Lec 6 Reduction and Scan

Dong Li, Tonghua Su
School of Software
Harbin Institute of Technology

Outline

1. Concept: Reduction
2. Reduction Algorithm
3. Concept: Scan
4. Scan Algorithm

Outline

1. Concept: Reduction
2. Reduction Algorithm
3. Concept: Scan
4. Scan Algorithm

Reduction

- **What is Reduction?**
  - ✓ Reduction is a class of transformations that pass over O(N) input data and generate a O(1) result computed with a reduction operator
- **Key requirements for a reduction operator ◦ are:**
  - ✓ Binary operator
  - ✓ Commutative: $a \circ b = b \circ a$
  - ✓ Associative: $a \circ (b \circ c) = (a \circ b) \circ c$
- **Together, they mean that the elements can be re-arranged and combined in any order**

Reduction

- Parallel reduction is a fundamental building block in parallel programming
- The most common reduction operation is computing the sum of a large array of values:
  - ✓ averaging in Monte Carlo simulation
  - ✓ computing RMS (root mean square,均方根) change in finite difference computation or an iterative solver
  - ✓ computing a vector dot product in a CG(conjugate gradient, 共轭梯度) or GMRES (广义最小残量方法) iteration
- Other common reduction operations are to compute a minimum or maximum

Quiz

- 乘法是否属于归约操作?
- 除法是否属于归约操作?
- 请写出计算数组a全部元素之和的CPU代码

Outline

1. Concept: Reduction
2. Reduction Algorithm
3. Concept: Scan
4. Scan Algorithm

Approach

- **Will describe things for a summation reduction – the extension to other reductions is obvious**
- **Assuming each thread starts with one value, the approach is to**
  - ✓ local: first add the values within each thread block, to form a partial sum
  - ✓ global: then add together the partial sums from all of the blocks

Local reduction

- **The first phase is constructing a partial sum of the values within a thread block**
- **Question 1: where is the parallelism?**
  - ✓ **"Standard" summation uses an accumulator, adding one value at a time ⇒ sequential**
  - ✓ **Parallel summation of N values:**
    - first sum them in pairs to get N/2 values
    - repeat the procedure until we have only one value

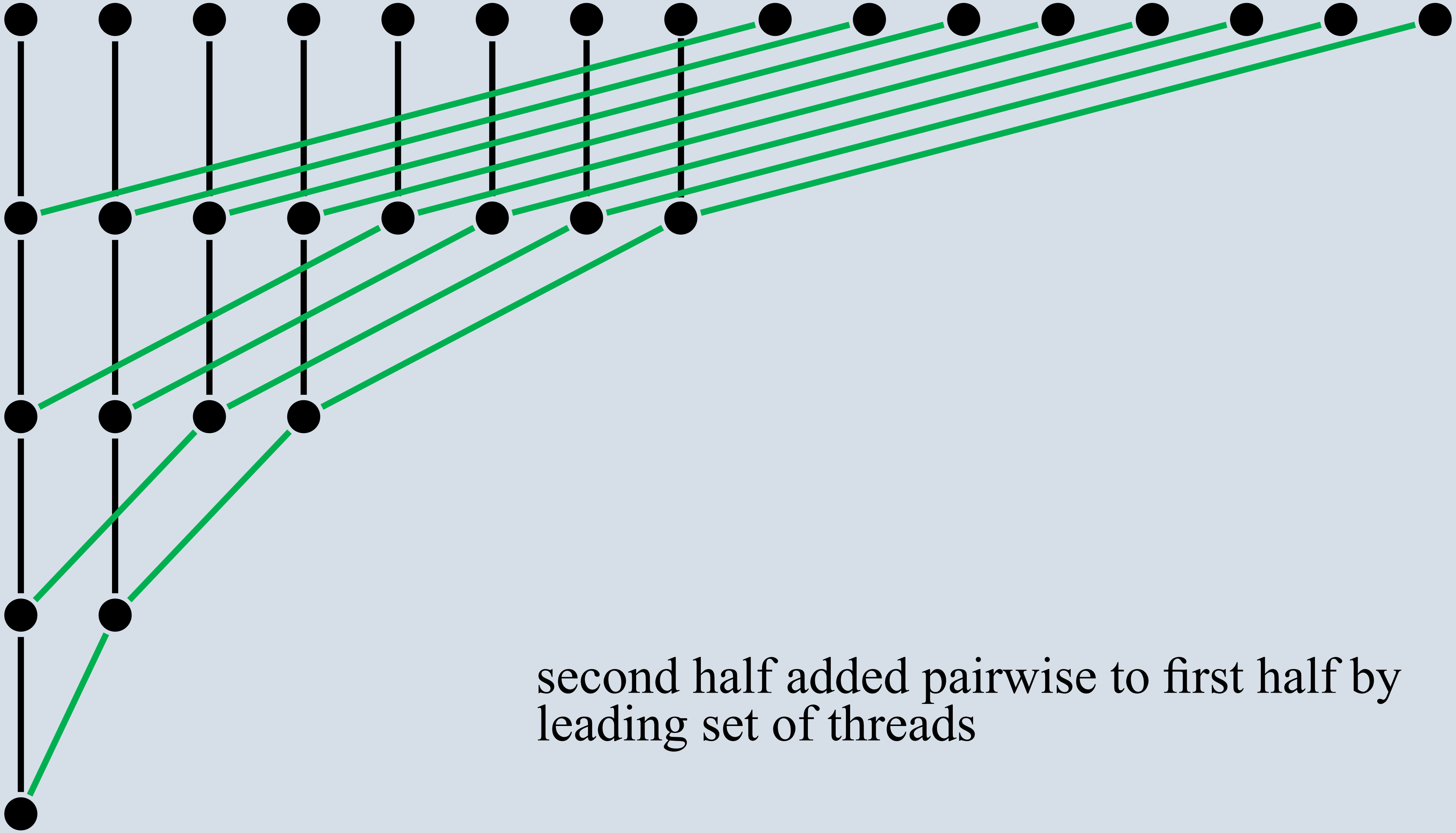
Local reduction

- **Question 2: any problems with warp divergence?**
  - ✓ **Note that not all threads can be busy all of the time:**
    - N/2 operations in first phase
    - N/4 in second
    - N/8 in third
    - etc.
  - ✓ **For efficiency, we want to make sure that each warp is either fully active or fully inactive, as far as possible**

Local reduction

- **Question 3: where should data be held?**
  - ✓ **Threads need to access results produced by other threads:**
    - global device arrays would be too slow, so use shared memory
    - need to think about synchronisation

Local reduction



second half added pairwise to first half by leading set of threads

Local reduction

```
__global__ void sum(float *d_sum,float *d_data)
{
    extern __shared__ float temp[];
    int tid = threadIdx.x;

    temp[tid] = d_data[tid+blockIdx.x*blockDim.x];

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

    if (tid==0) d_sum[blockIdx.x] = temp[0];
}
```

Local reduction

- **Note:**
  - ✓ use of dynamic shared memory – size has to be declared when the kernel is called

    Kernel<<<gridSize, blockSize, sharedMem, Stream>>>(Parameters…)
  - ✓ use of __syncthreads to make sure previous operations have completed
  - ✓ first thread outputs final partial sum into specific place for that block

Global reduction: version 1

- This version of the local reduction puts the partial sum for each block in a different entry in a global array
- These partial sums can be transferred back to the host for the final summation
- Alternatively, if there are a lot of partial sums then can re-use the same kernel to combine them

Global reduction: version 2

- Alternatively, can use the atomic lock mechanism discussed previously, and replace

```
if (tid==0) d_sum[blockIdx.x] = temp[0];
```

by

```
if (tid==0) {
    atomicAdd(d_sum,temp[0]);
}
```

```
__global__ void sum(float *d_sum,float *d_data)
{
    extern __shared__ float temp[];
    int tid = threadIdx.x;
    int idx = blockIdx.x*blockDim.x+tid;
    temp[tid] = d_data[idx];
    for (int d=blockDim.x>>1; d>=1; d>>=1) {
        __syncthreads();
        if (tid<d) temp[tid] += temp[tid+d];
    }
    if (tid==0) atomicAdd(d_sum,temp[0]);
}
```

Extensions

Additional Reading

- **Mark Harris's Slides**
  - ✓ **"Optimizing Parallel Reduction in CUDA"**
  - ✓ **Accompanying sample codes**
- **COOK Sect. 9.5.2**
- **WILT Ch12**
- **Reduction on Kepler**
  - ✓ **http://devblogs.nvidia.com/parallelforall/faster-parallel-reductions-kepler/**

Lab 4 Reduction Algorithm

- 在样例程序上，按实验指导书修改Reduction算法
  - ✓ 实现任意block大小的reduction
  - ✓ 实现多个block的reduction
  - ✓ 基于reduction算法计算实验3.2 laplace3d中的均方根
  - ✓ 基于reduction算法计算实验2.2中的距离

Outline

1. Concept: Reduction
2. Reduction Algorithm
3. Concept: Scan
4. Scan Algorithm

Scan operation

- Scan is a kind of transformation that pass over N input data and generate N results using a binary associative operator $\oplus$
  - ✓ inclusive scan: $[a_0,a_1,\ldots,a_{N-1}] \Rightarrow [a_0,a_0\oplus a_1,\ldots,(a_0\oplus a_1\oplus\ldots\oplus a_{N-1})]$
  - ✓ exclusive scan: $[a_0,a_1,\ldots,a_{N-1}] \Rightarrow [id_{\oplus},a_0,a_0\oplus a_1,\ldots,(a_0\oplus a_1\oplus\ldots\oplus a_{N-2})]$
- Prefix Sum is a special scan: Given an input vector u_i, $i = 0,\ldots,I-1$, the objective of a scan operation is to compute

$$v_j = \sum_{i<j} u_i \quad \text{for all } j < I.$$

Input:	6	4	16	10	16	14	2	8
Output:	6	10	26	36	52	66	68	76

Scan operation

- **Why is this important?**
  - ✓ a key part of many sorting routines (Quicksort , Radix sort)
  - ✓ arises also in: Sparse matrix-vector multiplication, Minimum spanning tree construction, particle filter methods ......
  - ✓ related to solving long recurrence equations:

$$v_{n+1} = (1-\lambda_n)v_n + \lambda_n u_n$$

  - ✓ a good example that looks impossible to parallelise

Scan operation in Quicksort

- **Quicksort procedure**
  - ✓ **Pick a pivot element**
  - ✓ **Partition all the data into:**
    - A. the elements less than the pivot (Scan operation)
    - B. the pivot
    - C. the elements greater than the pivot (Scan operation)
  - ✓ **Recursively sort A and C**

Quiz

- 请写出可实现前缀求和的CPU串行代码

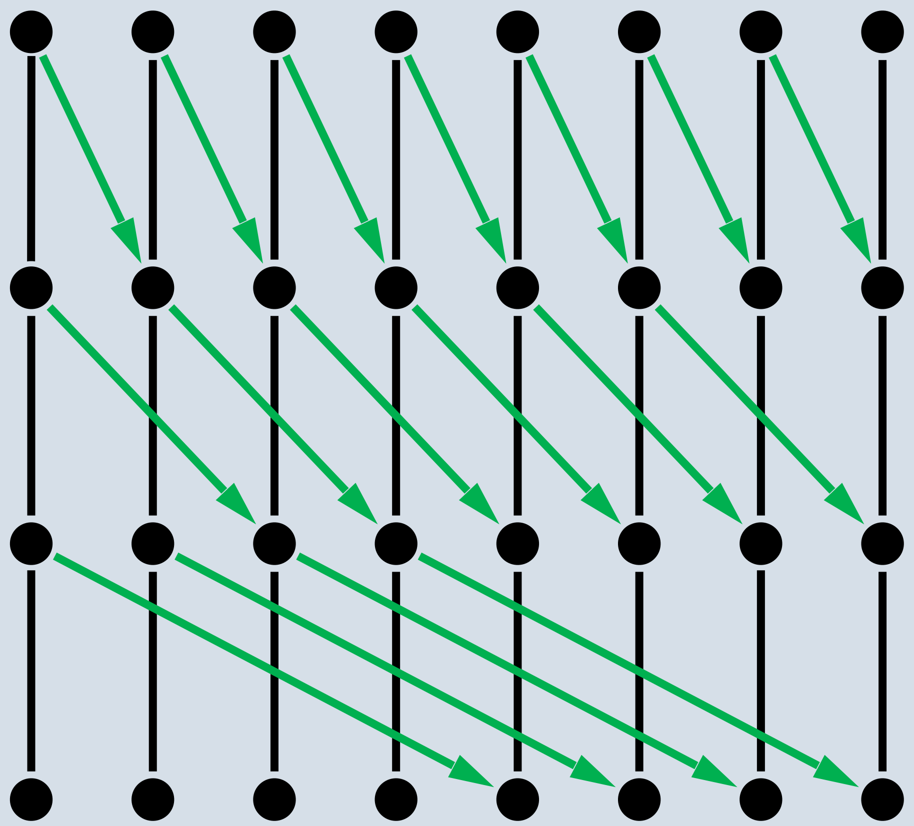
Outline

1. Concept: Reduction
2. Reduction Algorithm
3. Concept: Scan
4. Scan Algorithm

Scan operation

- **Similar to the global reduction, the top-level strategy is**
  - ✓ **perform local scan within each block**
  - ✓ **add on sum of all preceding blocks**
- **Two approaches to the local scan, both similar to the local reduction but in slightly different ways**
  - ✓ **first approach: very simple but O(N logN) operations**
  - ✓ **second approach:**
    - similar to binary tree summation but with both downward and upward passes
    - O(N) operations so slightly more efficient

Local scan: version 1



- after n passes, each sum has local plus preceding 2^{n-1} values
- $\log_2 N$ passes, and O(N) operations per pass $\Rightarrow$ O(N logN) operations in total

Local scan: version 1

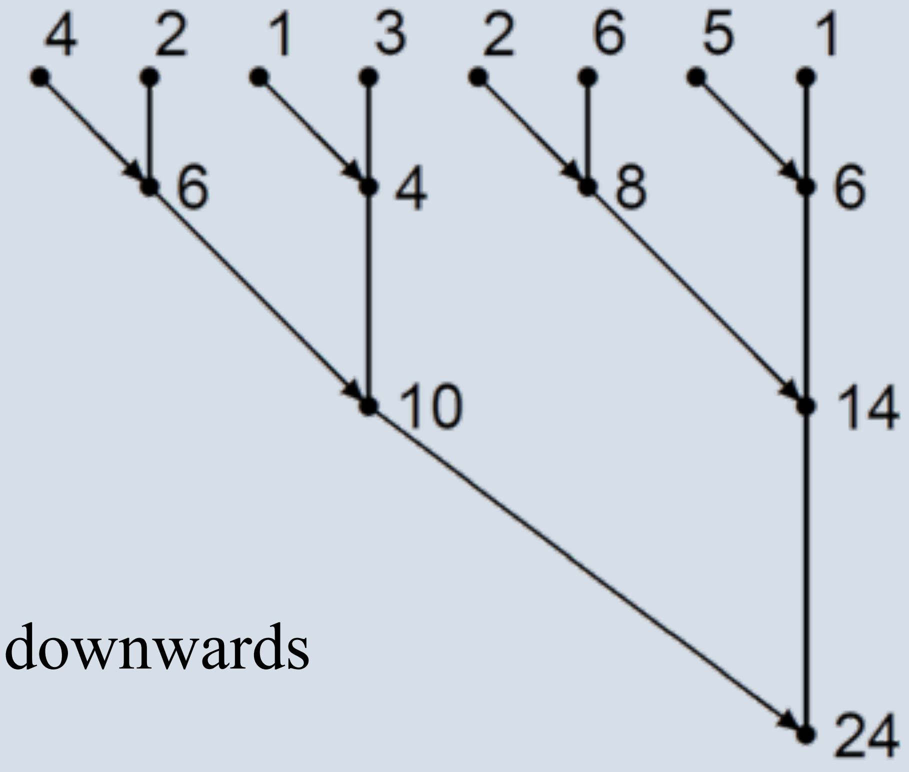
```
__global__ void scan(float *d_sum,float *d_data)
{
    extern __shared__ float temp[];
    int tid = threadIdx.x;
    int idx= tid+blockIdx.x*blockDim.x;
    temp[tid] = d_data[idx];

    for (int d=1; d<blockDim.x; d<<=1) {
        __syncthreads();
        float temp2 = (tid >= d) ? temp[tid-d] : 0;
        __syncthreads();
        temp[tid] += temp2;
    }
    ...
}
```

Local scan: version 1

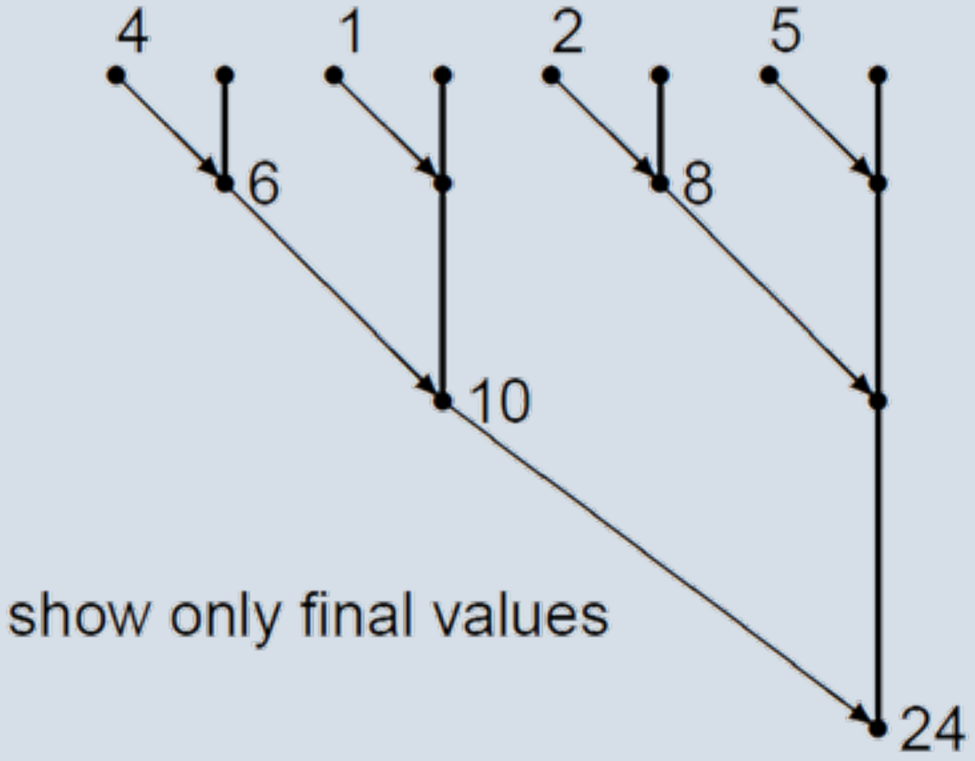
- **Notes:**
  - ✓ code due to Oswaldo Cadenas at Univ. of Reading, similar to other code by Mark Harris
  - ✓ much simpler than version 2
  - ✓ at most only 40% slower
  - ✓ increment is set to zero if no element to the left
  - ✓ both __syncthreads(); are needed

Local scan: version 2

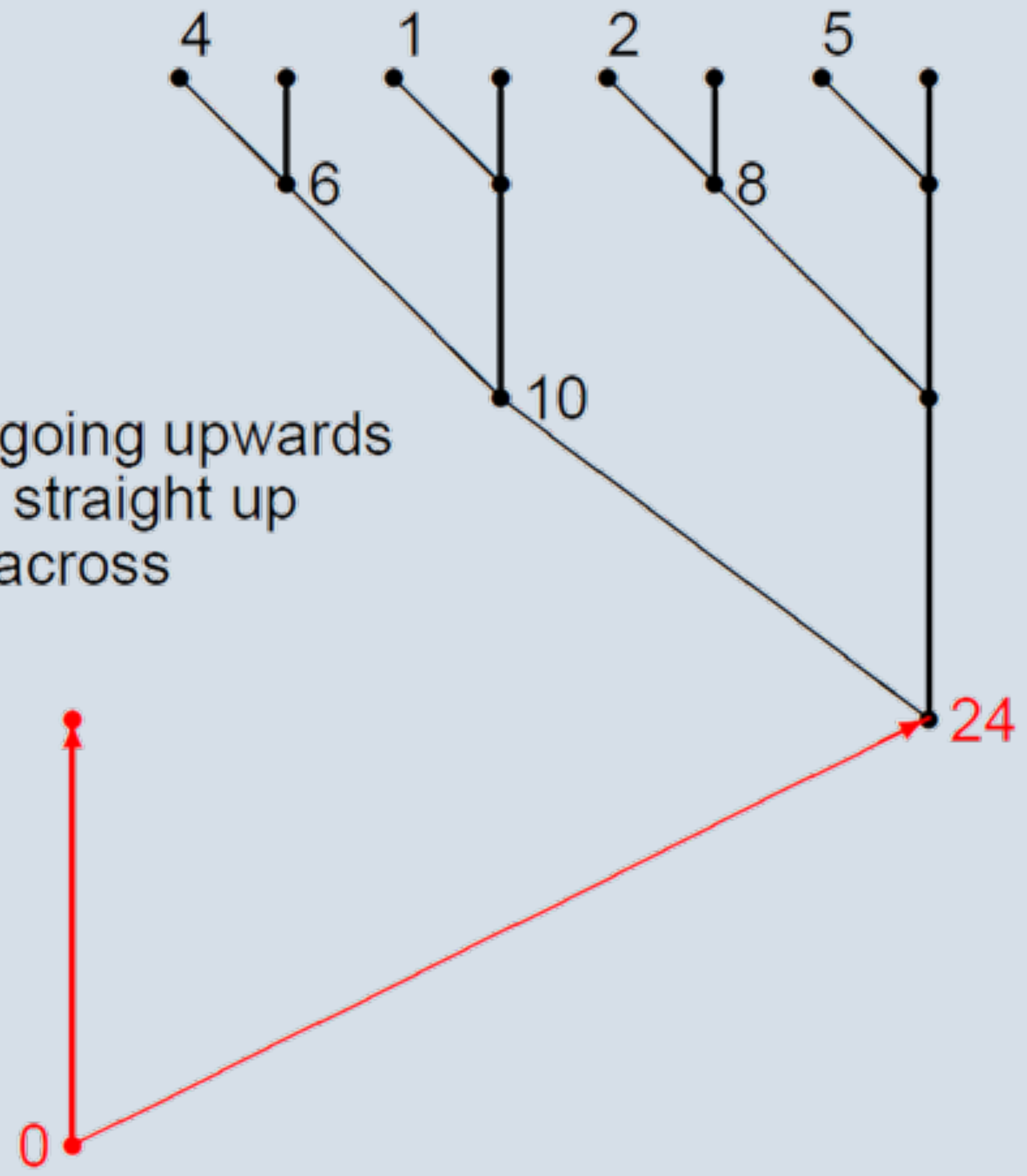


sum downwards

Local scan: version 2

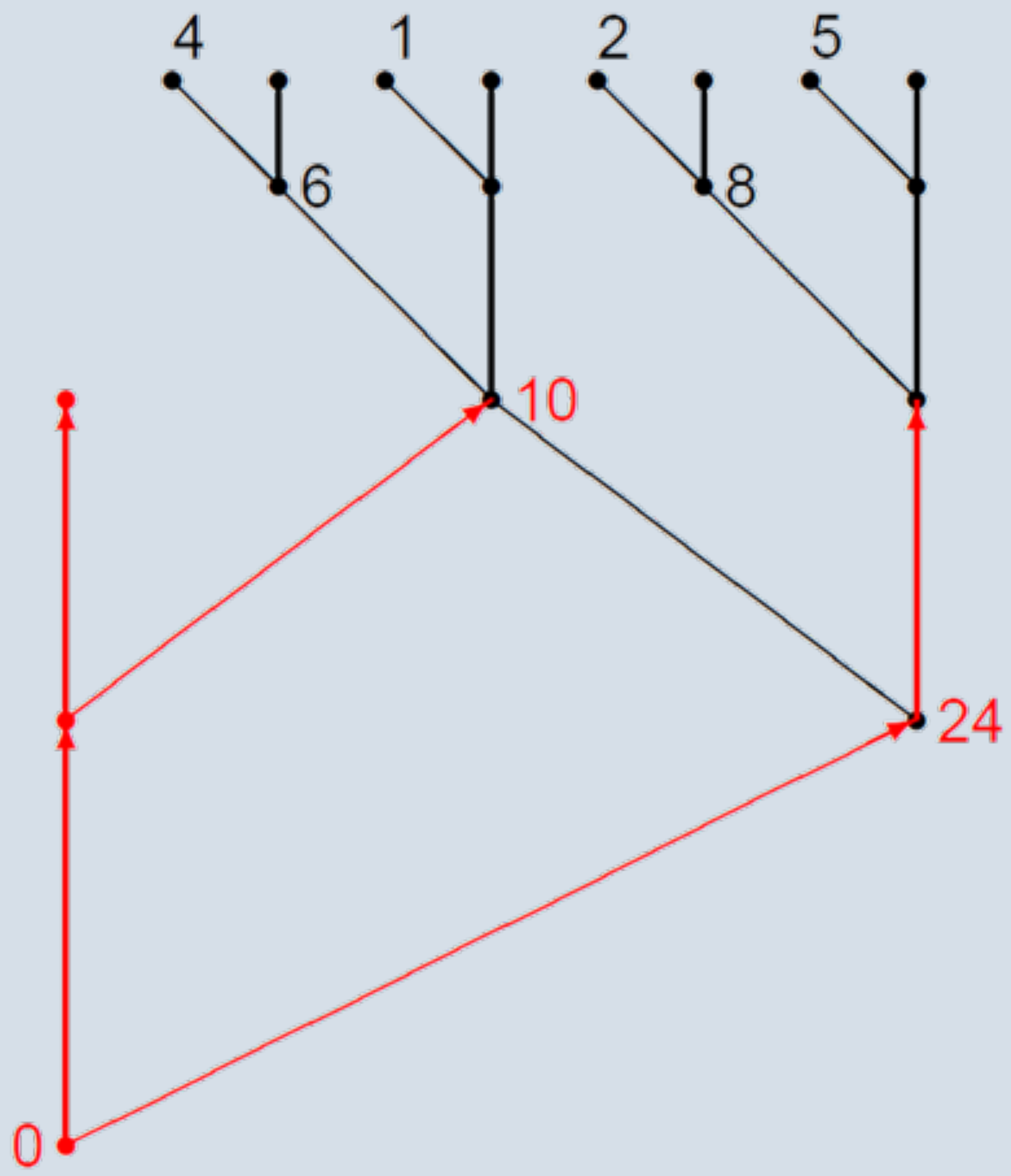


Local scan: version 2

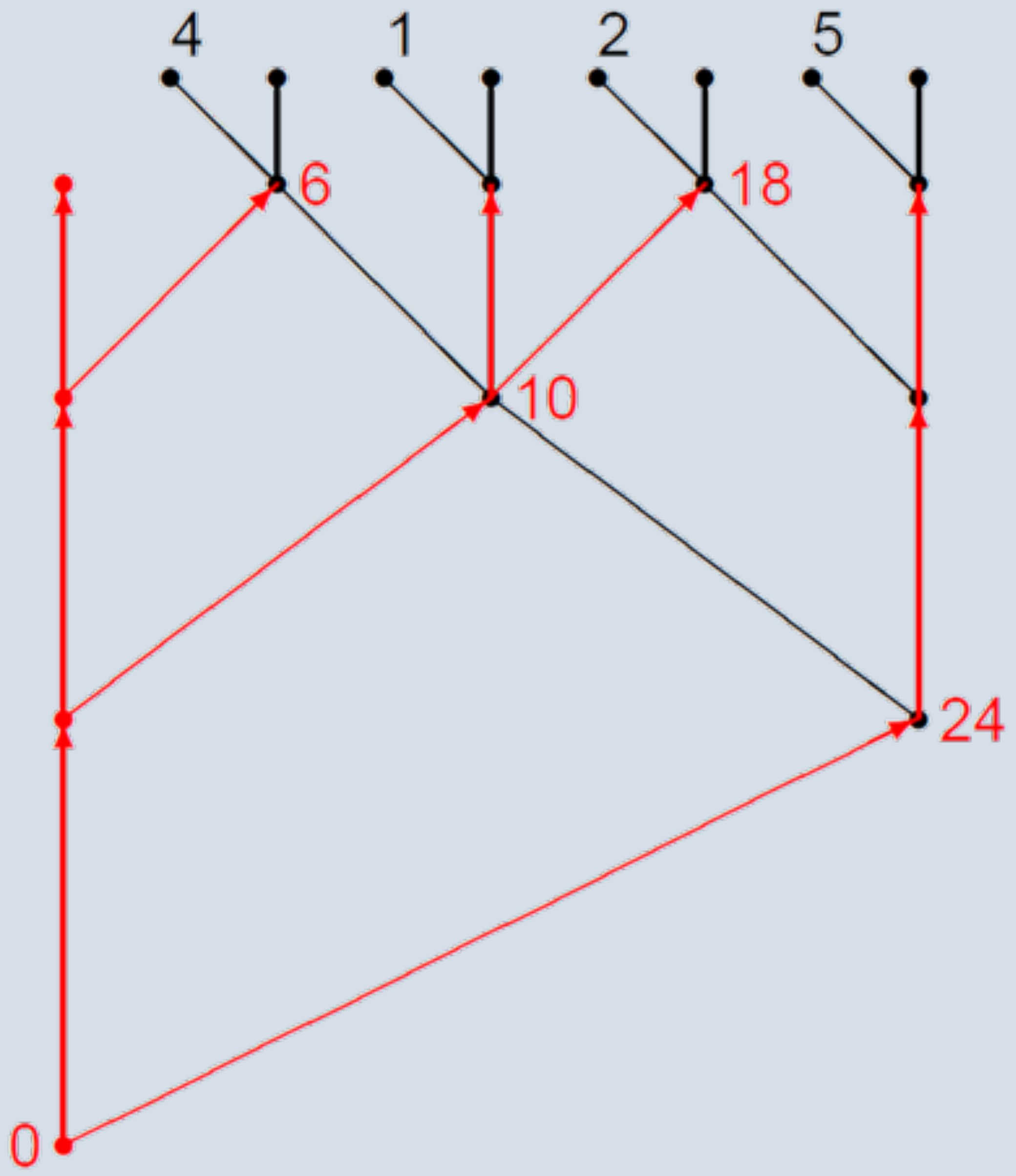


start going upwards
carry straight up
sum across

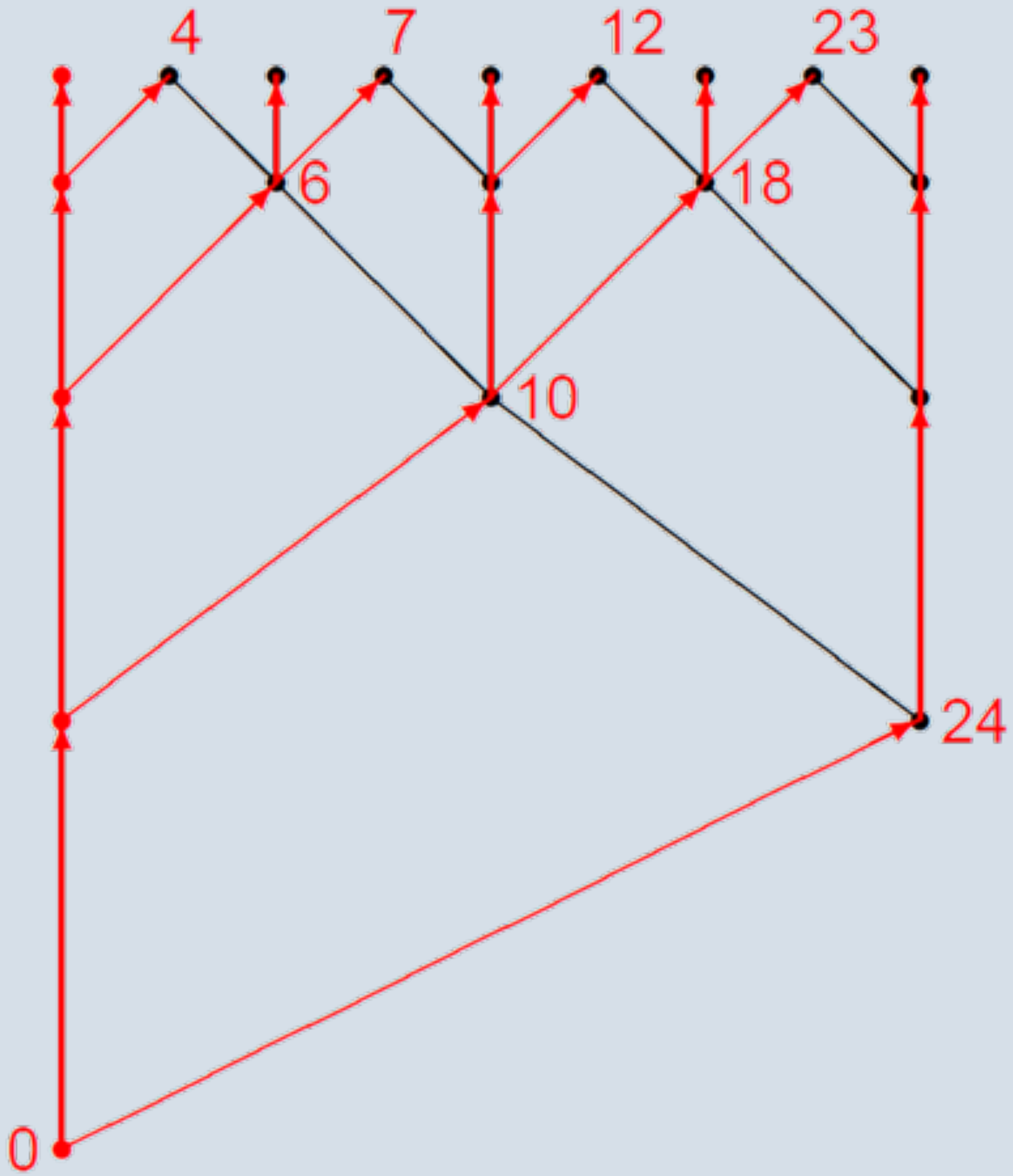
Local scan: version 2



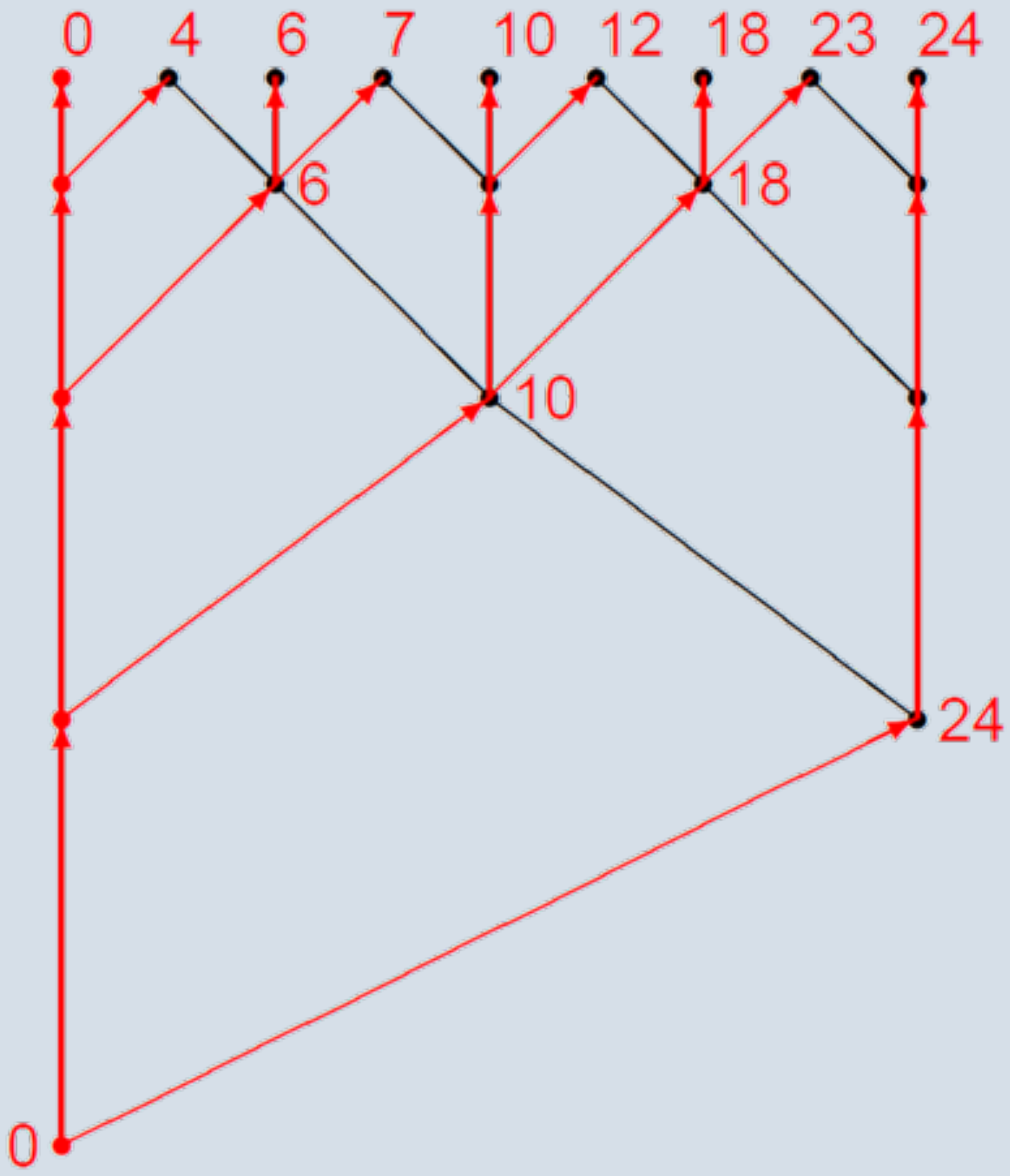
Local scan: version 2



Local scan: version 2



Local scan: version 2



Local scan: version 2

- **Notes:**
  - ✓ similar to code by Mark Harris
  - ✓ not very easy to follow, maybe best to go through the example above to check it's doing the right thing
  - ✓ in the practical, the code puts the local scan values back in the global device array
  - ✓ however, really we need to complete the process by performing a global scan at the higher level

Global scan: version 1

- To complete the global scan there are two options
- First alternative:
  - ✓ use one kernel to do local scan and compute partial sum for each block
  - ✓ use host code to perform a scan of the partial sums
  - ✓ use another kernel to add sums of preceding blocks

Global scan: version 2

- **Second alternative – do it all in one kernel call**
- **However, this needs the sum of all preceding blocks to**
  - ✓ add to the local scan values in version 1
  - ✓ replace initial value 0 at the start of the upward sweep in version 2
- **Problem: blocks are not necessarily processed in order, so could end up in deadlock waiting for results from a block which doesn't get a chance to start.**
- **Solution: use atomic increments**

Global scan: version 2

- Declare a global device variable

```
__device__ int my_block_count = 0;
```

and at the beginning of the kernel code use

```
__shared__ unsigned int my_blockId;
if (threadIdx.x==0) {
    my_blockId = atomicInc( &my_block_count,(unsigned int) -1 );
}
__syncthreads();
```

- which returns the old value of my_block_count and increments it, all in one operation. The -1 cast ensures atomicInc always increments the counter.
- This gives us a way of launching blocks in strict order.

Global scan: version 2

- In the second approach to the global scan, the kernel code does the following:
  - ✓ get in-order block ID
  - ✓ do downward pass
  - ✓ wait until another global counter my_block_count2 shows that preceding block has computed the sum of the blocks so far
- get the sum of blocks so far, increment the sum with the local partial sum, then increment my_block_count2
- do upwards pass and store the results

Additional Reading

- CUDA Samples
  - ✓ Accompanying sample codes
- WILT Ch13

Quiz

- 选做：根据本节的讲解，是写出具有完整功能的scan内核