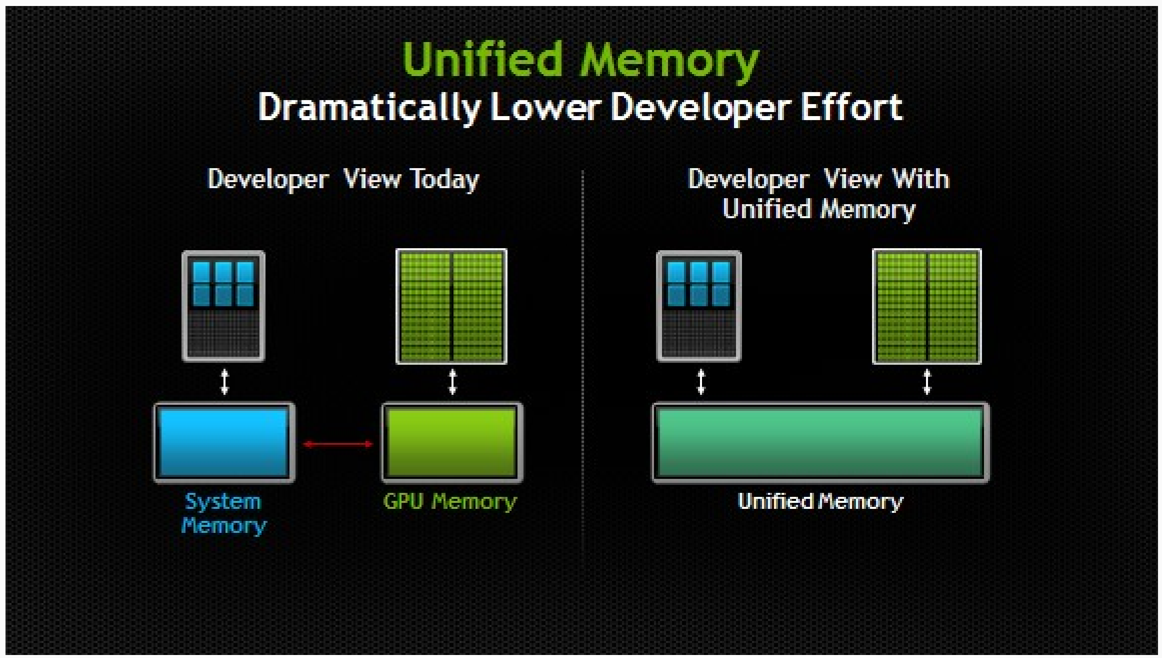
Applications: Scan/Histo/Conv

Leiming Yu
yu.lei@husky.neu.edu

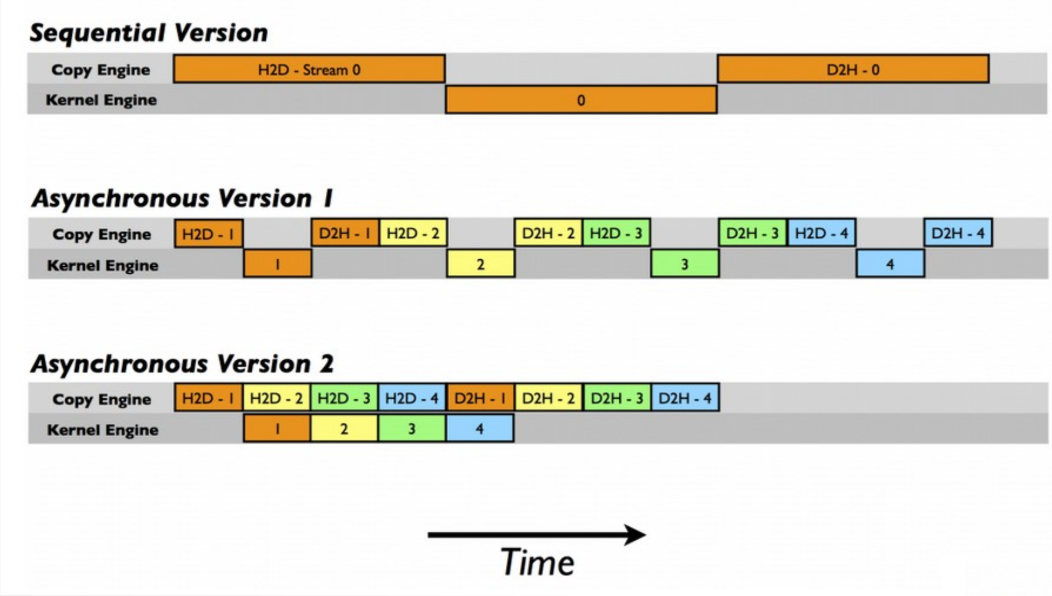
Topics

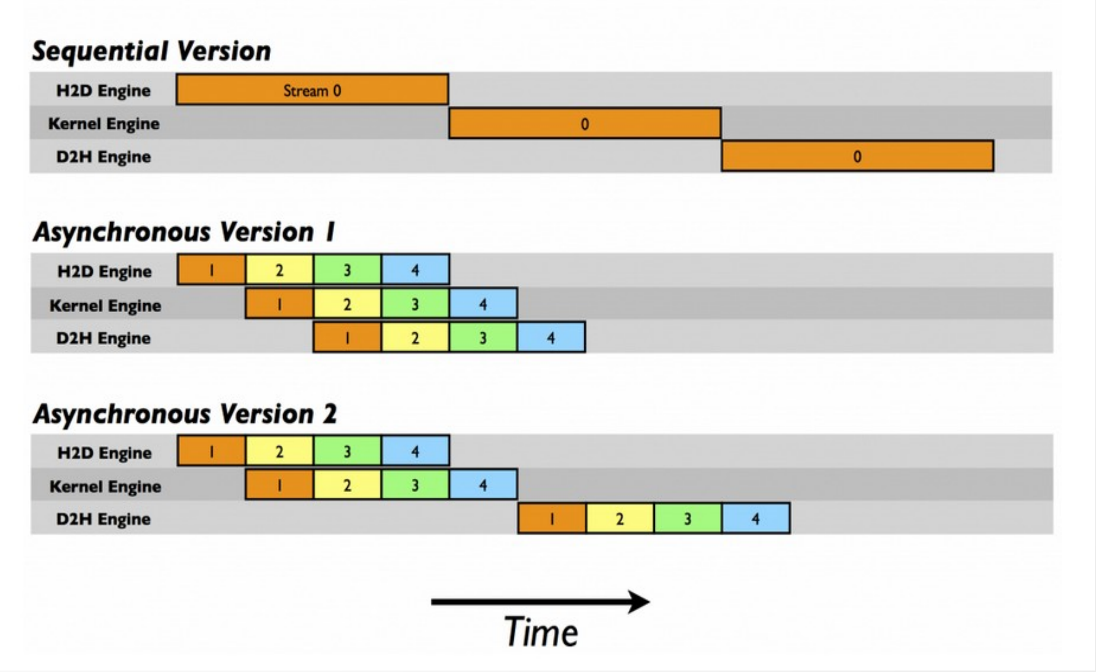
- Prefix Sum
- Histogram
- Convolution

Catch up


Unified Memory
Dramatically Lower Developer Effort
Developer View Today
Developer View With Unified Memory
System Memory
GPU Memory
Unified Memory

Catch up


Sequential Version
Copy Engine
H2D - Stream 0
D2H - 0
Kernel Engine
0
Asynchronous Version 1
Copy Engine
H2D - 1
D2H - 1
H2D - 2
D2H - 2
H2D - 3
D2H - 3
H2D - 4
D2H - 4
Kernel Engine
1
2
3
4
Asynchronous Version 2
Copy Engine
H2D - 1
H2D - 2
H2D - 3
H2D - 4
D2H - 1
D2H - 2
D2H - 3
D2H - 4
Kernel Engine
1
2
3
4
Time


Sequential Version
H2D Engine
Stream 0
Kernel Engine
0
D2H Engine
0
Asynchronous Version 1
H2D Engine
1
2
3
4
Kernel Engine
1
2
3
4
D2H Engine
1
2
3
4
Asynchronous Version 2
H2D Engine
1
2
3
4
Kernel Engine
1
2
3
4
D2H Engine
1
2
3
4
Time

Catch up

```
template <unsigned int blockSize>
__device__ void warpReduce(volatile int *sdata, unsigned int tid) {
   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];
}
template <unsigned int blockSize>
__global__ void reduce6(int *g_idata, int *g_odata, unsigned int n) {
   extern __shared__ int sdata[];
   unsigned int tid = threadIdx.x;
   unsigned int i = blockIdx.x*(blockSize*2) + tid;
   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();

   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) warpReduce(sdata, tid);
   if (tid == 0) g_odata[blockIdx.x] = sdata[0];
}
```

Final Optimized Kernel

Parallel Prefix Sum (Scan)

What is prefix sum?

- Definition:

  The all-prefix-sums operation takes a binary associative operator $\oplus$ with identity I, and an array of n elements

  $$[a_0, a_1, \ldots, a_{n-1}]$$

  and returns the ordered set

  $$[I, a_0, (a_0 \oplus a_1), \ldots, (a_0 \oplus a_1 \oplus \ldots \oplus a_{n-2})].$$

- Example:

  if $\oplus$ is addition, then scan on the set

  [3 1 7 0 4 1 6 3]

  returns the set

  [0 3 4 11 11 15 16 22]

Exclusive scan: last input element is not included in the result

Parallel Prefix Sum (Scan)

Where it is applied?

- Useful in implementation of several parallel algorithms:
  - radix sort
  - quicksort
  - String comparison
  - Lexical analysis
  - Stream compaction
  - Polynomial evaluation
  - Solving recurrences
  - Tree operations
  - Histograms
  - Etc.

Assigning camp slots

Assigning farmer market space

Allocating memory to parallel threads

Allocating memory buffer for communication channels

Parallel Prefix Sum (Scan)

A Naïve Inclusive Parallel Scan

Assign one thread to calculate each y element

Have every thread to add up all x elements needed for the y element

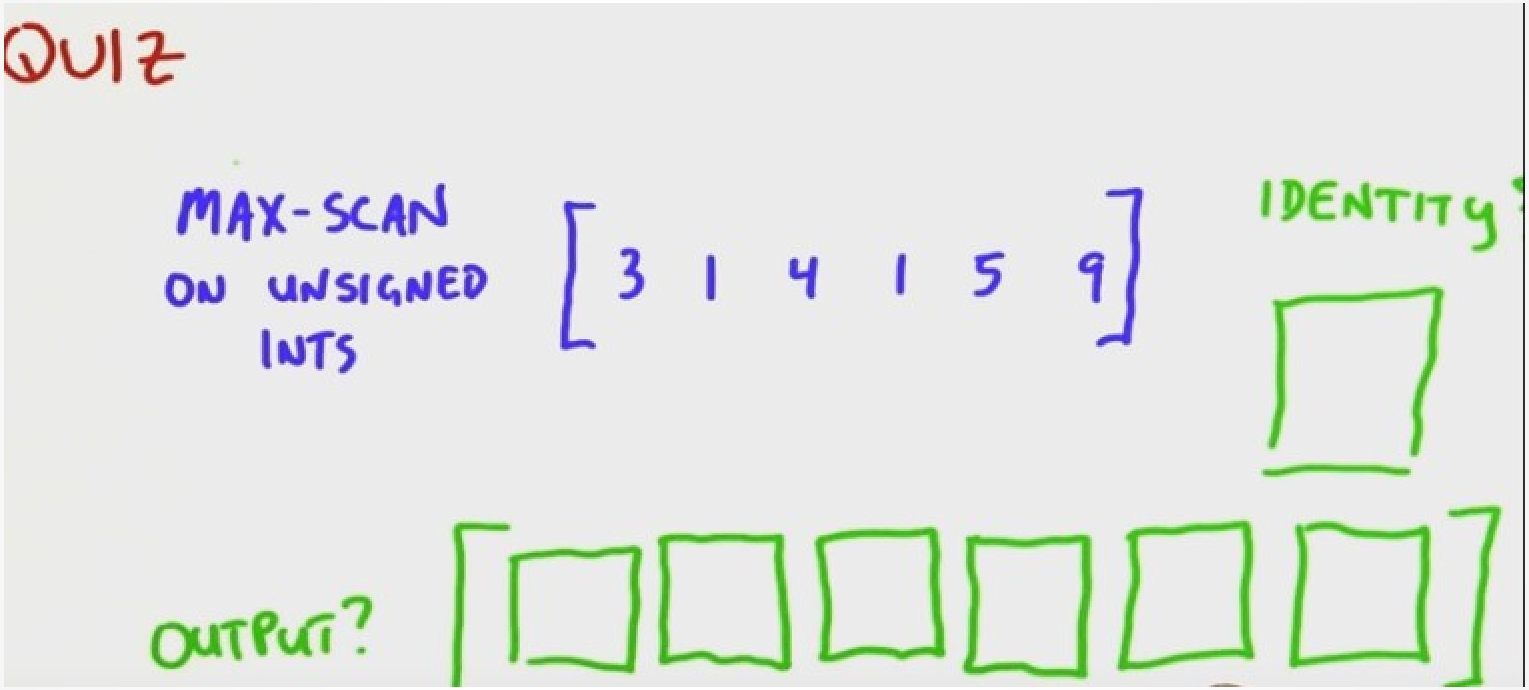
$$y0 = x0$$

$$y1 = x0 + x1$$

$$y2 = x0 + x1 + x2$$

"Parallel programming is easy as long as you do not care about performance."

Parallel Prefix Sum (Scan)

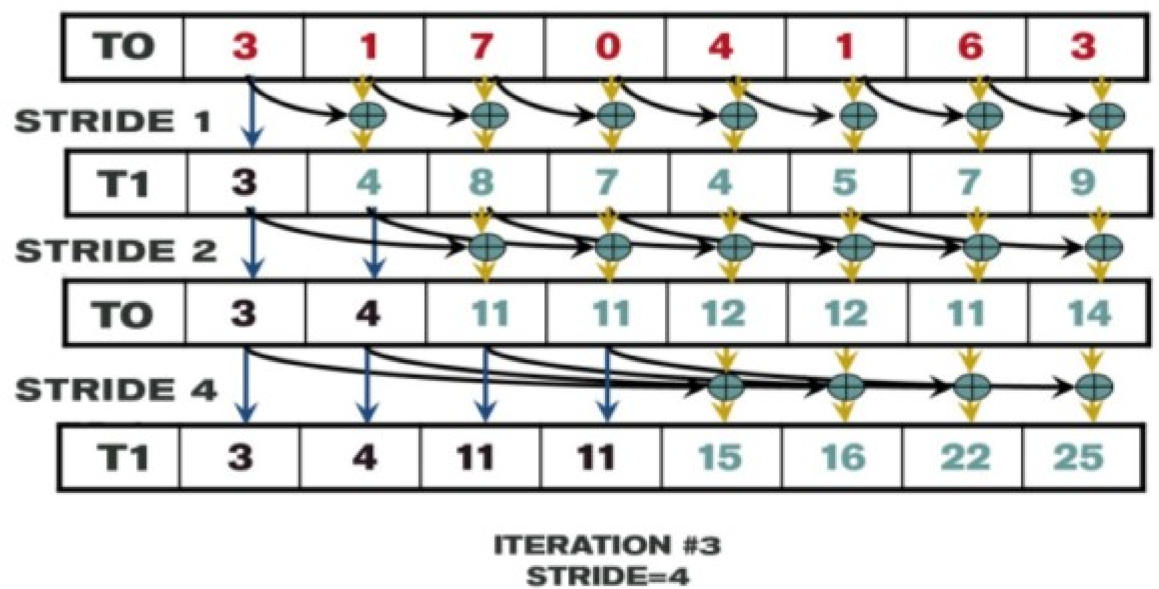


Be aware of inclusive and exclusive scan.

Parallel Prefix Sum (Scan)

Compared to naïve version, a slight better version - **Hillis Steele Scan**.

1. ...
2. Iterate log(n) times: Threads stride to n: Add pairs of elements stride elements apart. Double stride at each iteration. (note must double buffer shared memory arrays)
3. Write output from shared memory to device memory



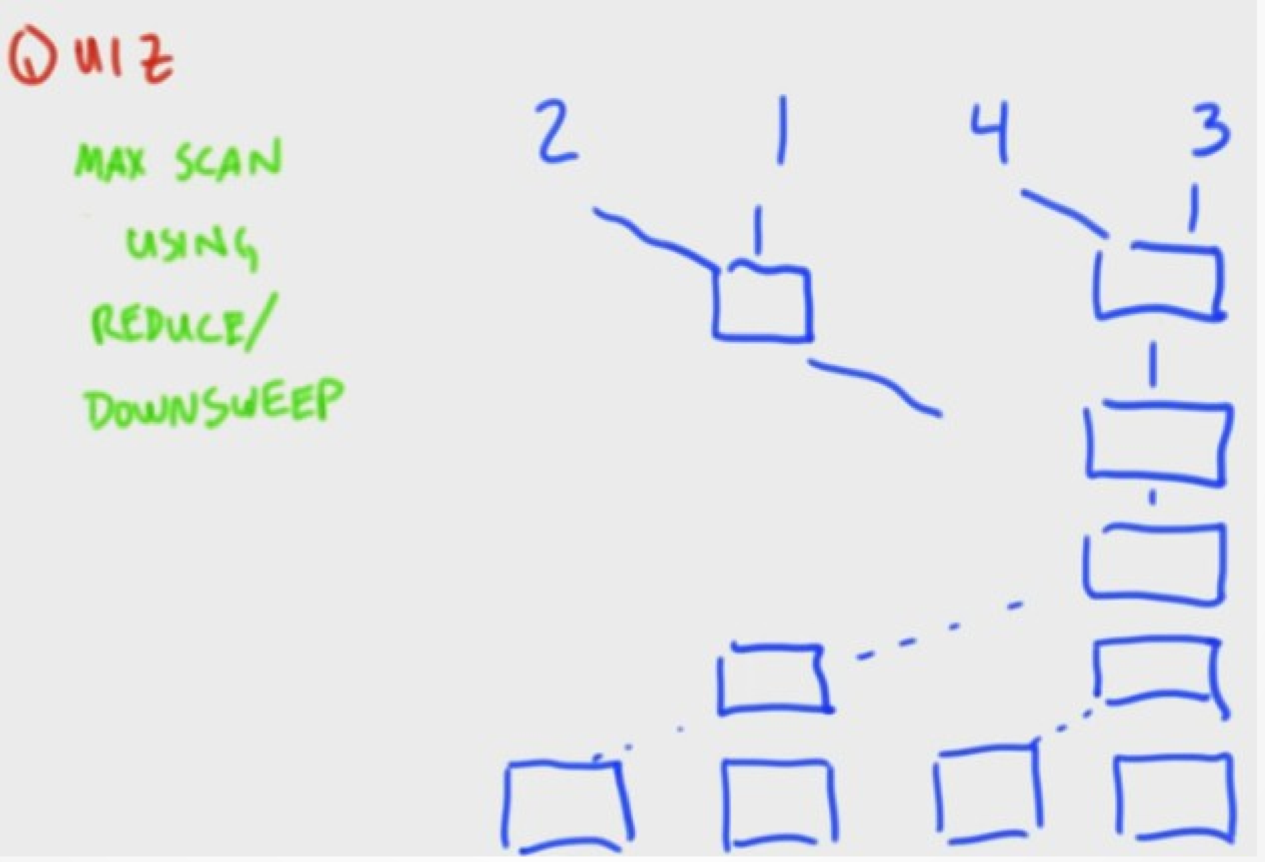
- This scan algorithm is not that work efficient
  - Sequential scan algorithm does *n-1* adds
  - A factor of log(n) might hurt: 20x more work for 10^6 elements!

Parallel Prefix Sum (Scan)

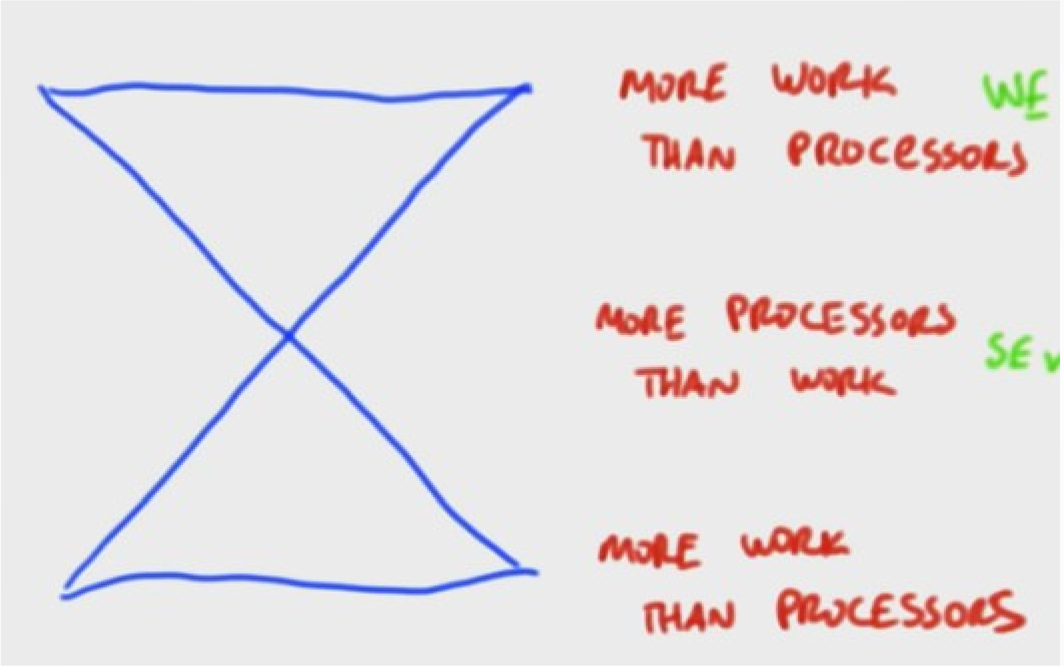
Work-efficient: **Blelloch Scan**

Parallel Prefix Sum (Scan)

Work-efficient: **Blelloch Scan**



Parallel Prefix Sum (Scan)

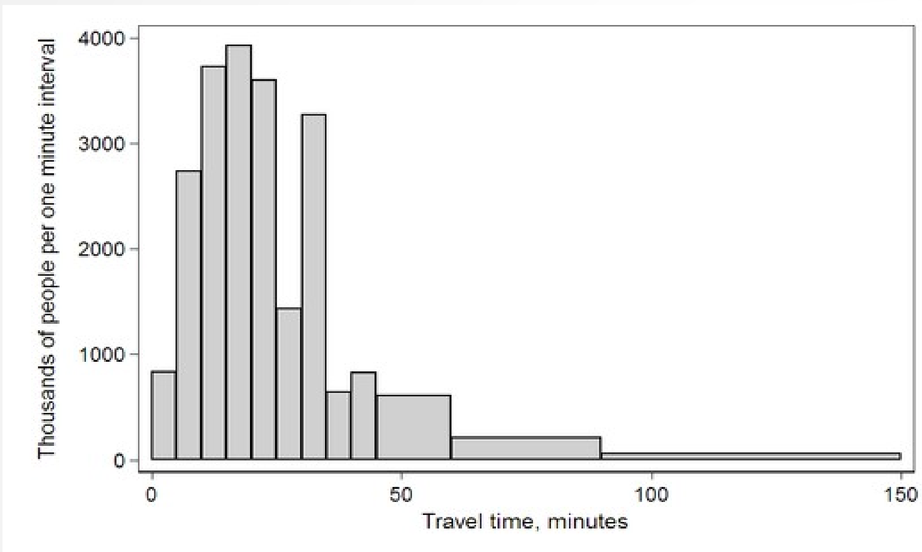


Parallel Prefix Sum (Scan)

Topics

- Prefix Sum
- ***Histogram***
- Convolution

Histogram

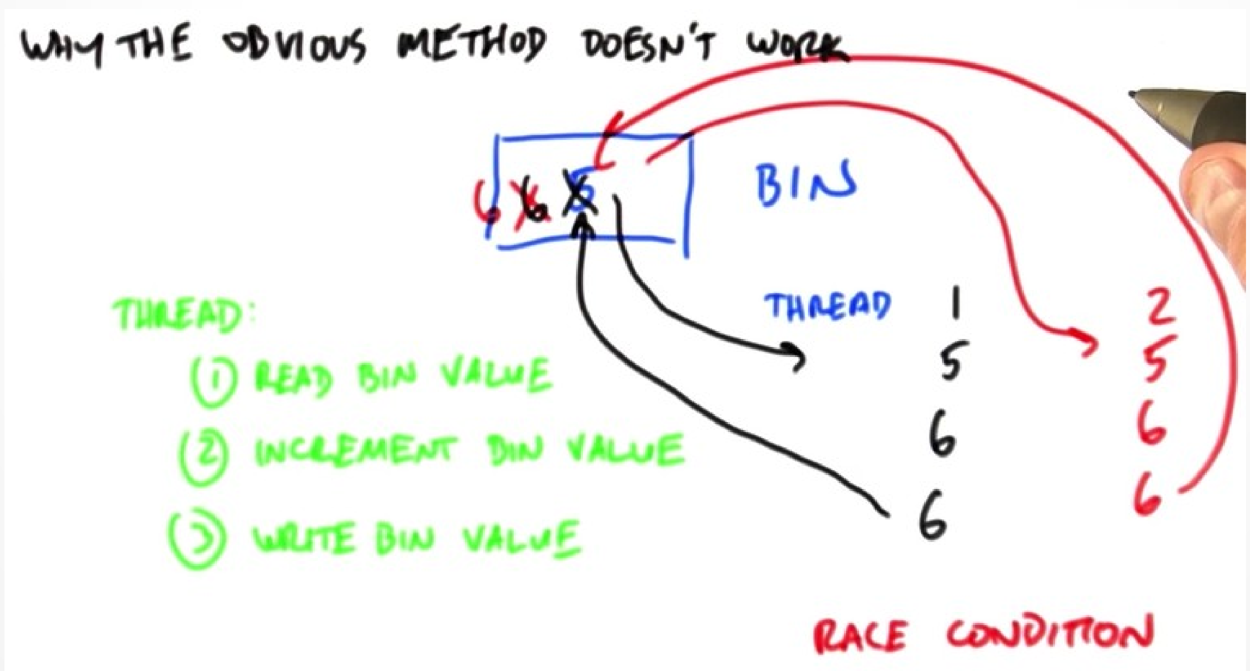


SERIAL ALGORITHM: HISTOGRAM

```
for (i=0; i< BIN_COUNT; i++)
    result[i]=0;
for (i=0; i< measurements.size(); i++)
    result[computeBin(measurements[i])]++;
```

Histogram

```
__global__ void naive_histo(int *d_bins, const int *d_in, const int BIN_COUNT)
{
    int myId = threadIdx.x + blockDim.x * blockIdx.x;
    int myItem = d_in[myId];
    int myBin = myItem % BIN_COUNT;
    d_bins[myBin]++;
}
```

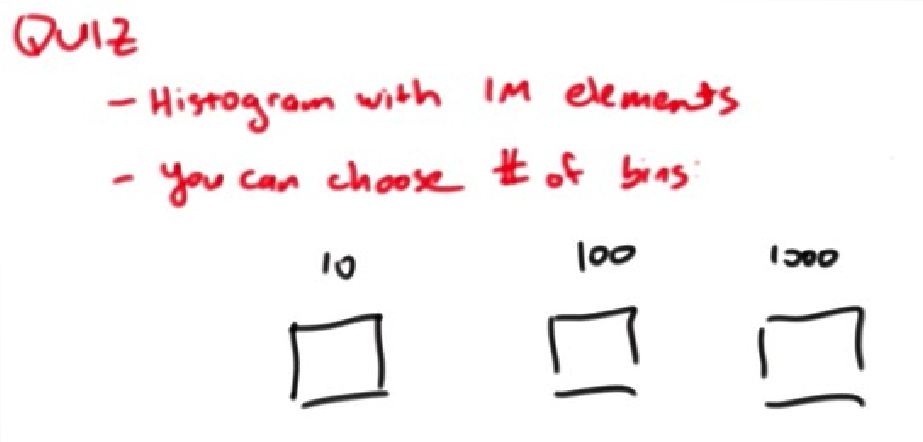


Histogram

Method 1 : Atomics

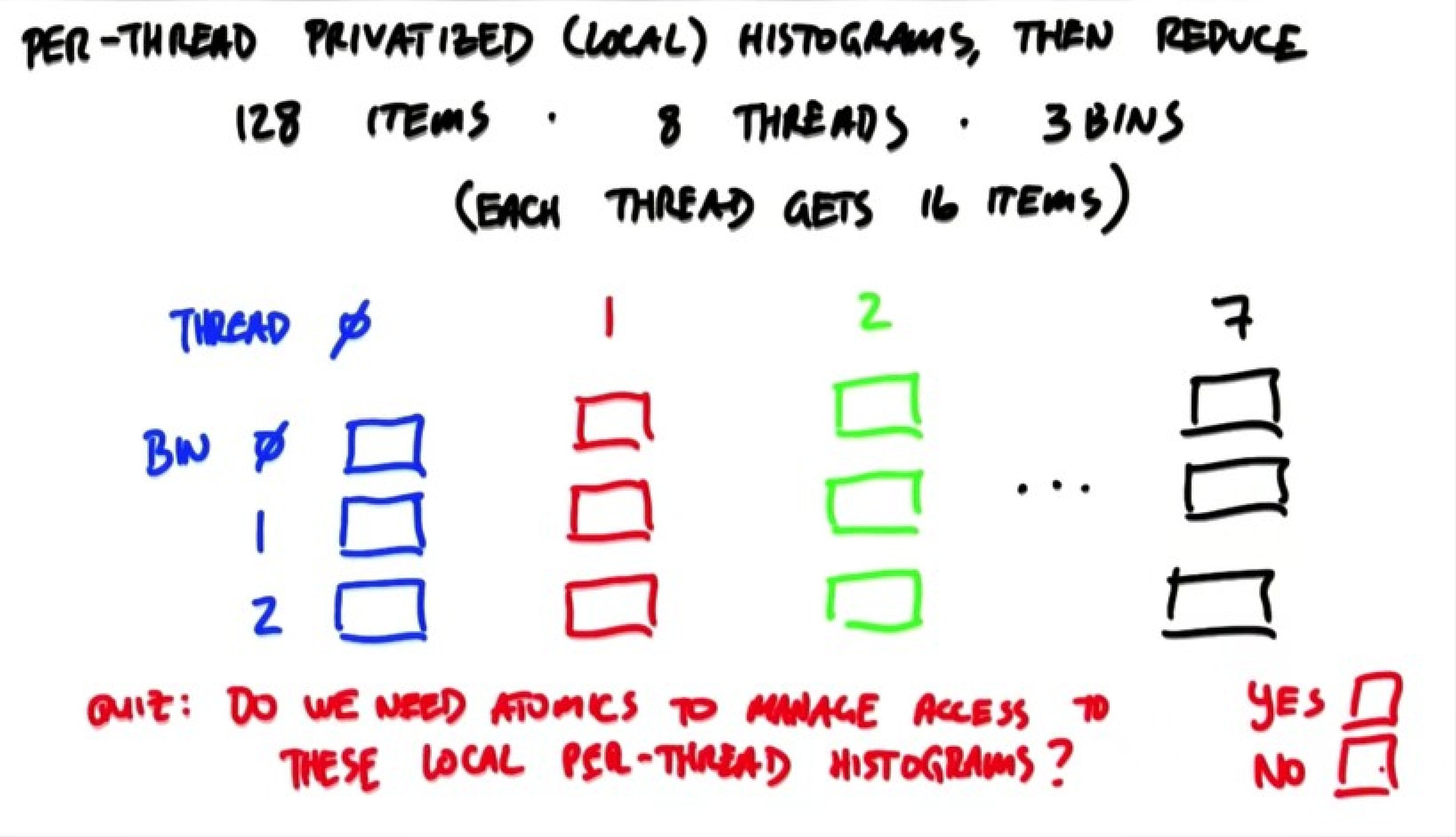
RAW hazard

```
__global__ void simple_histo(int *d_bins, const int *d_
{
    int myId = threadIdx.x + blockDim.x * blockIdx.x;
    int myItem = d_in[myId];
    int myBin = myItem % BIN_COUNT;
    atomicAdd(&(d_bins[myBin]), 1);
}
```



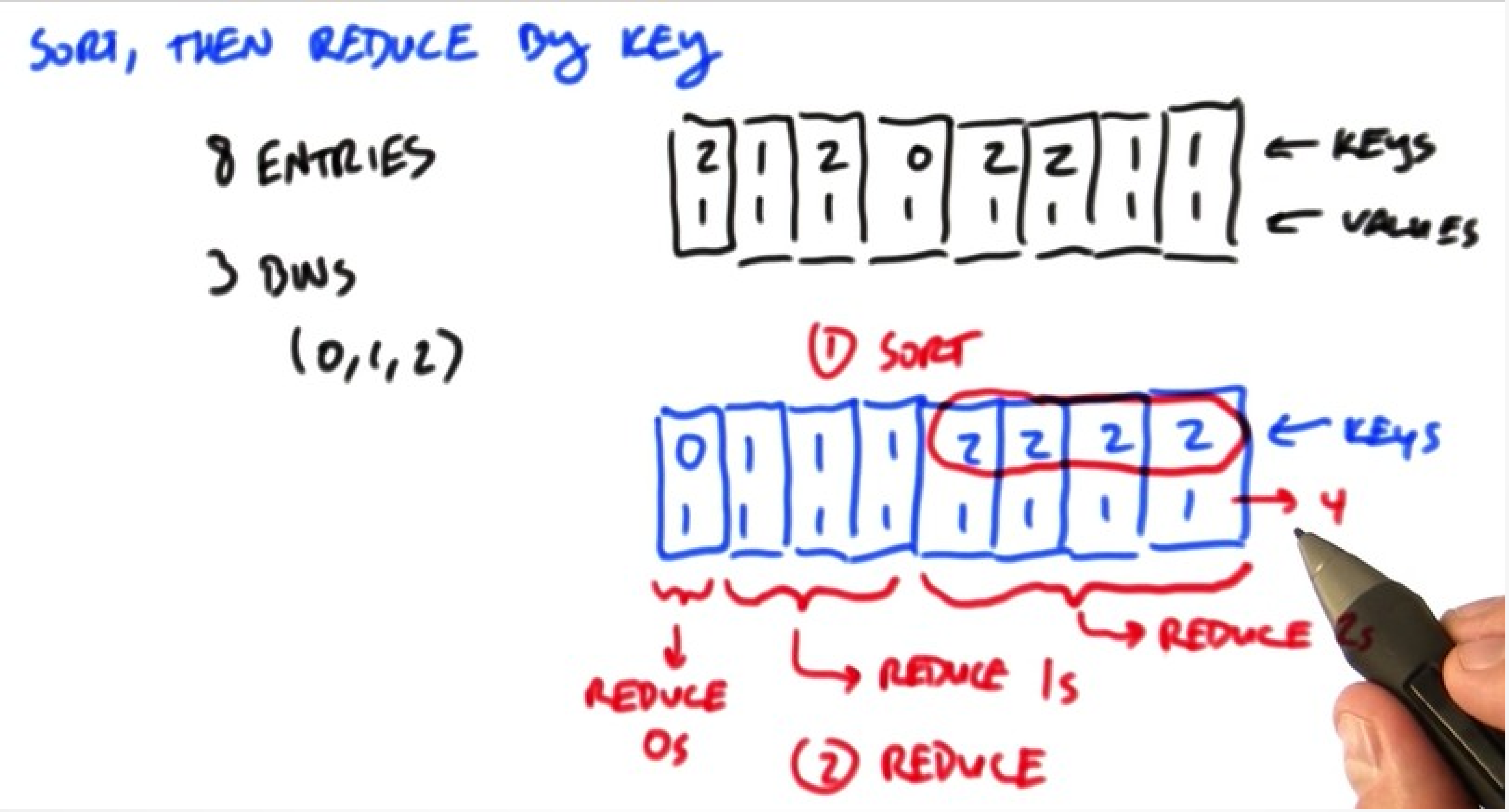
Histogram

Method 2 : local histogram + reduction

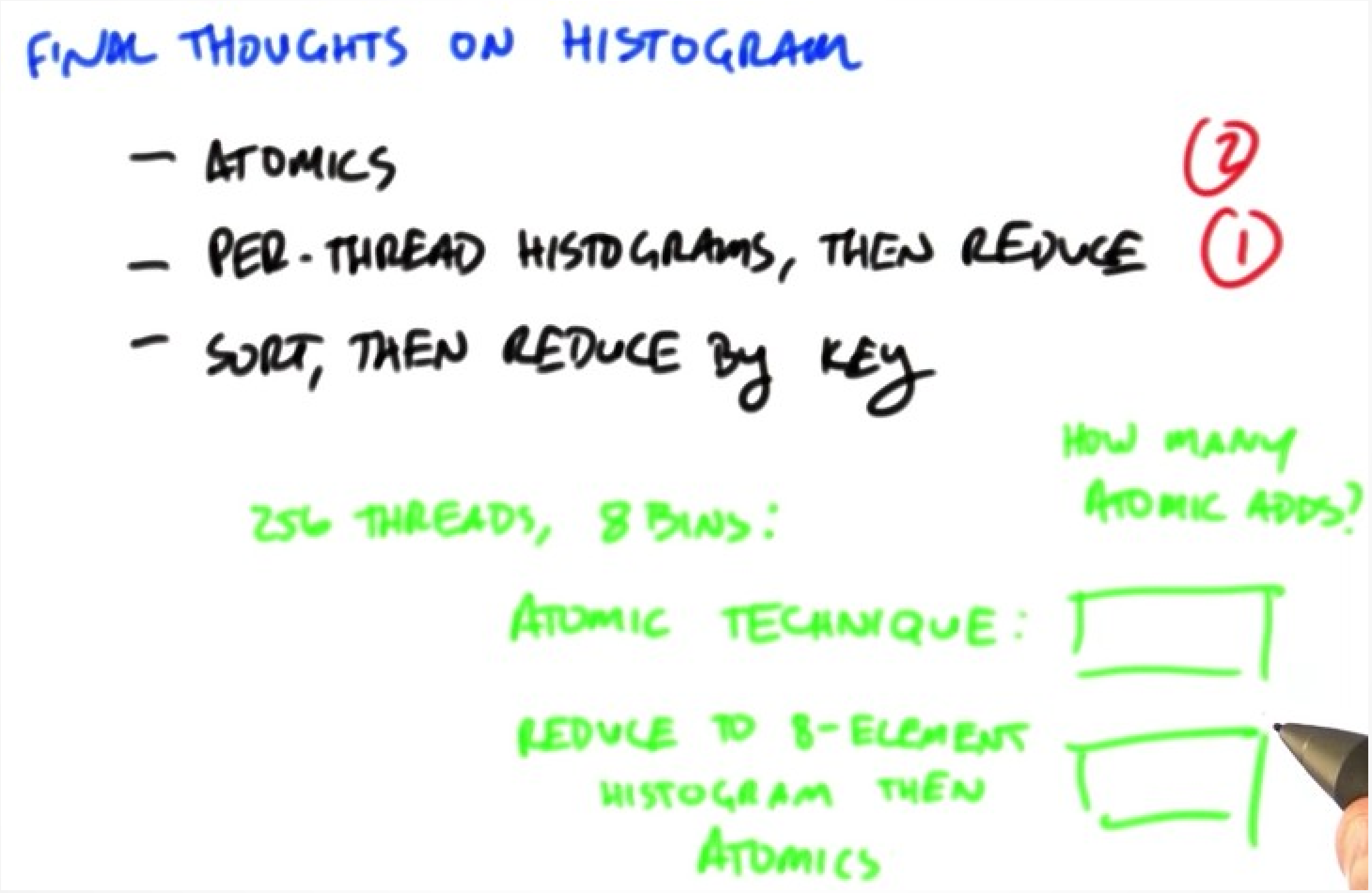


Histogram

Method 3 : sort then reduce by key



Histogram



FINAL THOUGHTS ON HISTOGRAM
– ATOMICS
2
– PER-THREAD HISTOGRAMS, THEN REDUCE
1
– SORT, THEN REDUCE BY KEY
HOW MANY ATOMIC ADDS?
256 THREADS, 8 BINS:
ATOMIC TECHNIQUE:
REDUCE TO 8-ELEMENT HISTOGRAM THEN ATOMICS

Topics

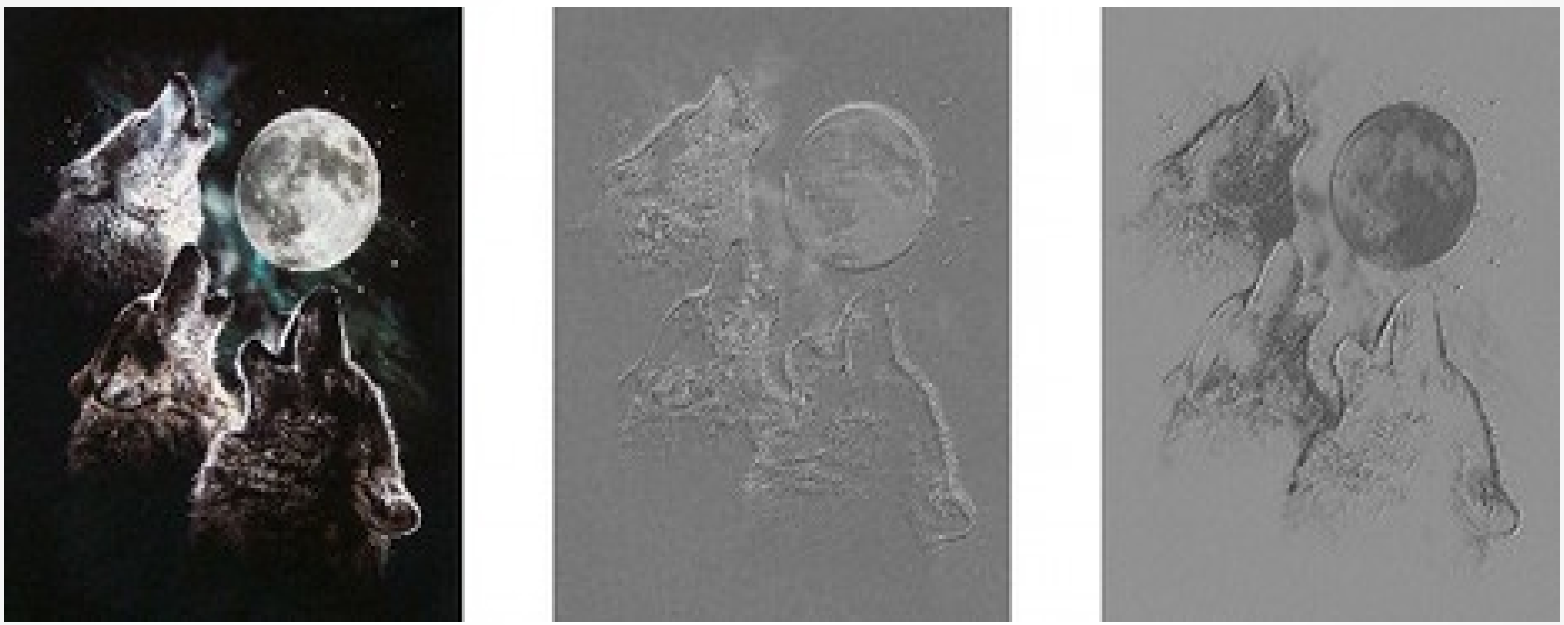
- Prefix Sum
- Histogram
- ***<u>Convolution</u>***

Convolution

Convolution Applications

- **A popular array operation that is used in various forms in signal processing, digital recording, image processing, video processing, and computer vision.**
- **Convolution is often performed as a filter that transforms signals and pixels into more desirable values.**
    - Some filters smooth out the signal values so that one can see the big-picture trend
    - Others like Gaussian filters can be used to sharpen boundaries and edges of objects in images..

Convolution



http://deeplearning.net/tutorial/lenet.html

Convolution

- **An array operation where each output data element is a weighted sum of a collection of neighboring input elements**

- **The weights used in the weighted sum calculation are defined by an input mask array, commonly referred to as the** *convolution kernel*

    - We will refer to these mask arrays as convolution masks to avoid confusion.

    - The same convolution mask is typically used for all elements of the array.

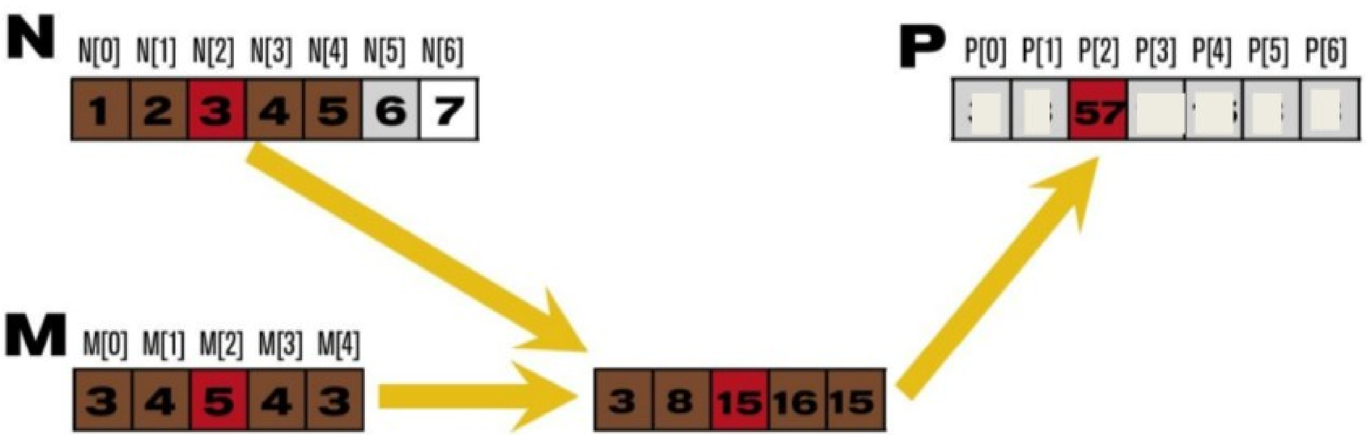
Convolution

1D Convolution Example

Commonly used for audio processing

- Mask size is usually an odd number of elements for symmetry (5 in this example)

Calculation of P[2]



Convolution

Definition [edit]

For a causal discrete-time FIR filter of order *N*, each value of the output sequence is a weighted sum of the most recent input values:

$$\begin{aligned} y[n] &= b_0 x[n] + b_1 x[n-1] + \cdots + b_N x[n-N] \\ &= \sum_{i=0}^{N} b_i \cdot x[n-i], \end{aligned}$$

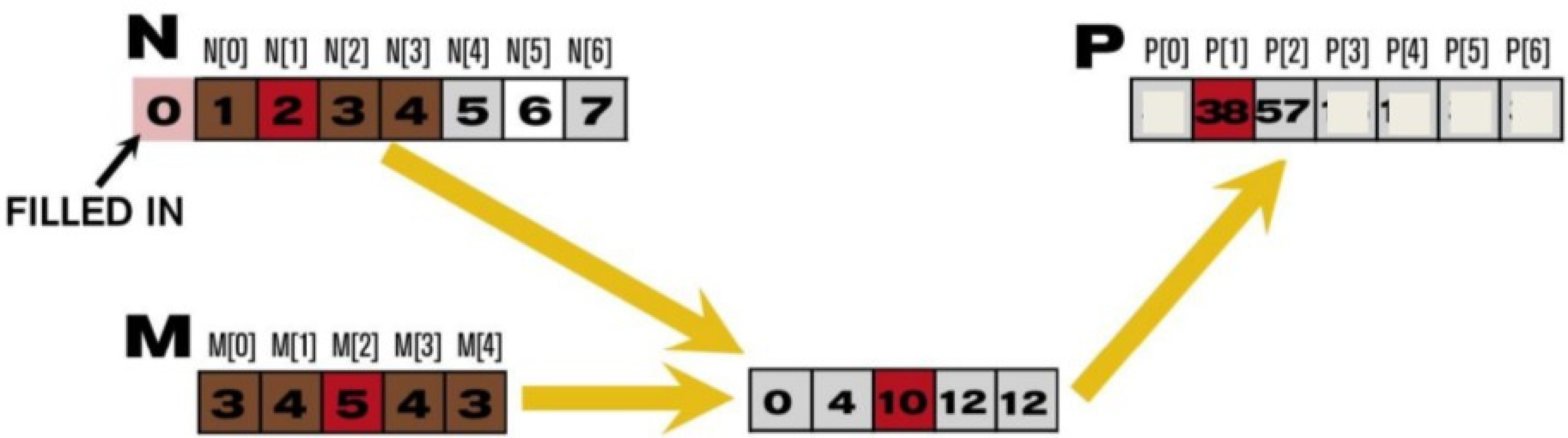
where:

- $x[n]$ is the input signal,
- $y[n]$ is the output signal,
- N is the filter order; an Nth-order filter has $(N+1)$ terms on the right-hand side
- b_i is the value of the impulse response at the *i*'th instant for $0 \leq i \leq N$ of an Nth-order FIR filter. If the filter is a direct form FIR filter then b_i is also a coefficient of the filter .

Convolution

Calculation of output elements near the boundaries (beginning and end) of the input array need to deal with “ghost” elements

- Different policies (0, replicates of boundary values, etc.

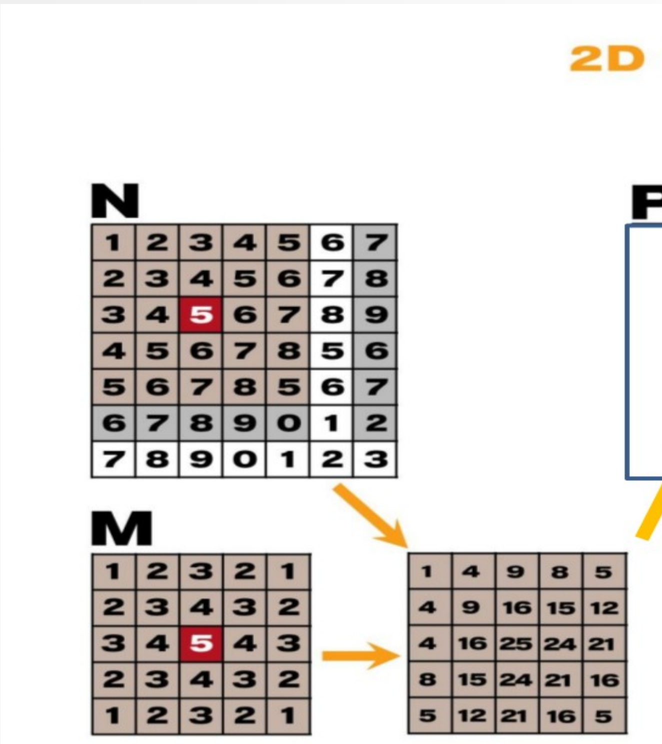


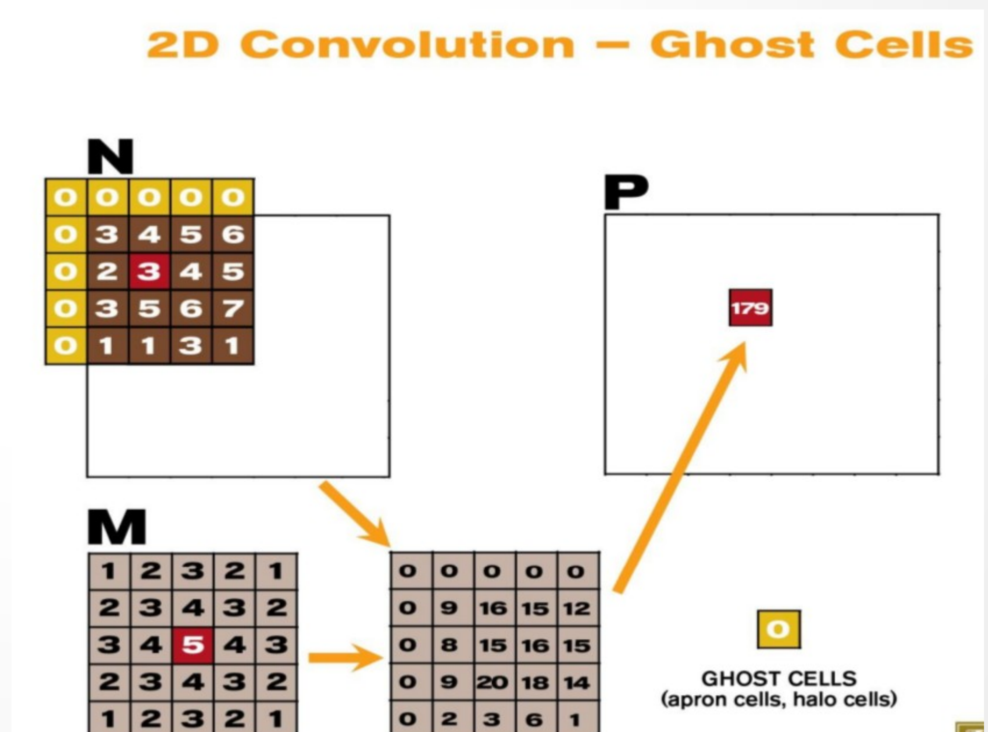
Convolution

- This kernel forces all elements outside the image to 0

```
__global__ void convolution_1D_basic_kernel(float *N, float *M, float *P,
 int Mask_Width, int Width) {

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

 float Pvalue = 0;
 int N_start_point = i - (Mask_Width/2);
 for (int j = 0; j < Mask_Width; j++) {
  if (N_start_point + j >= 0 && N_start_point + j < Width) {
   Pvalue += N[N_start_point + j]*M[j];
  }
 }
 P[i] = Pvalue;
}
```

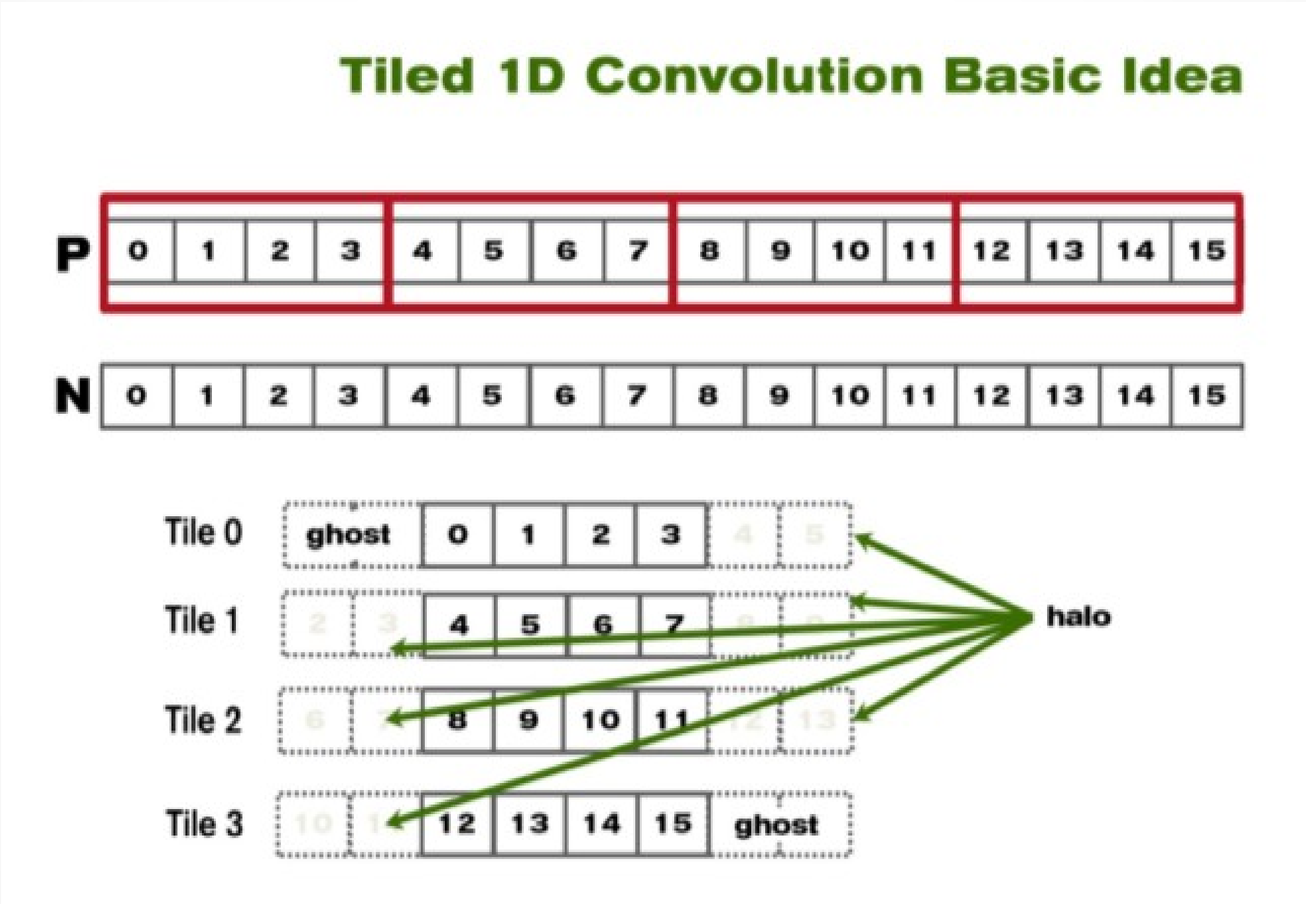
Convolution





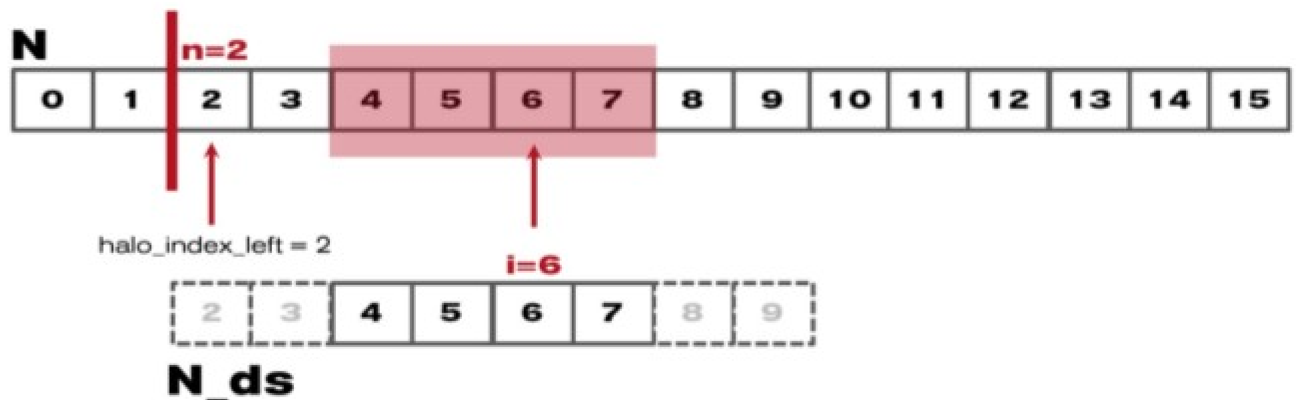
Convolution

Tiled Convolution



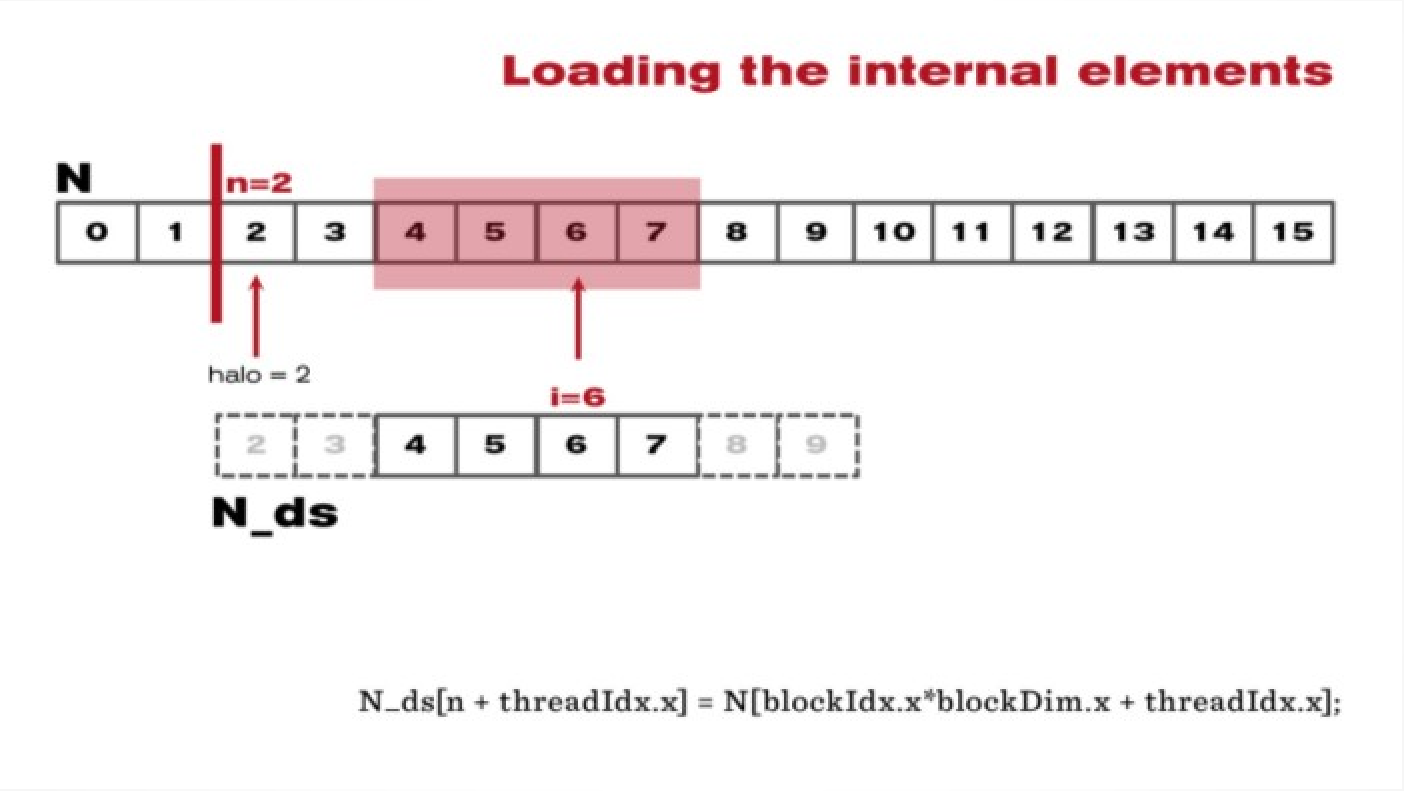
Convolution

Loading the left halo



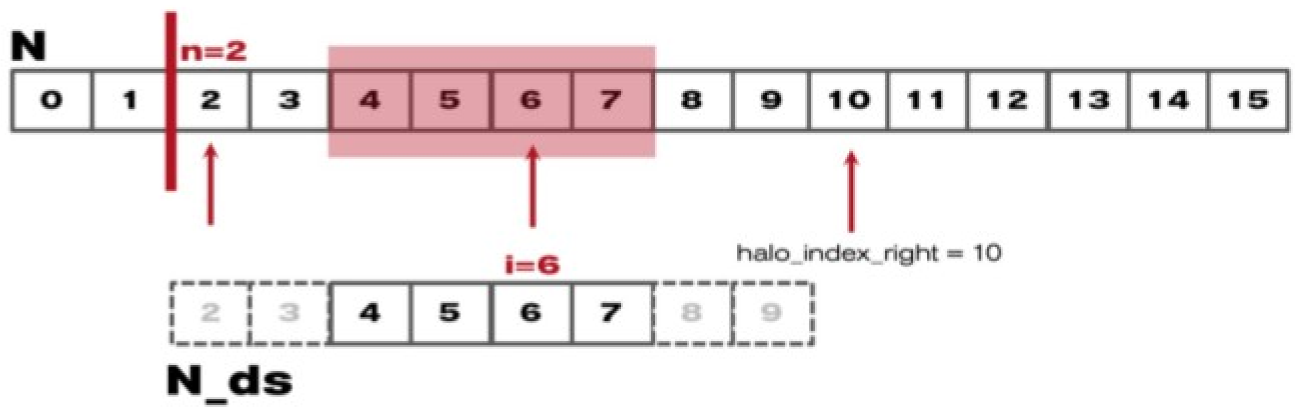

```
int n = Mask_Width/2;
int halo_index_left = (blockIdx.x - 1)*blockDim.x + threadIdx.x;
 if (threadIdx.x >= blockDim.x - n) {
  N_ds[threadIdx.x - (blockDim.x - n)] =
   (halo_index_left < 0) ? 0 : N[halo_index_left];
 }
```

Convolution



Convolution

Loading the right halo



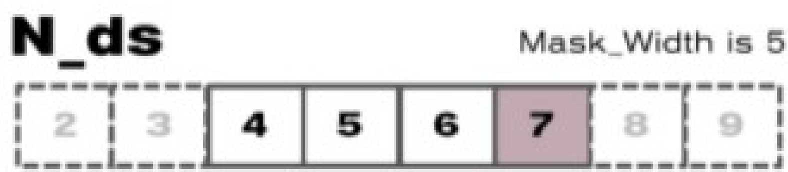

```
int halo_index_right = (blockIdx.x + 1)*blockDim.x + threadIdx.x;
if (threadIdx.x < n) {
   N_ds[n + blockDim.x + threadIdx.x] =
     (halo_index_right >= Width) ? 0 : N[halo_index_right];
 }
```

Convolution

```
__global__ void convolution_1D_basic_kernel(float *N, const float __restrict__ *M,
            float *P, int Mask_Width, int Width) {

 int i = blockIdx.x*blockDim.x + threadIdx.x;
 __shared__ float  N_ds[TILE_SIZE + MAX_MASK_WIDTH - 1];

 int n = Mask_Width/2;

 int halo_index_left = (blockIdx.x - 1)*blockDim.x + threadIdx.x;
 if (threadIdx.x >= blockDim.x - n) {
  N_ds[threadIdx.x - (blockDim.x - n)] =
   (halo_index_left < 0) ? 0 : N[halo_index_left];
 }

 N_ds[n + threadIdx.x] = N[blockIdx.x*blockDim.x + threadIdx.x];

 int halo_index_right = (blockIdx.x + 1)*blockDim.x + threadIdx.x;
 if (threadIdx.x < n) {
  N_ds[n + blockDim.x + threadIdx.x] =
   (halo_index_right >= Width) ? 0 : N[halo_index_right];
 }

 __syncthreads();

 float Pvalue = 0;
 for(int j = 0; j < Mask_Width; j++) {
  Pvalue += N_ds[threadIdx.x + j]*M[j];
 }
 P[i] = Pvalue;

}
```

Convolution

Shared Memory Data Reuse



Element 2 is used by thread 4 (1X)
Element 3 is used by threads 4, 5 (2X)
Element 4 is used by threads 4, 5, 6 (3X)
Element 5 is used by threads 4, 5, 6, 7 (4X)
Element 6 is used by threads 4, 5, 6, 7 (4X)
Element 7 is used by threads 5, 6, 7 (3X)
Element 8 is used by threads 6, 7 (2X)
Element 9 is used by thread 7 (1X)

References

- http://www.eecs.umich.edu/courses/eecs570/hw/parprefix.pdf
- http://http.developer.nvidia.com/GPUGems3/gpugems3_ch39.html
- Udacity : intro to parallel programming
- Coursera : heterogenous parallel programming