Introduction to Parallel and Distributed Processing

CUDA Reduce and Prefix

Jaroslaw 'Jaric' Zola

http://www.jzola.org/

Classic Problem

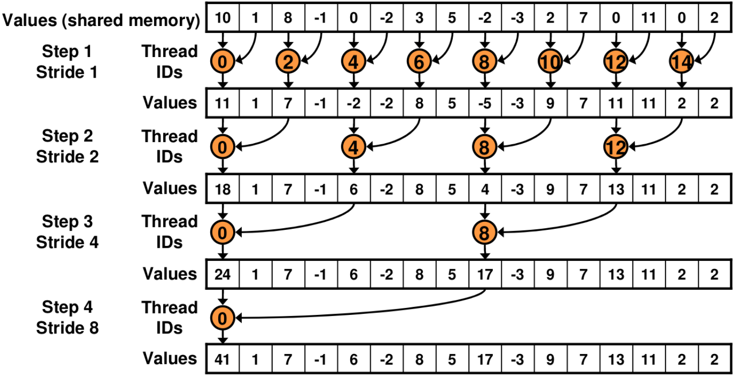
- Given vector $X = (x_1, \ldots, x_n)$ compute $\sum_{i=1}^{n} x_i$
- Smells like a simple 1D problem...

Direct Approach

```
__global__ void reduce0(int* gin, int* gout) {
  extern __shared__ int sdata[];

  int tid = threadIdx.x;
  int i = blockIdx.x * blockDim.x + tid;

  sdata[tid] = gin[i];
  __syncthreads();

  for (int s = 1; s < blockDim.x; s *= 2) {
    if (tid % (2 * s) == 0) sdata[tid] += sdata[tid + s];
    __syncthreads();
  }

  if (tid == 0) gout[blockIdx.x] = sdata[0];
} // reduce0
```

Direct Approach – Problem

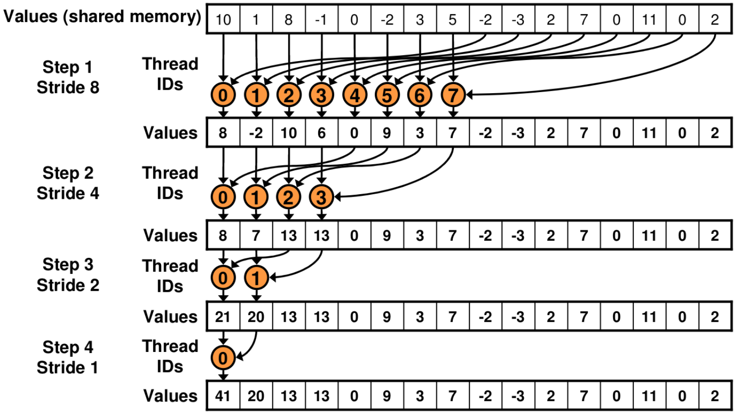
Divergent branching:



Values (shared memory)	10	1	8	-1	0	-2	3	5	-2	-3	2	7	0	11	0	2
Values	11	1	7	-1	-2	-2	8	5	-5	-3	9	7	11	11	2	2
Values	18	1	7	-1	6	-2	8	5	4	-3	9	7	13	11	2	2
Values	24	1	7	-1	6	-2	8	5	17	-3	9	7	13	11	2	2
Values	41	1	7	-1	6	-2	8	5	17	-3	9	7	13	11	2	2

Better Approach

Sequential addressing:



Values (shared memory)		10	1	8	-1	0	-2	3	5	-2	-3	2	7	0	11	0	2
Step 1 Stride 8	Thread IDs	0	1	2	3	4	5	6	7								
	Values	8	-2	10	6	0	9	3	7	-2	-3	2	7	0	11	0	2
Step 2 Stride 4	Thread IDs	0	1	2	3												
	Values	8	7	13	13	0	9	3	7	-2	-3	2	7	0	11	0	2
Step 3 Stride 2	Thread IDs	0	1														
	Values	21	20	13	13	0	9	3	7	-2	-3	2	7	0	11	0	2
Step 4 Stride 1	Thread IDs	0															
	Values	41	20	13	13	0	9	3	7	-2	-3	2	7	0	11	0	2

Sequential Addressing

```
__global__ void reduce1(int* gin, int* gout) {
  extern __shared__ int sdata[];

  int tid = threadIdx.x;
  int i = blockIdx.x * blockDim.x + tid;

  sdata[tid] = gin[i];
  __syncthreads();

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

  if (tid == 0) gout[blockIdx.x] = sdata[0];
} // reduce1
```

Even Better Sequential Addressing

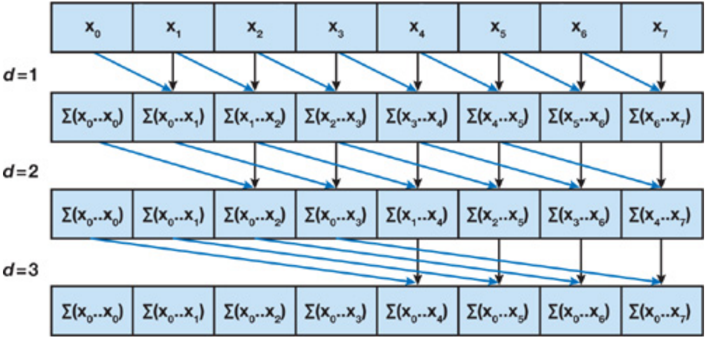
```
// we halve the number of blocks
__global__ void reduce2(int* gin, int* gout) {
  extern __shared__ int sdata[];

  int tid = threadIdx.x;
  int i = blockIdx.x * (blockDim.x * 2) + threadIdx.x;

  sdata[tid] = gin[i] + gin[i + blockDim.x];
  __syncthreads();

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

  if (tid == 0) gout[blockIdx.x] = sdata[0];
} // reduce2
```

What About Parallel Prefix?

- Again, 1D problem PREFIX_SUM

```
Input: $[x_0, x_1, \ldots, x_{n-1}]$
for $d = 1 \ldots \log(n)$ do
    for $j = 1 \ldots n$ pardo
        if $j \geq 2^d$ then
            $x[j] = x[j - 2^{d-1}] + x[j]$
```

What About Parallel Prefix?



x_0 x_1 x_2 x_3 x_4 x_5 x_6 x_7

$d=1$

$\Sigma(x_0..x_0)$ $\Sigma(x_0..x_1)$ $\Sigma(x_1..x_2)$ $\Sigma(x_2..x_3)$ $\Sigma(x_3..x_4)$ $\Sigma(x_4..x_5)$ $\Sigma(x_5..x_6)$ $\Sigma(x_6..x_7)$

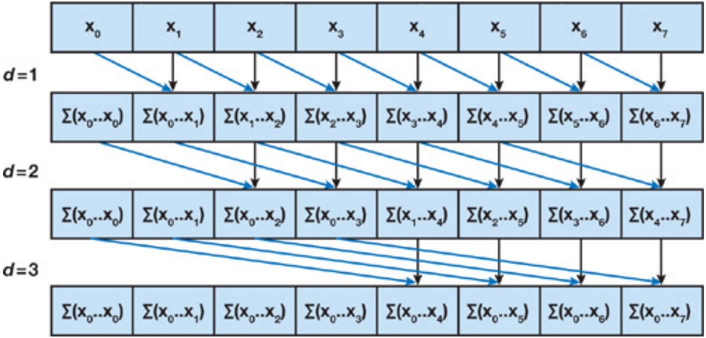
$d=2$

$\Sigma(x_0..x_0)$ $\Sigma(x_0..x_1)$ $\Sigma(x_0..x_2)$ $\Sigma(x_0..x_3)$ $\Sigma(x_1..x_4)$ $\Sigma(x_2..x_5)$ $\Sigma(x_3..x_6)$ $\Sigma(x_4..x_7)$

$d=3$

$\Sigma(x_0..x_0)$ $\Sigma(x_0..x_1)$ $\Sigma(x_0..x_2)$ $\Sigma(x_0..x_3)$ $\Sigma(x_0..x_4)$ $\Sigma(x_0..x_5)$ $\Sigma(x_0..x_6)$ $\Sigma(x_0..x_7)$

What About Parallel Prefix?

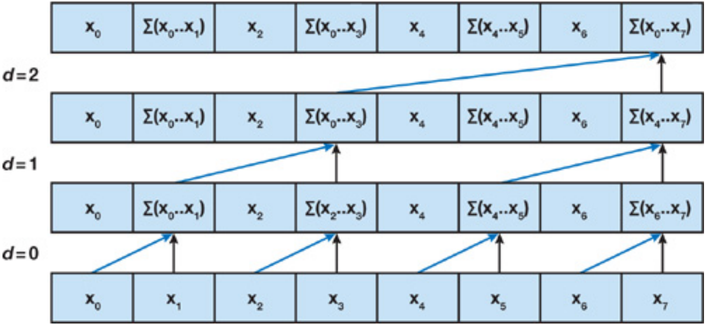


	x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7
$d=1$								
	$\Sigma(x_0..x_0)$	$\Sigma(x_0..x_1)$	$\Sigma(x_1..x_2)$	$\Sigma(x_2..x_3)$	$\Sigma(x_3..x_4)$	$\Sigma(x_4..x_5)$	$\Sigma(x_5..x_6)$	$\Sigma(x_6..x_7)$
$d=2$								
	$\Sigma(x_0..x_0)$	$\Sigma(x_0..x_1)$	$\Sigma(x_0..x_2)$	$\Sigma(x_0..x_3)$	$\Sigma(x_1..x_4)$	$\Sigma(x_2..x_5)$	$\Sigma(x_3..x_6)$	$\Sigma(x_4..x_7)$
$d=3$								
	$\Sigma(x_0..x_0)$	$\Sigma(x_0..x_1)$	$\Sigma(x_0..x_2)$	$\Sigma(x_0..x_3)$	$\Sigma(x_0..x_4)$	$\Sigma(x_0..x_5)$	$\Sigma(x_0..x_6)$	$\Sigma(x_0..x_7)$

Problem: total work $O(n \log(n))$

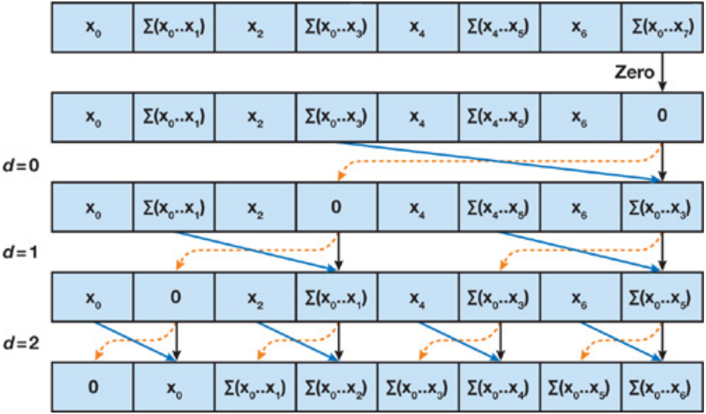
Going Work Efficient

- We resolve to our old friend :-)
- Decompose problem into two phases: up-sweep (reduction) and down-sweep (update)


x_0 | $\Sigma(x_0..x_1)$ | x_2 | $\Sigma(x_0..x_3)$ | x_4 | $\Sigma(x_4..x_5)$ | x_6 | $\Sigma(x_0..x_7)$
$d=2$
x_0 | $\Sigma(x_0..x_1)$ | x_2 | $\Sigma(x_0..x_3)$ | x_4 | $\Sigma(x_4..x_5)$ | x_6 | $\Sigma(x_4..x_7)$
$d=1$
x_0 | $\Sigma(x_0..x_1)$ | x_2 | $\Sigma(x_2..x_3)$ | x_4 | $\Sigma(x_4..x_5)$ | x_6 | $\Sigma(x_6..x_7)$
$d=0$
x_0 | x_1 | x_2 | x_3 | x_4 | x_5 | x_6 | x_7

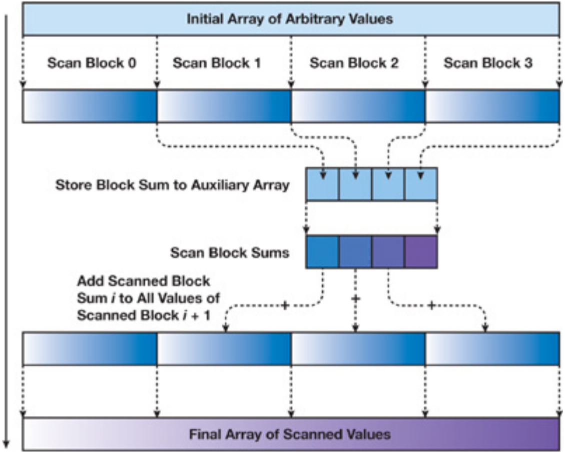
Going Work Efficient

- We resolve to our old friend :-)
- Decompose problem into two phases: up-sweep (reduction) and down-sweep (update)



x_0	$\Sigma(x_0..x_1)$	x_2	$\Sigma(x_0..x_3)$	x_4	$\Sigma(x_4..x_5)$	x_6	$\Sigma(x_0..x_7)$
x_0	$\Sigma(x_0..x_1)$	x_2	$\Sigma(x_0..x_3)$	x_4	$\Sigma(x_4..x_5)$	x_6	0
x_0	$\Sigma(x_0..x_1)$	x_2	0	x_4	$\Sigma(x_4..x_5)$	x_6	$\Sigma(x_0..x_3)$
x_0	0	x_2	$\Sigma(x_0..x_1)$	x_4	$\Sigma(x_0..x_3)$	x_6	$\Sigma(x_0..x_5)$
0	x_0	$\Sigma(x_0..x_1)$	$\Sigma(x_0..x_2)$	$\Sigma(x_0..x_3)$	$\Sigma(x_0..x_4)$	$\Sigma(x_0..x_5)$	$\Sigma(x_0..x_6)$

Working on Large Arrays


Initial Array of Arbitrary Values
Scan Block 0
Scan Block 1
Scan Block 2
Scan Block 3
Store Block Sum to Auxiliary Array
Scan Block Sums
Add Scanned Block Sum *i* to All Values of Scanned Block *i* + 1
+
+
+
Final Array of Scanned Values

For Fun

- Modify the prefix code to work with large arrays.