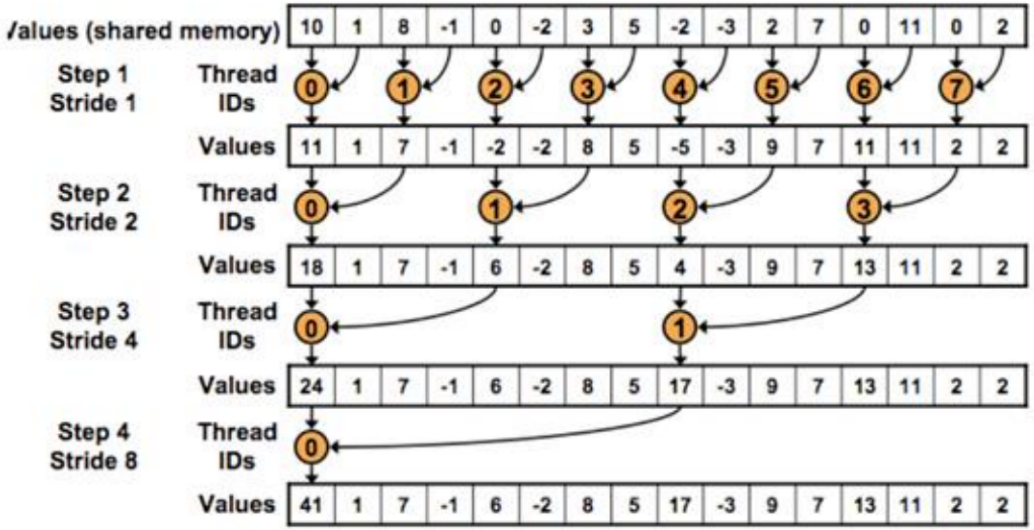
Step 1: Interleaved Addressing



Picture courtesy: EECS471, UM

Step 1: Interleaved Addressing

```
6   __global__ void reduce(int *g_idata, int *g_odata, int len)
7   {
8     //Method1: simple reduction using intervleaved addresing======
9     __shared__ int sdata[BLOCK_SIZE];
10    int tid = threadIdx.x;
11    int i = blockIdx.x * blockDim.x + threadIdx.x;
12
13
14    sdata[tid] = g_idata[i];
15    __syncthreads();
16
17    // do reduction in shared mem
18    for (int s = 1; s < blockDim.x; s *= 2)
19    {
20      if (tid % (2 * s) == 0)
21      {
22        sdata[tid] += sdata[tid + s];
23      }
24      __syncthreads();
25    }
```

- Summing 1G elements on MI250 takes 11.79ms
- Divergent branching

Step 1: Problem: Inactive threads & branching

Pipeline Stats

	Avg	Min	Max	Unit
VALU Active Threads	45.85	45.85	45.85	Threads

Instruction Mix

Profile & analyze using Omniperf:

- omniperf profile –n reduce_X -- ./reduce_X
- omniperf analyze -p workloads/reduce_X/mi200/ --gui --random-port

Step 2: Avoid divergent branching and %

Replace

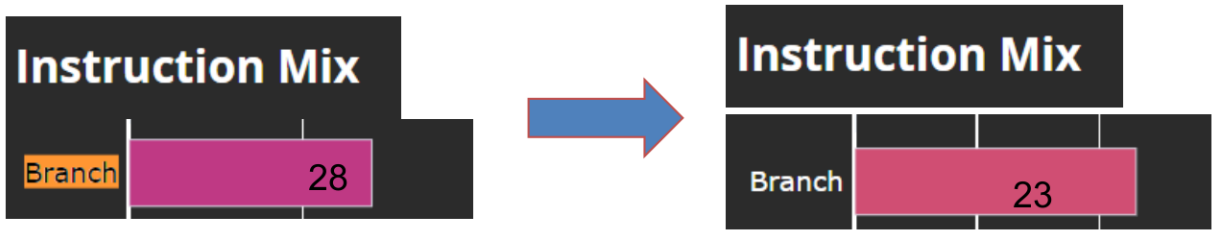
```
19    // do reduction in shared mem
20    for (int s = 1; s < blockDim.x; s *= 2)
21    {
22      if (tid % (2 * s) == 0)
23      {
24        sdata[tid] += sdata[tid + s];
25      }
26      __syncthreads();
27    }
```

With:

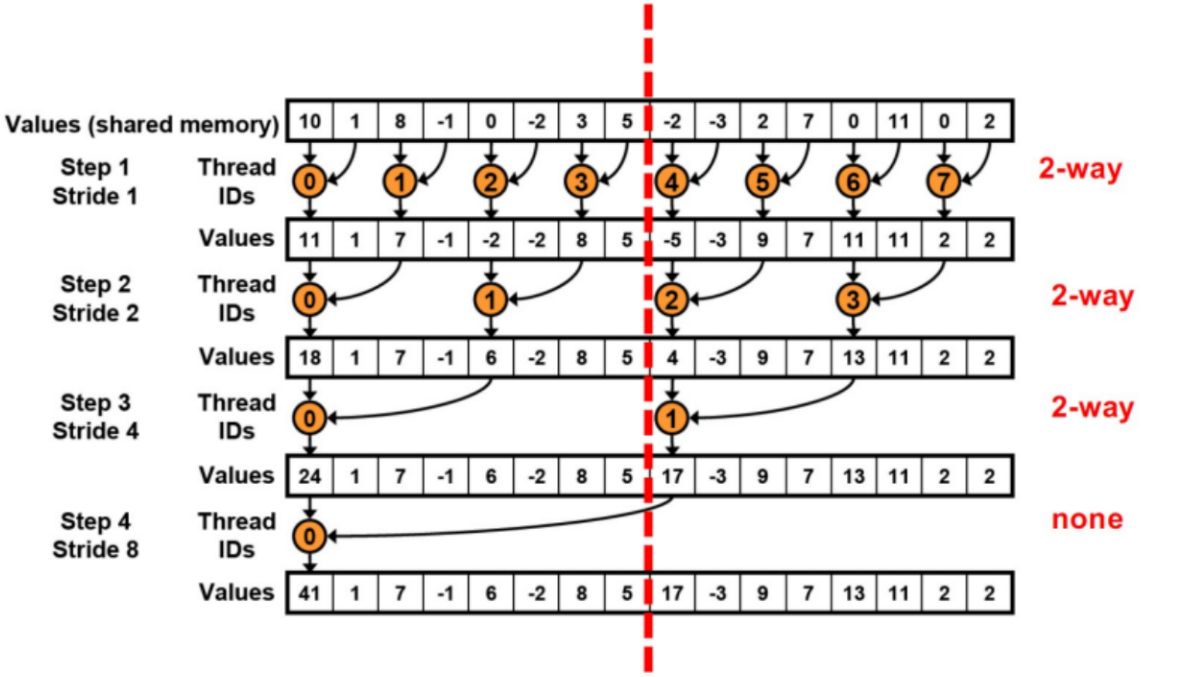
```
19    // do reduction in shared mem
20    for (int s = 1; s < blockDim.x; s=s<<1)
21    {
22      int index= s*tid<<1;
23      if (index< blockDim.x)
24      {
25        sdata[index] += sdata[index + s];
26      }
27      __syncthreads();
28    }
```

Step 2: More active threads & less branching

Pipeline Stats	Avg	Min	Max	Unit
VALU Active Threads	60.23	60.23	60.23	Threads



Step 2: Avoid divergent branching and %



2-way bank conflicts at every step

Recall: 64 threads per wavefront and 32 banks of LDS

To see the conflicts in the picture above (8 threads) assume banks = 8

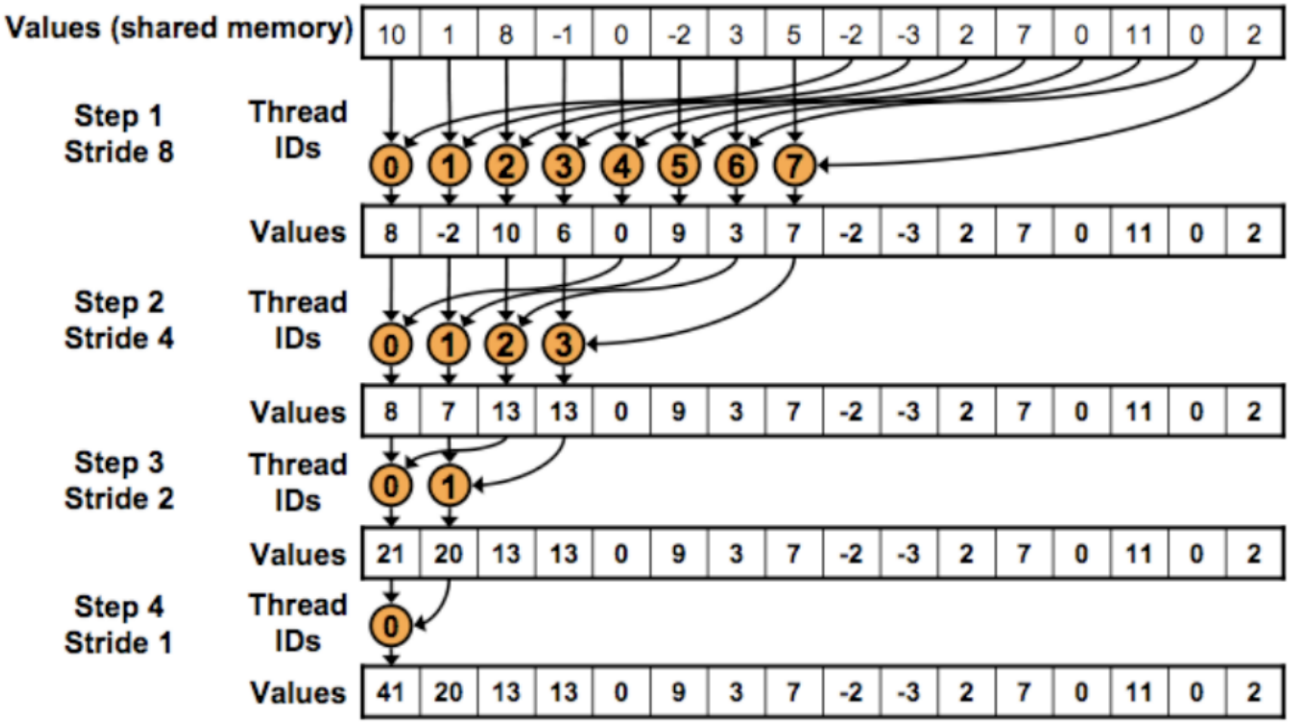
Picture courtesy: EECS471, UM

Step 2: Problem: LDS bank conflicts

12.2 LDS Stats

Bank Conflict	28.13	28.13	28.13	Cycles per wave

Step 3: Sequential addressing



Picture courtesy: EECS471, UM

Step 3: No LDS bank conflicts

12.2 LDS Stats

Bank Conflict	0.00	0.00	0.00	Cycles per wave

Step 3: Sequential addressing

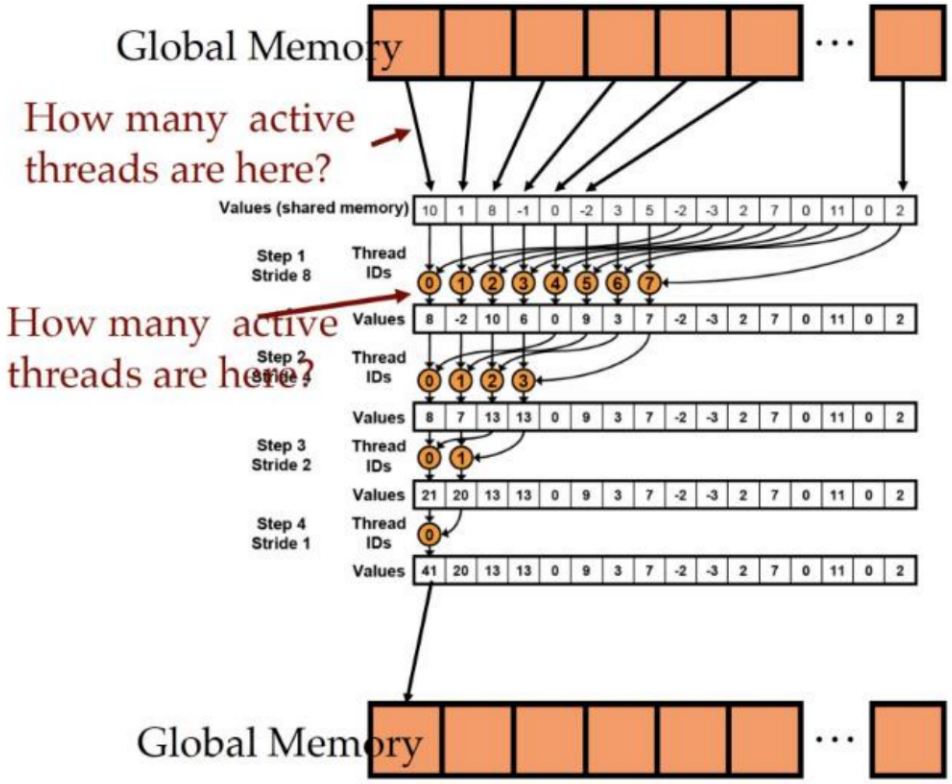
Replace

```
19    // do reduction in shared mem
20    for (int s = 1; s < blockDim.x; s=s<<1)
21    {
22      int index= s*tid<<1;
23      if (index< blockDim.x)
24      {
25        sdata[index] += sdata[index + s];
26      }
27      __syncthreads();
28    }
```

With:

```
19    // do reduction in shared mem
20    for (int s = blockDim.x/2; s >0; s>>=1)
21    {
22      if (tid<s)
23      {
24        sdata[tid] += sdata[tid + s];
25      }
26      __syncthreads();
27    }
```

Increasing #inactive threads after each iteration



Picture courtesy: EECS471, UM

Step 4: First add during load

Original: Each thread reads one element

```
10  int tid = threadIdx.x;
11  int i = blockIdx.x * blockDim.x + threadIdx.x;
12
13  sdata[tid] = g_idata[i];
14  __syncthreads();
```

Original: Halve the number of blocks & insert two loads and add into the first step

```
10  int tid = threadIdx.x;
11  int i = blockIdx.x * blockDim.x*2 + threadIdx.x;
12
13  sdata[tid] = g_data[i]+g_data[i+blockDim.x];
14  __syncthreads();
```

Use Step-4 or higher for HW3

Step 4: Increased VALU utilization

Metric	Value	Unit	Peak	PoP
VALU FLOPs	175.59	Gflop	22630.40	0.78

Metric	Value	Unit	Peak	PoP
VALU FLOPs	258.02	Gflop	22630.40	1.14

Step 5: Unroll the last iteration of the loop

- Unroll iteration of the loop
  - As instruction proceeds, #active threads decreases
  - We do not want all wavefronts executing the last iteration of the loop
  - When we have less than 64 threads
    - We don’t need __syncthreads()
    - Don’t need if statements

Step 5: Unroll the last iteration of the loop

```
__shared__ volatile int sdata[BLOCK_SIZE];
```

```
19  // do reduction in shared mem
20  for (int s = blockDim.x/2; s >64; s>>=1)
21  {
22    if (tid<s)
23    {
24      sdata[tid] += sdata[tid + s];
25    }
26    __syncthreads();
27  }
28  //Last loop iteration is unrolled
29  if(tid<64){
30    sdata[tid]+=sdata[tid+64];
31    sdata[tid]+=sdata[tid+32];
32    sdata[tid]+=sdata[tid+16];
33    sdata[tid]+=sdata[tid+8];
34    sdata[tid]+=sdata[tid+4];
35    sdata[tid]+=sdata[tid+2];
36    sdata[tid]+=sdata[tid+1];
37  }
```

(Optional) Step 6: Complete Unrolling

- Complete Unrolling
  - If we knew the #iterations at compile time we could completely unroll reduction
  - Use templates
    - Some of the branching will be evaluated at compile time
    - Block size will be our template parameter

Step 6: Complete unrolling using templates

```
template <unsigned int blockSize>
__global__ void reduce6(int *g_data, int *g_odata)
{
   /* ... global memory access + first reduction is somewhere here ... */

   if (blockSize >= 1024) {
        if (tid < 512) { sdata[tid] += sdata[tid + 512]; } __syncthreads();
   }
   if (blockSize >= 512) {
        if (tid < 256) { sdata[tid] += sdata[tid + 256]; } __syncthreads();
   }
   if (blockSize >= 256) {
        if (tid < 128) { sdata[tid] += sdata[tid + 128]; } __syncthreads();
   }
   if (blockSize >= 128) {
        if (tid < 64) { sdata[tid] += sdata[tid + 64]; } __syncthreads();
   }
   if (tid < 32) {
        if (blockSize >= 64) sdata[tid] += sdata[tid + 32];
        if (blockSize >= 32) sdata[tid] += sdata[tid + 16];
        if (blockSize >= 16) sdata[tid] += sdata[tid + 8];
        if (blockSize >= 8) sdata[tid] += sdata[tid + 4];
        if (blockSize >= 4) sdata[tid] += sdata[tid + 2];
        if (blockSize >= 2) sdata[tid] += sdata[tid + 1];
   }
}
```

Step 6: Complete unrolling using constexpr

```
18      // Unroll loops based on BLOCK_SIZE
19      if constexpr (BLOCK_SIZE >= 512)
20      {
21          if (tid < 256)
22              sdata[tid] += sdata[tid + 256];
23          __syncthreads();
24      }
25
26      if constexpr (BLOCK_SIZE >= 256)
27      {
28          if (tid < 128)
29              sdata[tid] += sdata[tid + 128];
30          __syncthreads();
31      }
32      if constexpr (BLOCK_SIZE >= 128)
33      {
34          if (tid < 64)
35              sdata[tid] += sdata[tid + 64];
36      }
37
38      if (tid < 64)
39      {
40
41          if constexpr (BLOCK_SIZE >= 64)
42              sdata[tid] += sdata[tid + 32];
43          if constexpr (BLOCK_SIZE >= 32)
44              sdata[tid] += sdata[tid + 16];
45          if constexpr (BLOCK_SIZE >= 16)
46              sdata[tid] += sdata[tid + 8];
47          if constexpr (BLOCK_SIZE >= 8)
48              sdata[tid] += sdata[tid + 4];
49          if constexpr (BLOCK_SIZE >= 4)
50              sdata[tid] += sdata[tid + 2];
51          if constexpr (BLOCK_SIZE >= 2)
52              sdata[tid] += sdata[tid + 1];
53      }
```

(Optional)Step 7: Algorithm cascading: Mix parallel & sequential execution

- Replace "load and reduce two elements"

```
// each thread loads two elements from global to shared mem
// end performs the first step of the reduction
unsigned int tid = threadIdx.x;
unsigned int i = blockIdx.x* blockDim.x * 2 + threadIdx.x;
sdata[tid] = g_idata[i] + g_idata[i + blockDim.x];
__syncthreads();
```

- With a loop to reduce as many elements as necessary

```
unsigned int tid = threadIdx.x;
unsigned int i = blockIdx.x * blockSize * 2 + threadIdx.x;
unsigned int gridSize = blockSize * 2 * gridDim.x;

sdata[tid] = 0;
while (i < n) {
    sdata[tid] += g_idata[i] + g_idata[i + blockSize];
    i += gridSize;
  }
__syncthreads();
```

gridSize steps to achieve coalescing

EEC471, Prof. Reetu Das

Reduction is memory bound

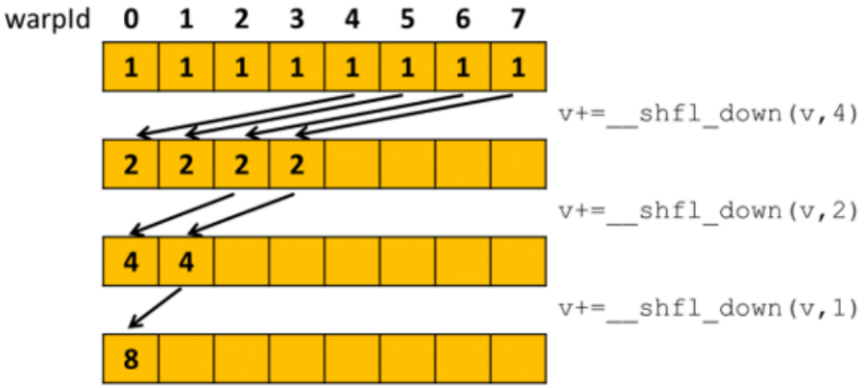
Arithmetic Intensity is 1

- AI measures FLOPs/Byte
- O(N) ops on O(N) bytes of data

(Optional) Step 8: Using warp shuffles for inter-thread communication

__shfl_down:

```
/* warp-level sum using registers within a warp */
int offset;
for (offset = warpSize / 2; offset > 0; offset /= 2)
{
          data += __shfl_down(data, offset );
}
```



Courtesy: ROCm Training from Prof. David Kaeli