ACCELERATING TOP-*K* COMPUTATION ON GPU

Christina Zhang & Yong Wang

OUTLINE

Algorithms for Top-*K*

Optimizing WarpSelect

Optimizing Radix Select

PROBLEM DEFINITION

Top-*K*: find *k* smallest (or largest) elements in a list of *N* elements.

For example, when *N* = 8, *k* = 3:

3	4	6	1	5	8	2	7

TWO USEFUL ENHANCEMENTS IN PRACTICE

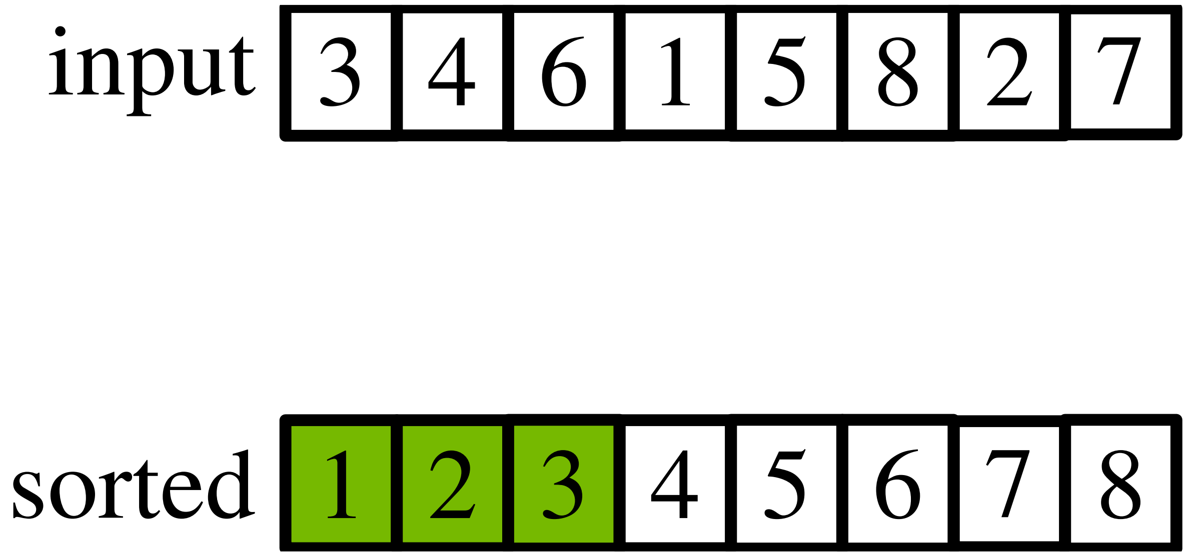
batch processing

3	4	6	1	5	8	2	7
4	2	6	1	8	3	7	5
8	6	4	5	2	7	3	1

return also the index

	3	4	6	1	5	8	2	7
index	0	1	2	3	4	5	6	7

ALGORITHM 1: SORTING



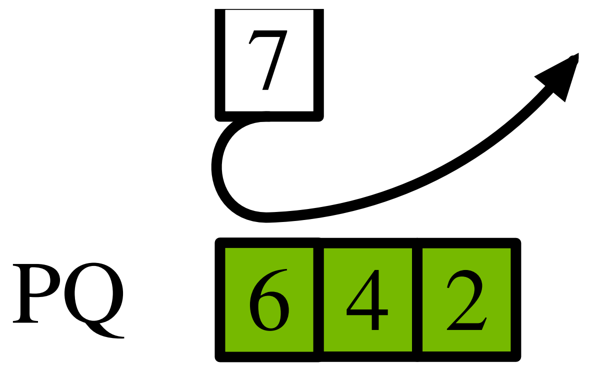
advantage: highly efficient sorting algorithms for GPU exist

disadvantage: do more work than necessary

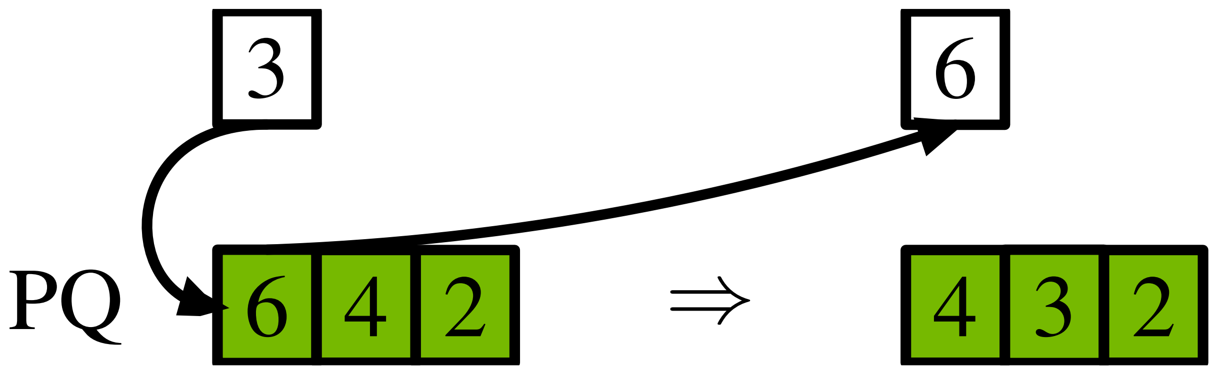
GPU implementation: cub::DeviceRadixSort from CUB library

ALGORITHM 2: PRIORITY QUEUE (PQ)

maintain a PQ of size *k*, and try to insert elements one after one:



a failed insertion



a successful insertion

advantages: read data only once, can handle streaming data; efficient when *k* is small

Disadvantage: depending on the implementation of PQ, the time complexity is usually at least proportional to log *k*, inefficient when *k* is large

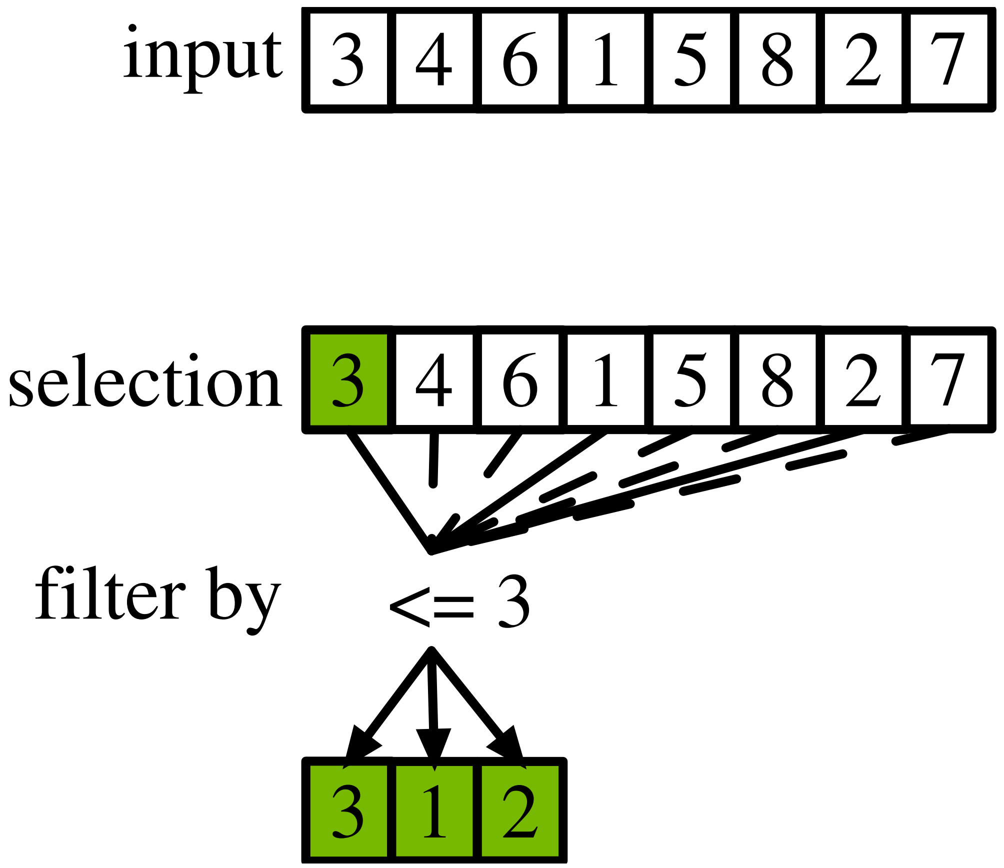
GPU implementation: WarpSelect from FAISS library (https://github.com/facebookresearch/faiss)

while PQ is commonly implemented with Heap for CPU, using parallel sorting can be more efficient on GPU.

ALGORITHM 3: SELECTION + FILTER

a closely related problem is *K*-selection Problem:

find the *k*-th smallest (or largest) element in a list of *N* elements



(note some selection algorithms can be modified to return top-*k* results by carefully maintaining intermediate data thus don't need the filter step)

ALGORITHM 3: SELECTION + FILTER

many sorting algorithms have corresponding selection algorithms:

sorting	selection
quicksort	quick select
samplesort	sample select
radix sort	radix select

advantages: selection algorithm usually has time complexity O(N), so efficient when k and N are large

disadvantages: may need to read input multiple times

GPU implementation: radix sort can be efficiently implemented on GPU, will explain radix select in detail

OPTIMIZING WARPSELECT

WARPSELECT

Billion-scale similarity search with GPUs, arXiv:1702.08734

WarpSelect: the basic idea is PQ, but implementation is highly optimized for GPU

using a single warp (32 threads on GPU), all data is kept in register

- PQ is maintained by a warp
- insertion into PQ is done by a warp using odd-size merging network
- not insert every time encountering a greater element, but accumulate them in a buffer for each thread
- accumulated elements is sorted using bitonic sorting network

BlockSelect

- use 2-4 warps to get intermediate results saved in shared memory
- merge these results until get *k* elements

ACCELERATE WARPSELECT BY INCREASING ITS OCCUPANCY

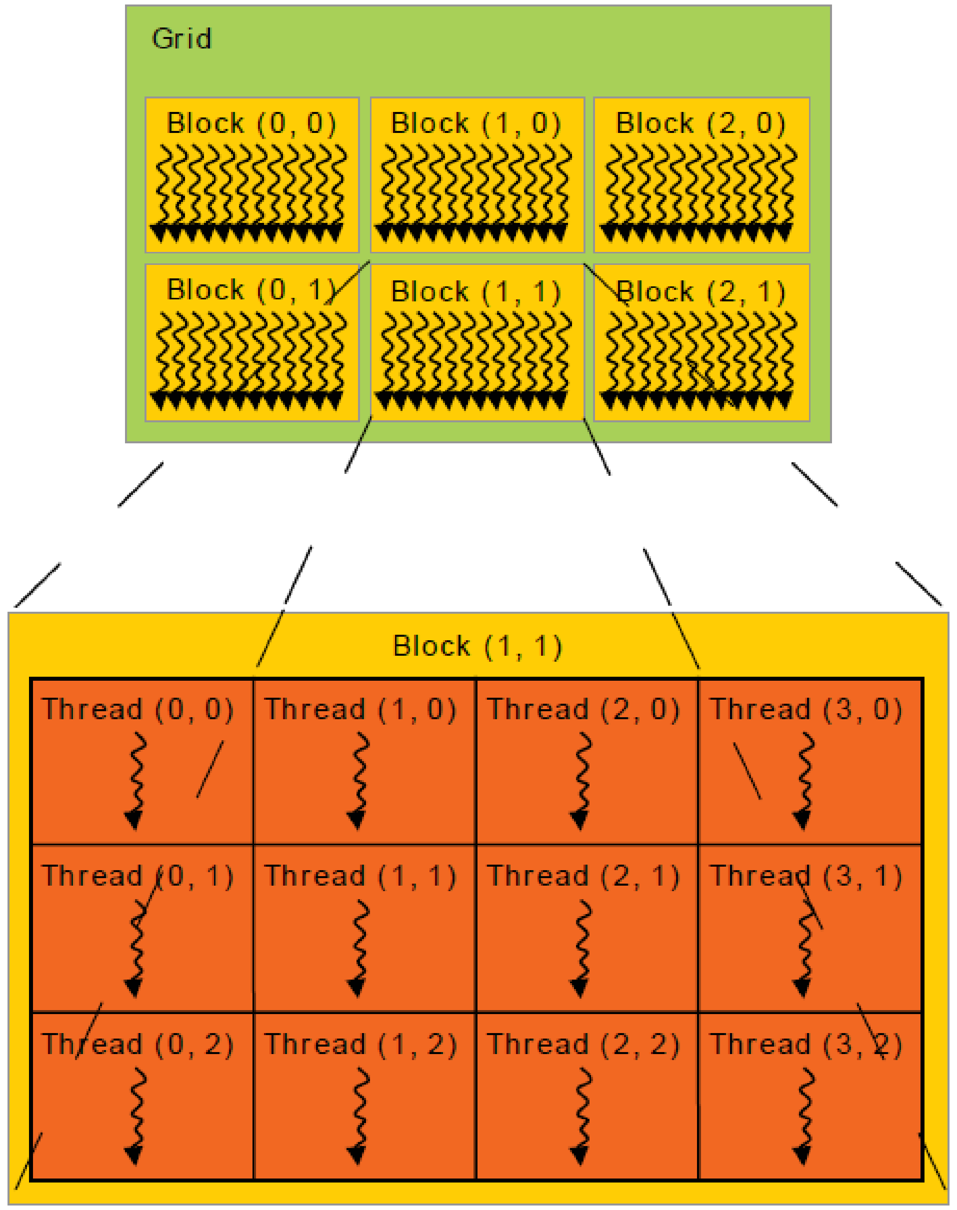
WarpSelect is highly efficient for a warp

it uses a single warp

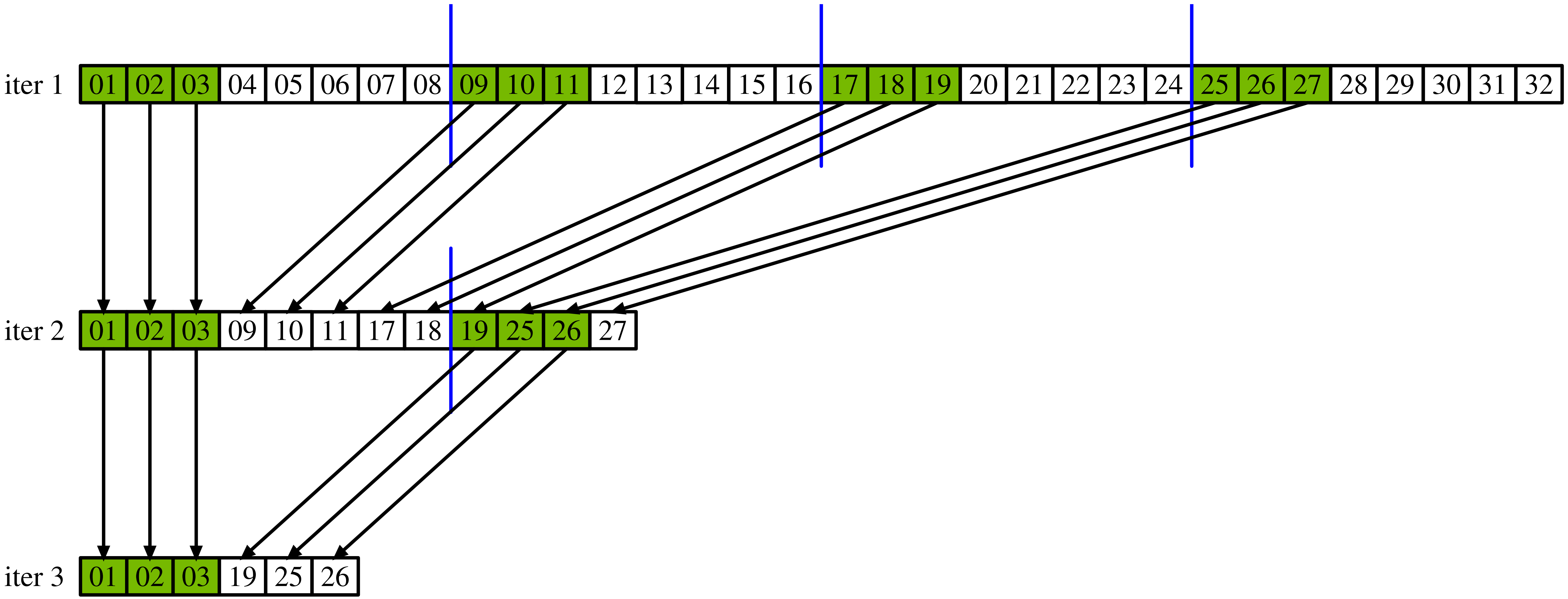
want to combine it to get a CUDA kernel that

- uses a block of threads or
- uses multiple blocks of threads

the occupancy will be increased thus runs faster



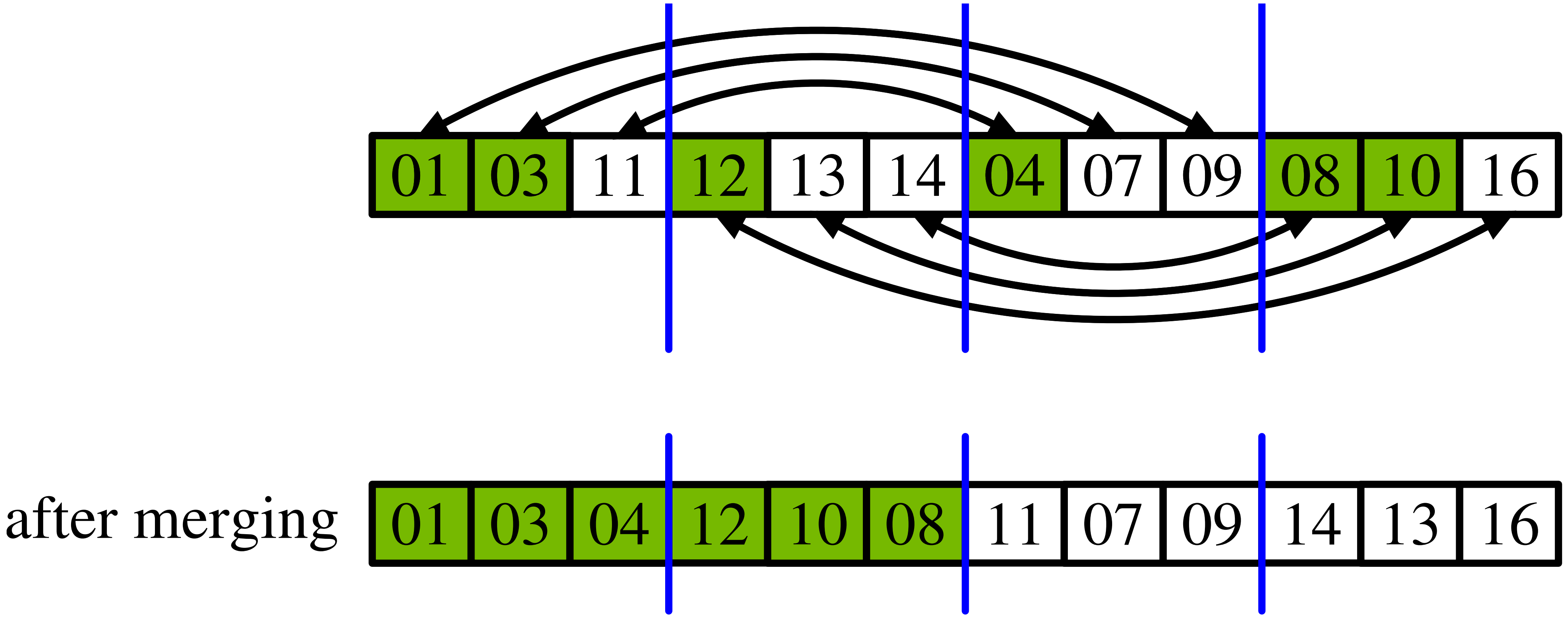
from “CUDA C++ Programming Guide”



iterates until *k* elements is found:

- divide input into multiple parts
- every parts is processed by a warp, the top *k* results of each part are gathered and used as input for next iteration

A TRICK FROM BITONIC MERGING



TRADE-OFF FOR AN INSTANCE IN A BATCH

	#thread	intermediate memory	sync cost
warp	32	register (lowest latency)	low
block	[32, 1024]	shared memory (low latency)	medium (__syncthreads())
multi-block	can be large	global memory (high latency)	high (e.g. extra kernel launch)

- set limit for the number of elements a warp/block can handle
- use more threads only when N of this part is larger than the limit

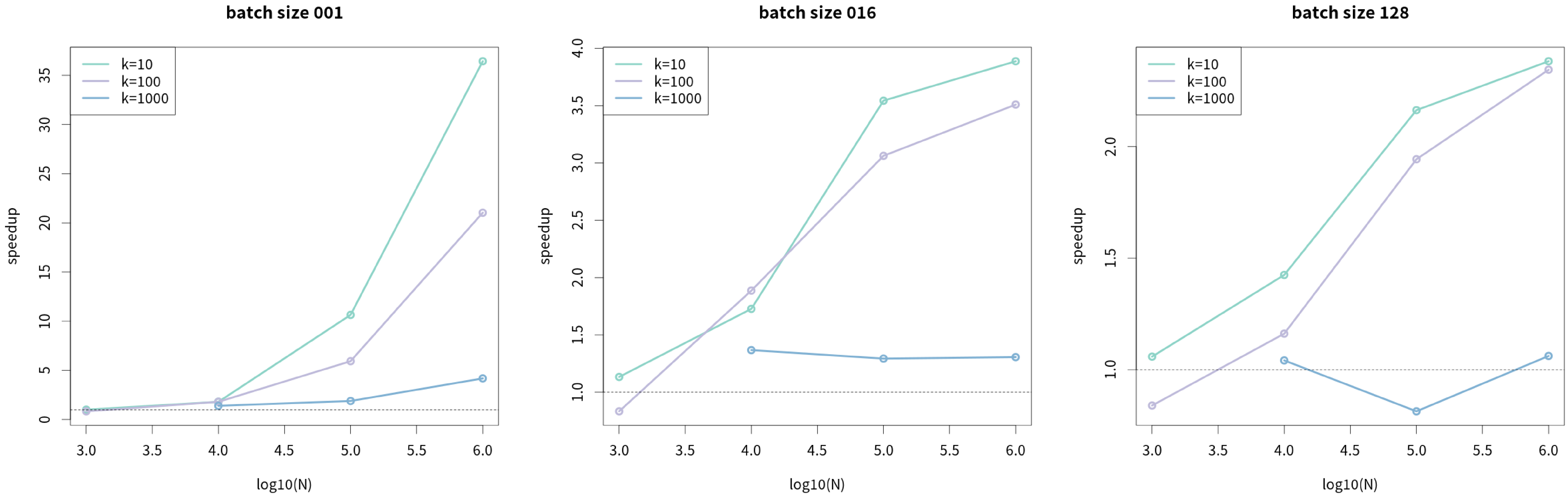
TRADE-OFF WHEN CONSIDERING BATCH SIZE

when batch size is large:

- the number of thread for an instance can be low to saturate GPU
- use fewer threads for an instance

BENCHMARK

speedup over BlockSelect on Telsa V100 GPU



OPTIMIZING RADIX SELECT

OUTLINE

Accelerate Radix Select with CUDA

- Radix select introduction
- Parallel implementation
- Implementation for different data size

OUTLINE

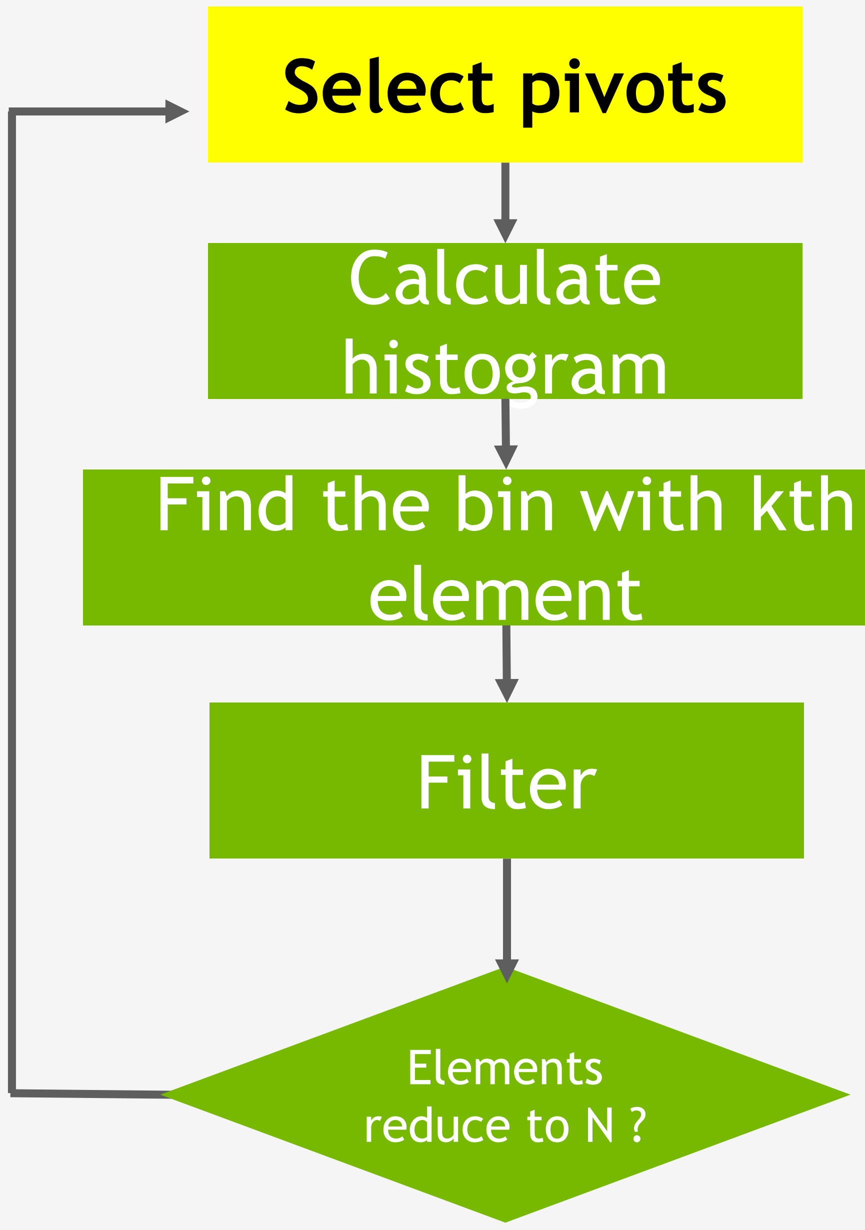
Accelerate Radix Select with CUDA

- **Radix select introduction**
- Parallel implementation
- Implementation for different data size

BUCKET/RANDOM/RADIX SELECT

Id	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Arr	11	15	13	12	18	16	14	10	9	8	20	17	7	23	22	19

k	7



For example :

5<9<10

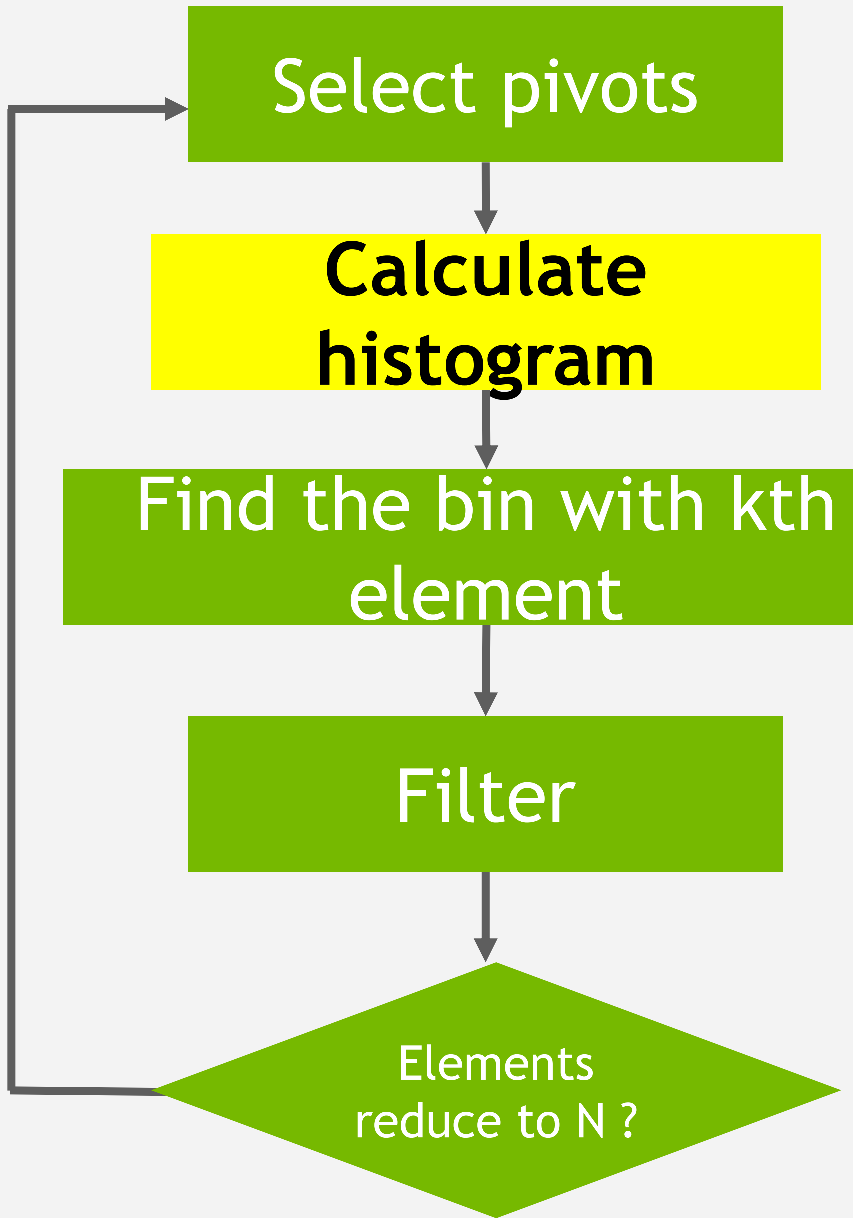
0–5	5–10	10–15	15–20	20–25
0	0	0	0	0

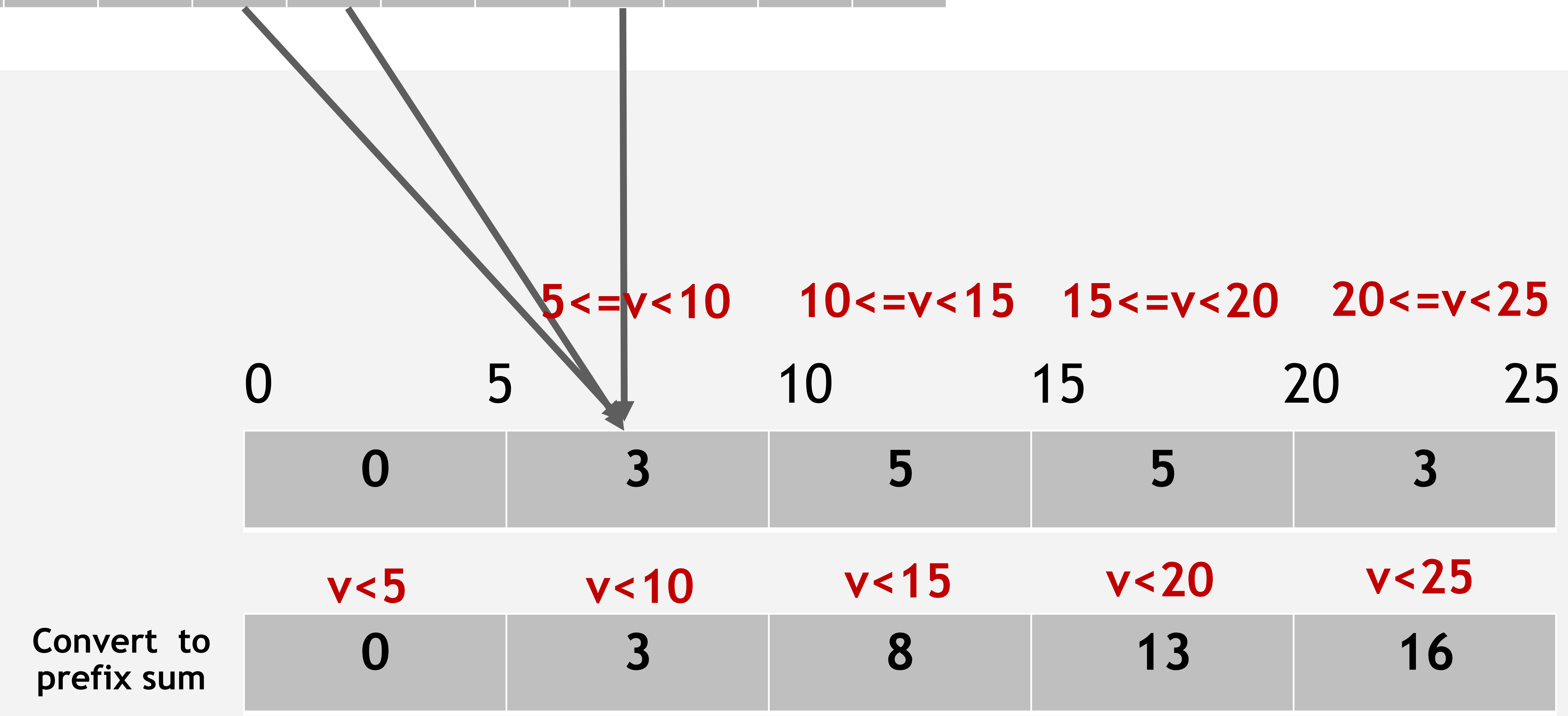
Alabi T , Blanchard J D , Gordon B , et al. Fast k -selection algorithms for graphics processing units[J]. Journal of Experimental Algorithmics, 2012, 17(1):4.2.

BUCKET/RANDOM /RADIX SELECT

Id	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Arr	11	15	13	12	18	16	14	10	9	8	20	17	7	23	22	19

k	7

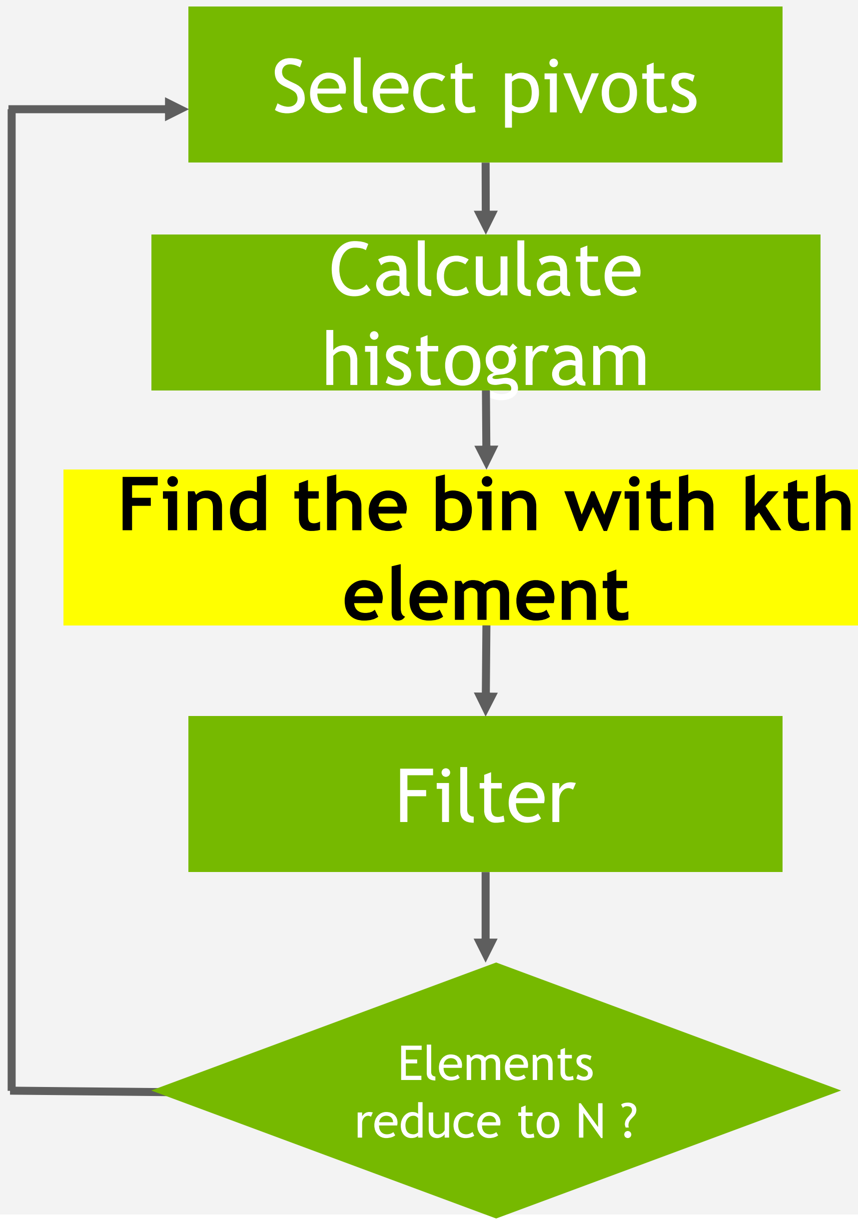




BUCKET/RANDOM /RADIX SELECT

Id	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Arr	11	15	13	12	18	16	14	10	9	8	20	17	7	23	22	19

k	7



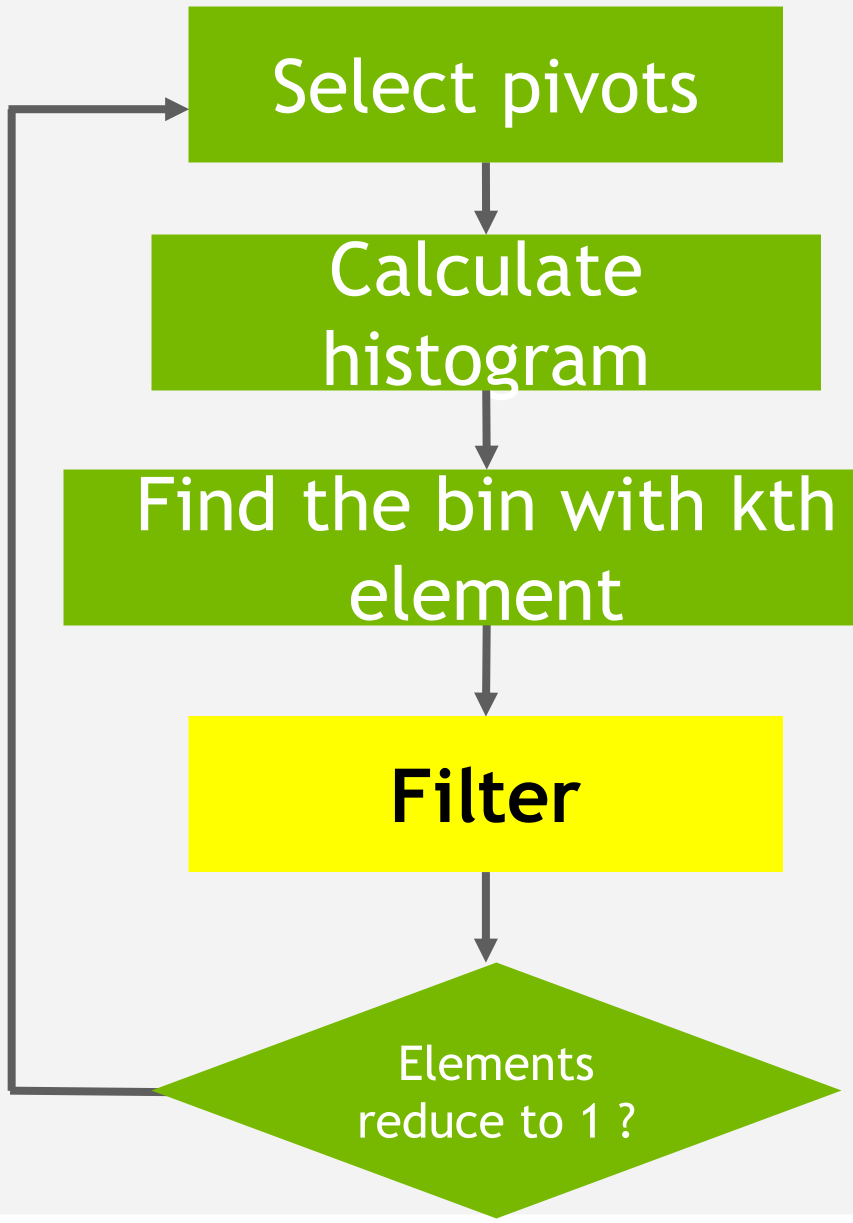
	0	5	10	15	20
	0	3	5	5	3
Convert to prefix sum	0	3	8	13	16
	0 < 7	3 < 7	7 < 8	7 < 13	7 < 16

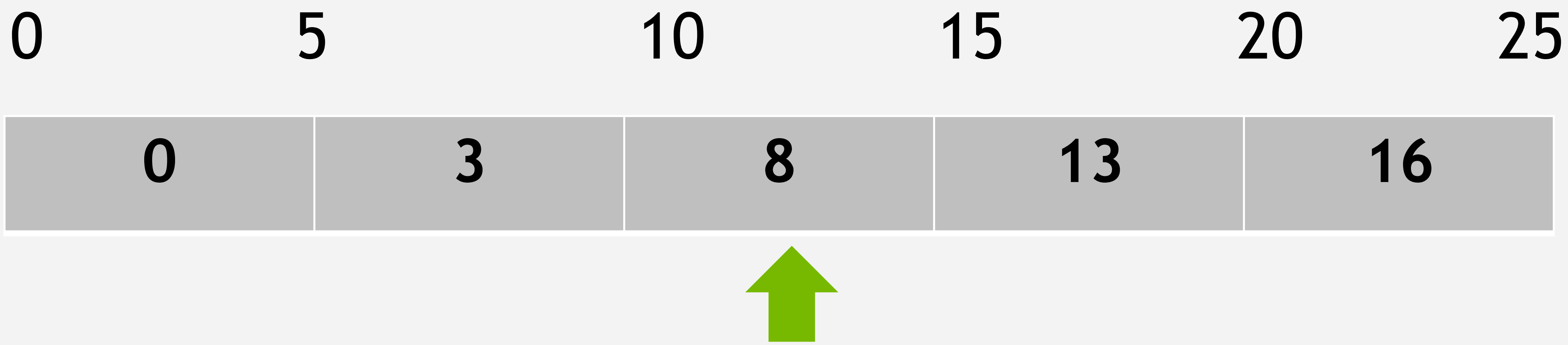
25

BUCKET/RANDOM /RADIX SELECT

Id	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Arr	11	15	13	12	18	16	14	10	9	8	20	17	7	23	22	19

k	7





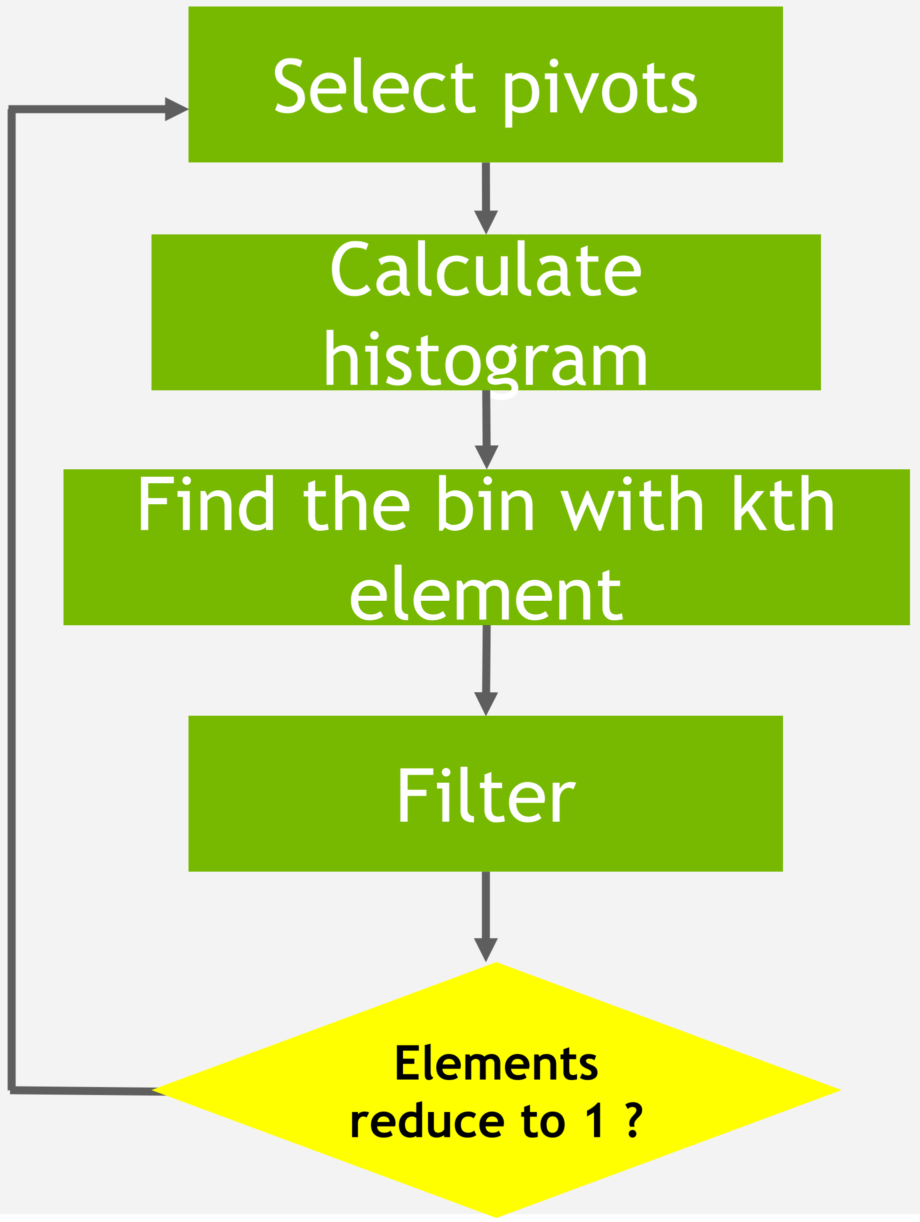
Id	0	1	2	3	4
Arr	11	13	12	14	10

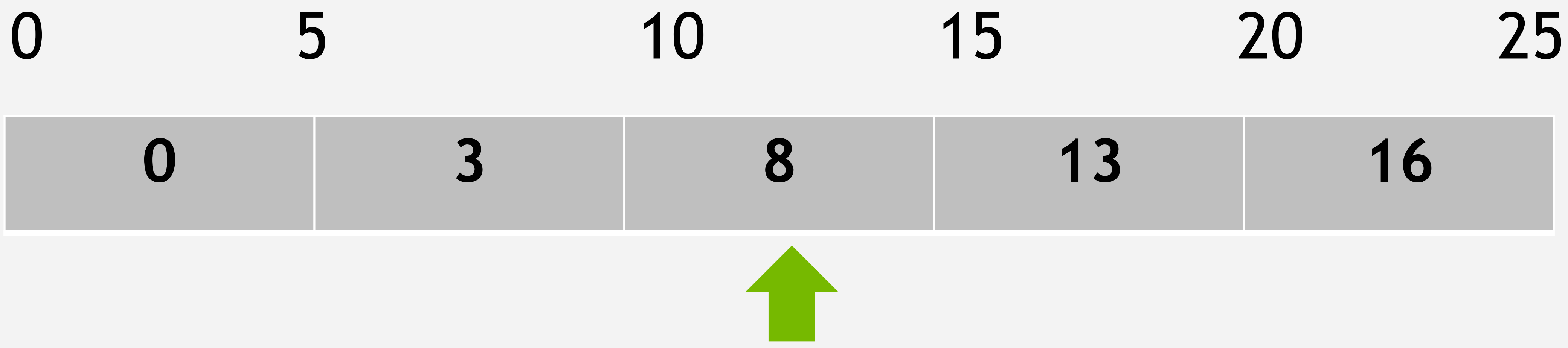
k	4

BUCKET/RANDOM /RADIX SELECT

Id	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Arr	11	15	13	12	18	16	14	10	9	8	20	17	7	23	22	19

k	7





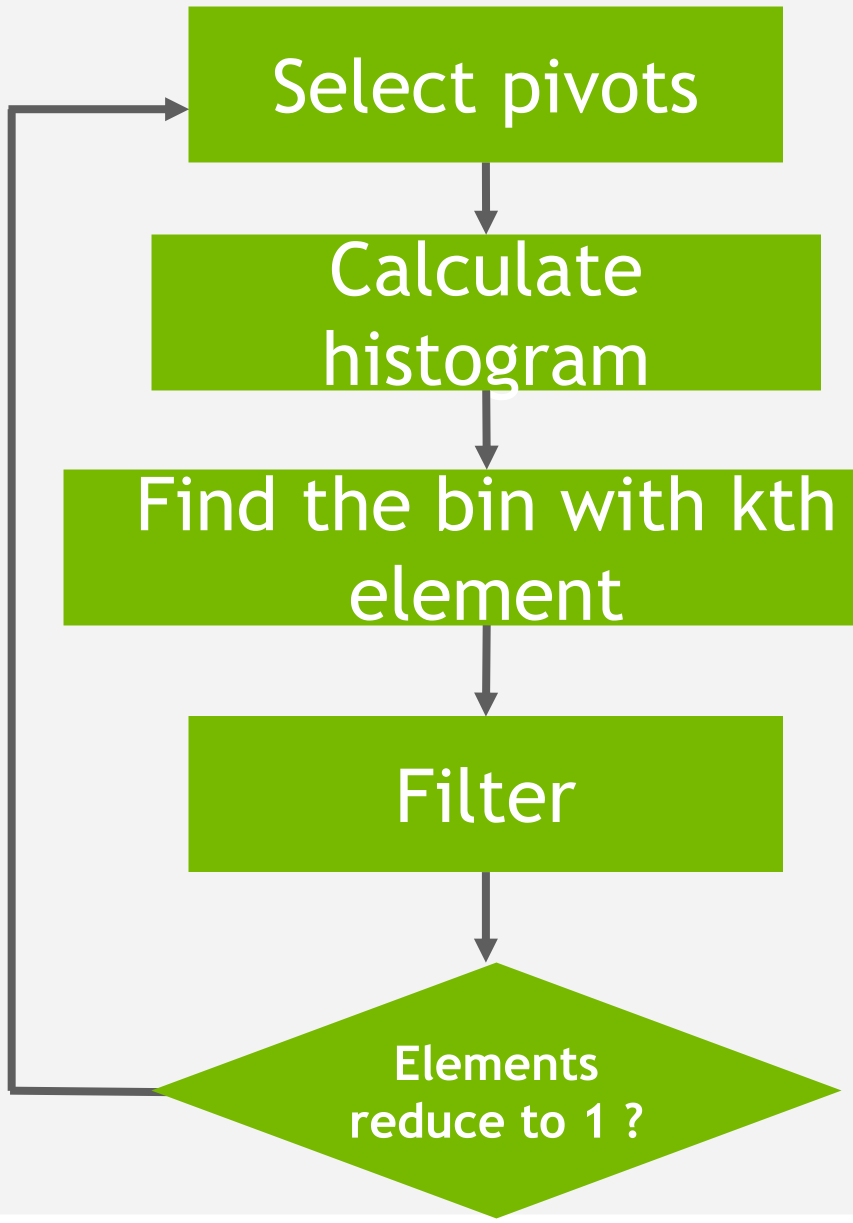
Id	0	1	2	3	4
Arr	11	13	12	14	10

k’	4

BUCKET/RANDOM /RADIX SELECT

Id	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Arr	11	15	13	12	18	16	14	10	9	8	20	17	7	23	22	19

k	7



Id	0	1	2	3	4
Arr	11	13	12	14	10

k’	4

	10–11	11–12	12–13	13–14	14–15
	1	1	1	1	1
Convert to prefix sum	1	2	3	4	5

Kth value	13

RADIX SELECT

For example:
32 bit unsigned int

0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
…				
15	1	1	1	1

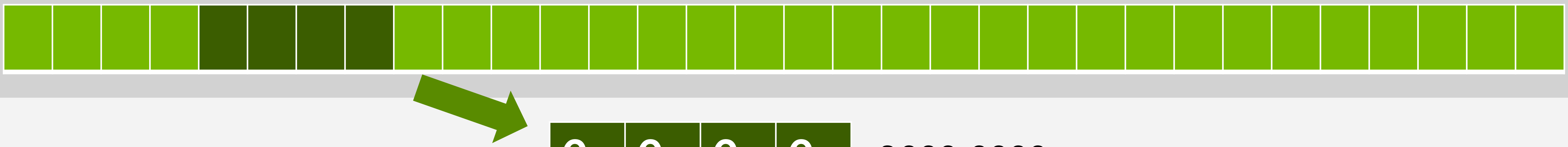
RADIX SELECT

For example:
32 bit unsigned int

0	0 0 0 0	0	0fff ffff	0 ~ 268435455
1	0 0 0 1	1000 0000	1fff ffff	268435456 ~ 536870911
2	0 0 1 0	2000 0000	2fff ffff	536870912 ~ 805306367
3	0 0 1 1	3000 0000	3fff ffff	805306368 ~ 1073741823
4	0 1 0 0	4000 0000	4fff ffff	1073741824 ~ 1342177279
5	0 1 0 1	5000 0000	5fff ffff	1342177280 ~ 1610612735
…		…	…	…
15	1 1 1 1	f000 0000	ffff ffff	4026531840 ~ 4294967295

RADIX SELECT

For example:
32 bit unsigned int



0	0	0	0	3000 0000
0	0	0	1	31ff ffff
0	0	1	0	32ff ffff
0	0	1	1	33ff ffff
0	1	0	0	34ff ffff
0	1	0	1	35ff ffff
				…
1	1	1	1	3fff ffff

RADIX SELECT

For example:
32 bit unsigned int

0	0 0 0 0	0	0 0 0 0	3000 0000	0 0 0 0	3400 0000
1	0 0 0 1	1fff ffff	0 0 0 1	31ff ffff	0 0 0 1	341f ffff
2	0 0 1 0	2fff ffff	0 0 1 0	32ff ffff	0 0 1 0	342f ffff
3	0 0 1 1	3fff ffff	0 0 1 1	33ff ffff	0 0 1 1	343f ffff
4	0 1 0 0	4fff ffff	0 1 0 0	34ff ffff	0 1 0 0	344f ffff
5	0 1 0 1	5fff ffff	0 1 0 1	35ff ffff	0 1 0 1	345f ffff
…		…		…		…
15	1 1 1 1	ffff ffff	1 1 1 1	3fff ffff	1 1 1 1	34ff ffff

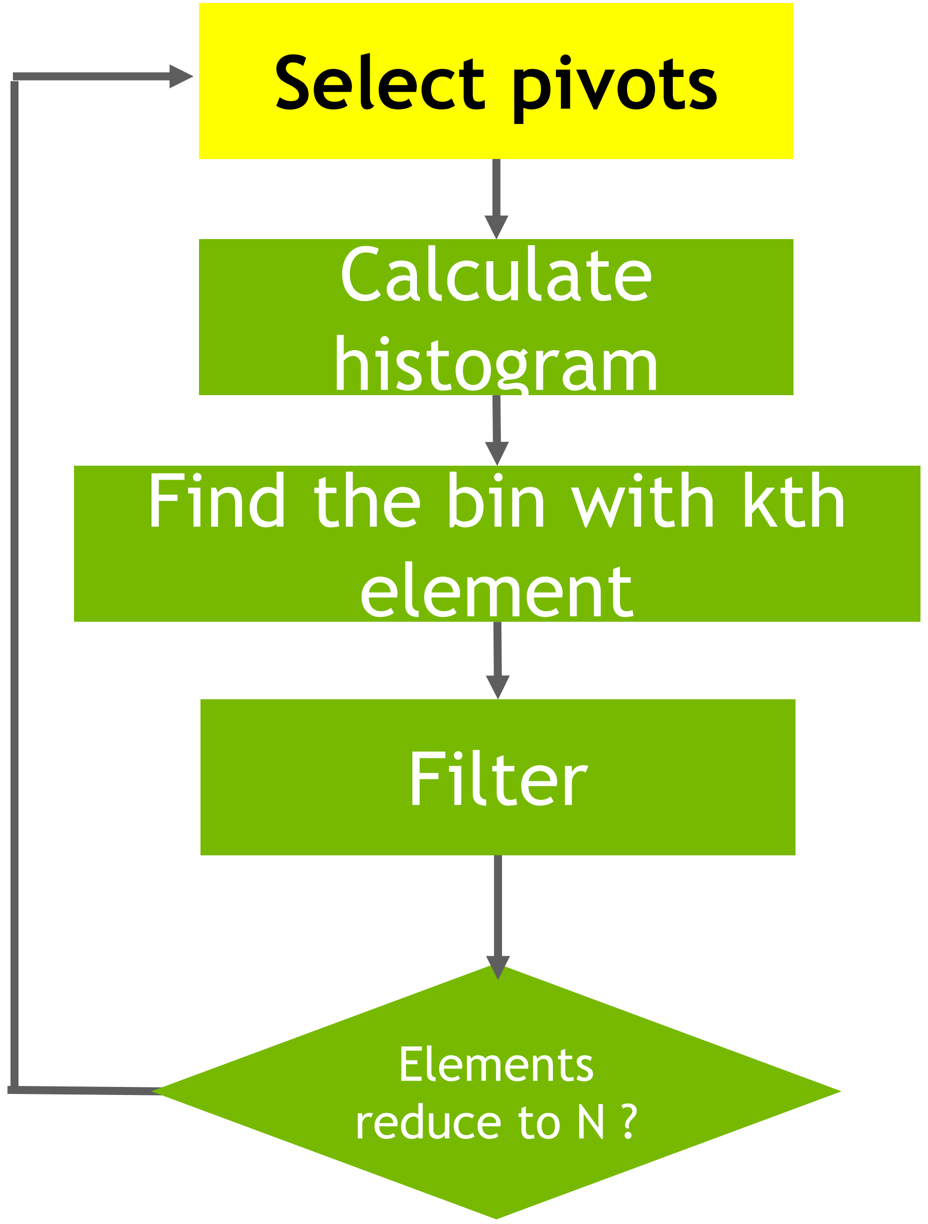
OUTLINE

Accelerate Radix Select with CUDA

- Radix select introduction
- **Parallel implementation**
- Implementation for different data size

PARALLEL IMPLEMENTATION

Select pivots



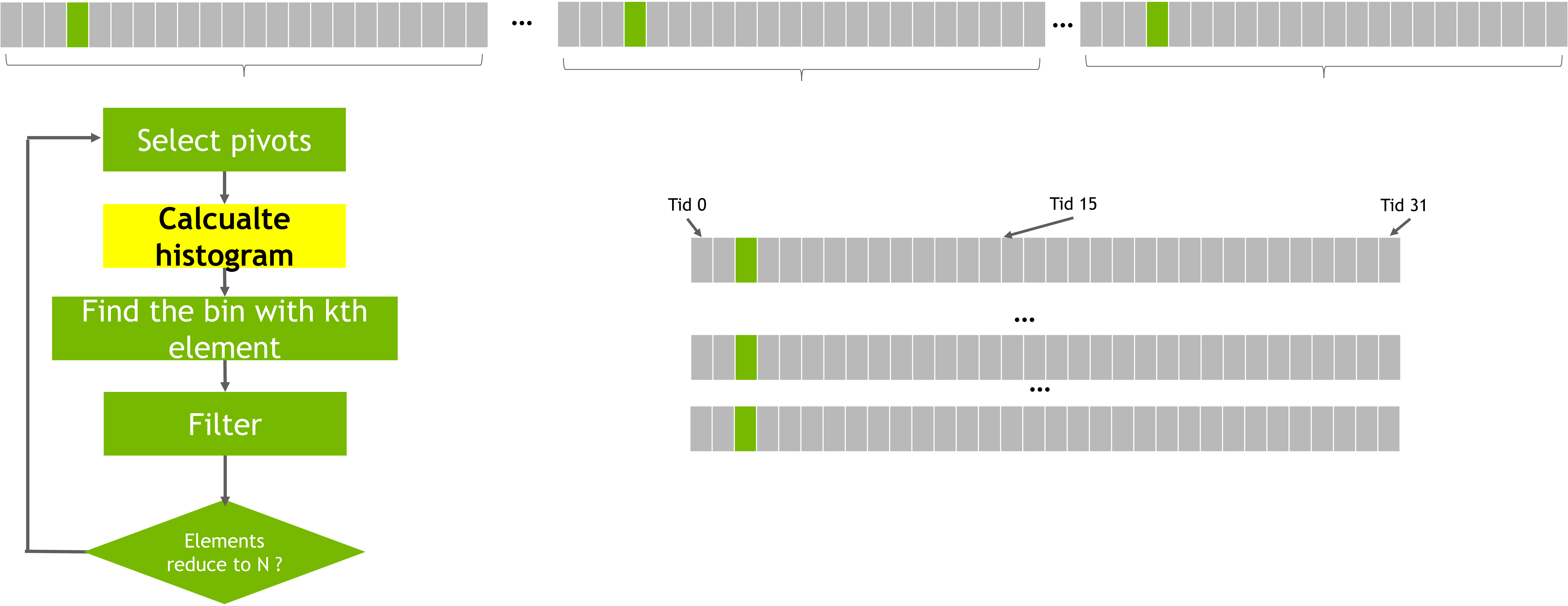
In each iteration, we use **8 bits/pass** to perform radix select.

```
__forceinline__  __device__ unsigned int float_flip(unsigned int f)
{
    unsigned int  mask = -int(f >> 31) | 0x80000000;
    return f ^ mask;
}
__forceinline__  __device__ unsigned int ifloat_flip(unsigned int  f)
{
    unsigned int  mask = ((f >> 31) - 1) | 0x80000000;
    return f ^ mask;
}
```

```
mask = float_flip((unsigned int&)in_buf[index]);
mask = mask>>(32- BIT_HIST*(digit+1));
idx_digit = mask&(0X00ff);
```

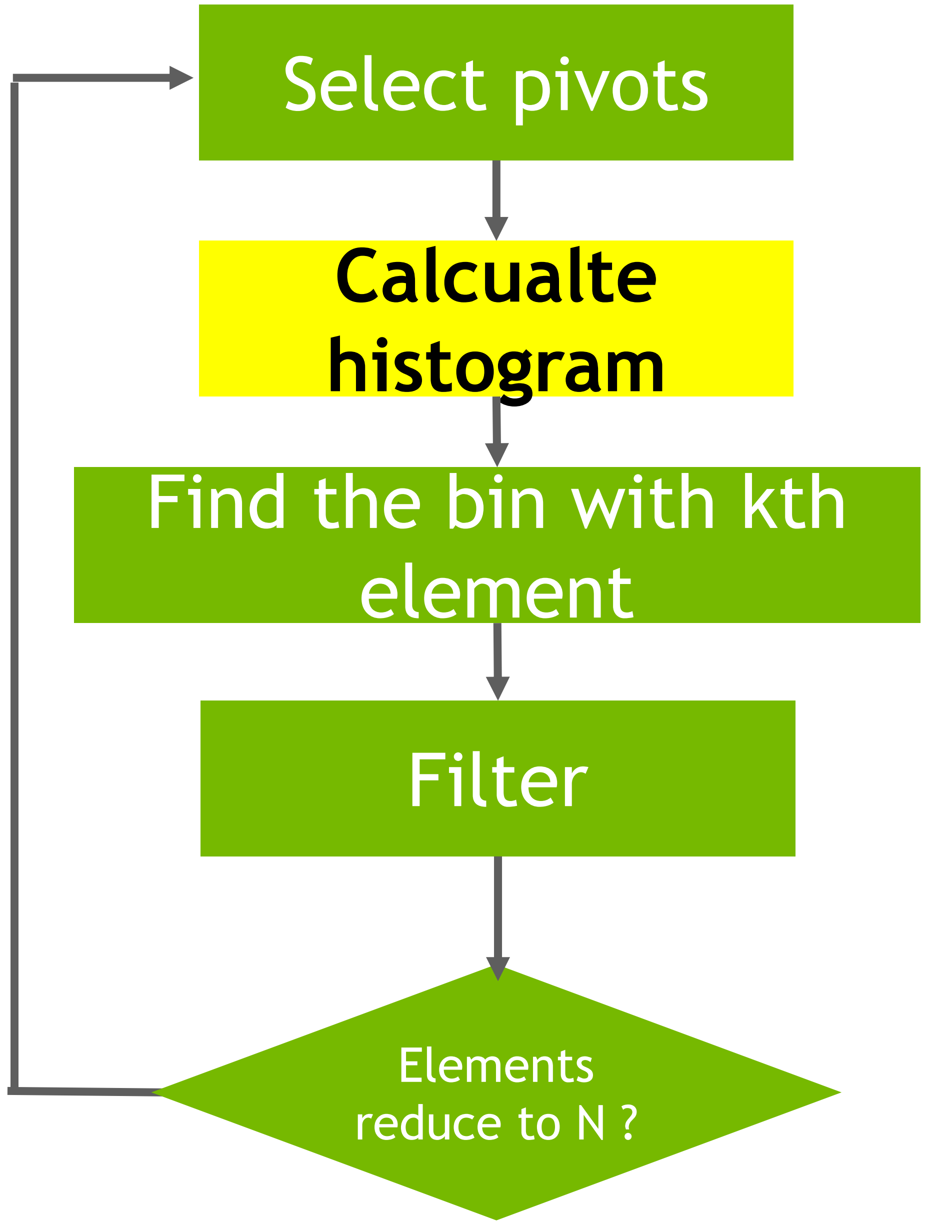
PARALLEL IMPLEMENTATION

Calculate histogram



PARALLEL IMPLEMENTATION

Calculate histogram



Digit value

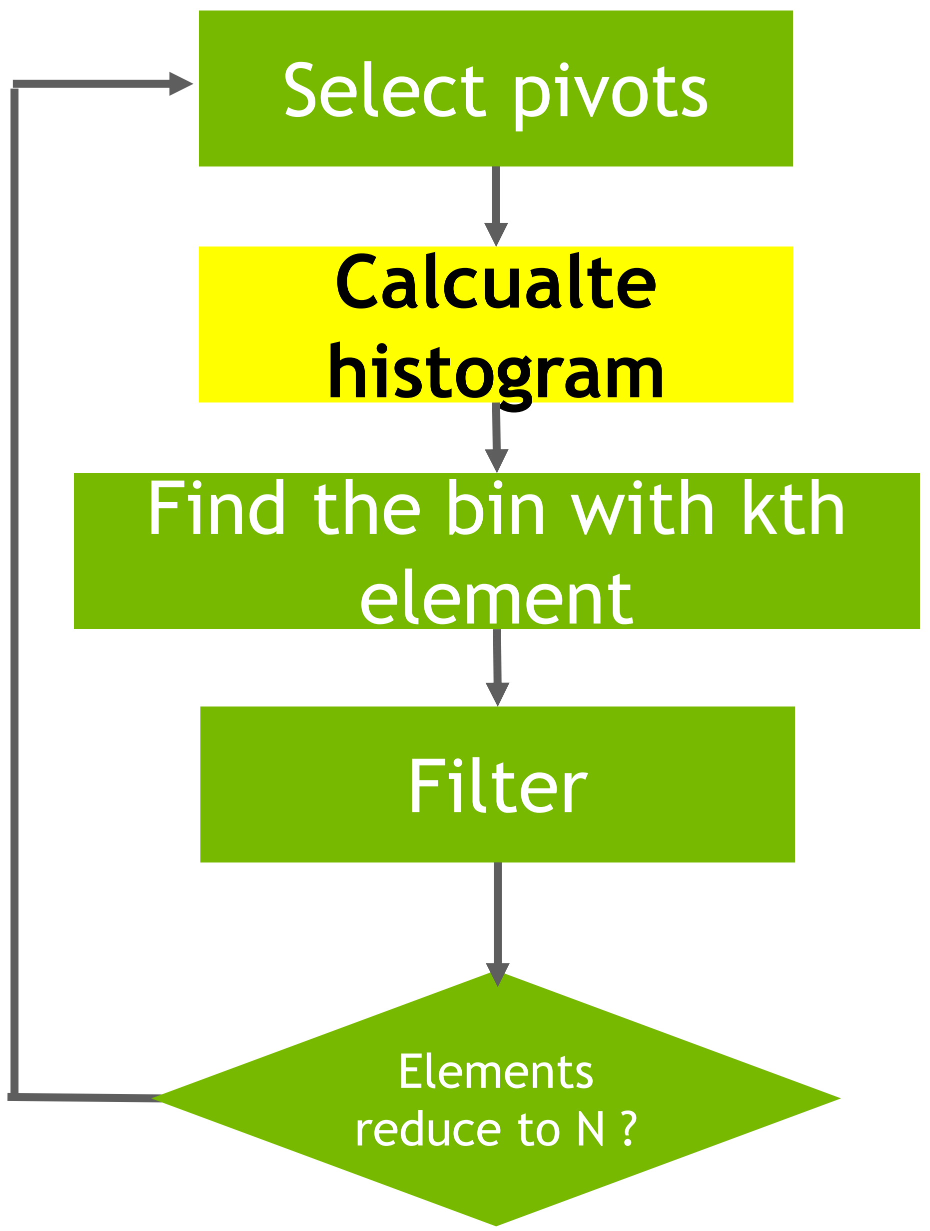
```
mask = float_flip((unsigned int&)in_buf[index]);
mask = mask>>(32-BIT_HIST*(digit+1));
idx_digit = mask&(0X00ff);
```

1	2	3	4	5	7	9	2	3	4	5	6	4	2	2	3	4	4	4	5	5	6	6	7	8	9	0	1	2	2	3	4

S21572, Andy Adinets, A FASTER RADIX SORT IMPLEMENTATION
https://developer.nvidia.com/blog/cuda-pro-tip-optimized-filtering-warp-aggregated-atomics/

PARALLEL IMPLEMENTATION

Calculate histogram



Digit value

```
mask = float_flip((unsigned int&)in_buf[index]);
mask = mask>>(32- BIT_HIST*(digit+1));
idx_digit = mask&(0X00ff);
```

1	2	3	4	5	7	9	2	3	4	5	6	4	2	2	3	4	4	4	5	5	6	6	7	8	9	0	1	2	2	3	4

Add to the block of histogram

```
atomicAdd(&(b_hist[idx_digit]), 1);
```

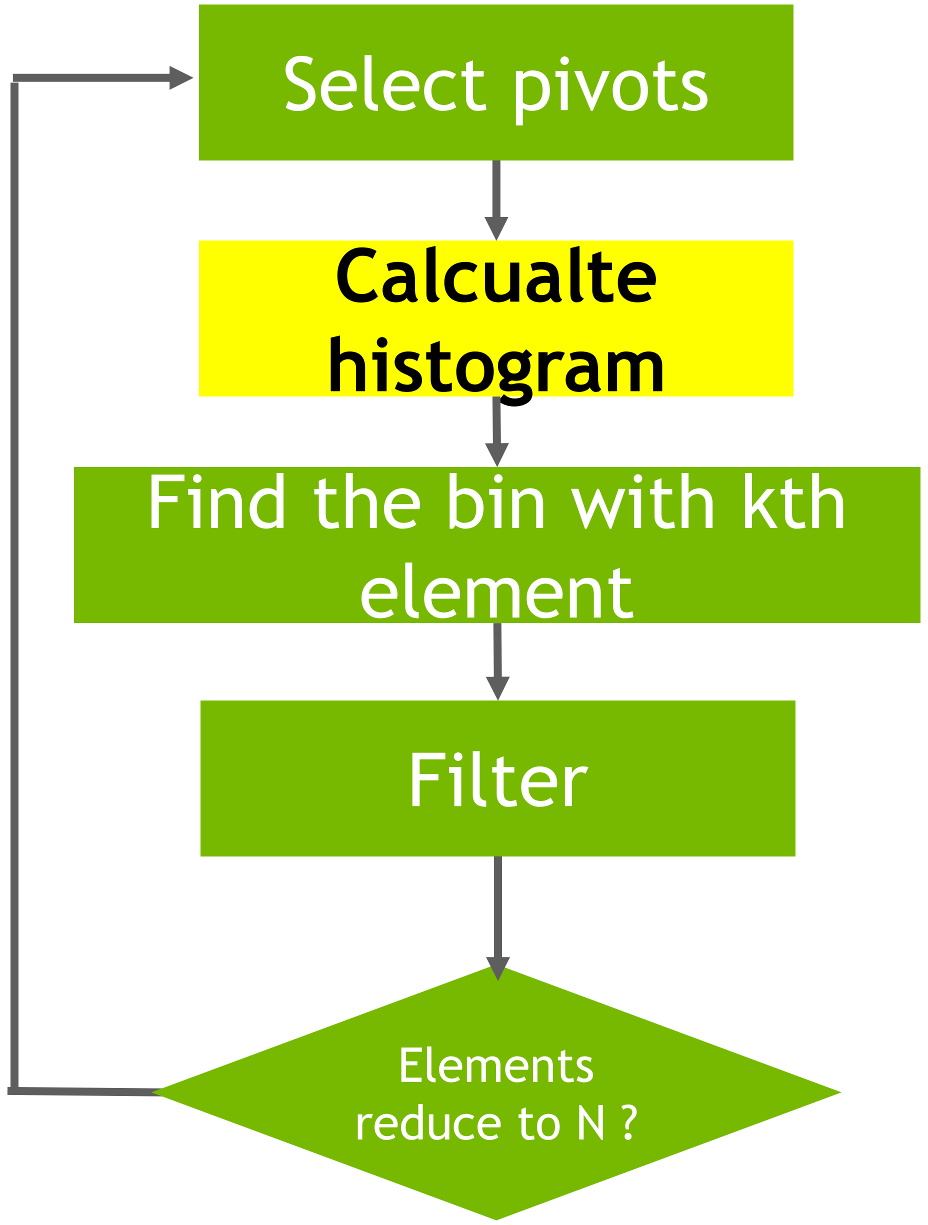
Add to the global of histogram

```
atomicAdd(&(g_hist[idx_digit]), b_hist[idx_digit]);
```

S21572, Andy Adinets, A FASTER RADIX SORT IMPLEMENTATION
https://developer.nvidia.com/blog/cuda-pro-tip-optimized-filtering-warp-aggregated-atomics/

PARALLEL IMPLEMENTATION

Calculate histogram



Add to the block of histogram

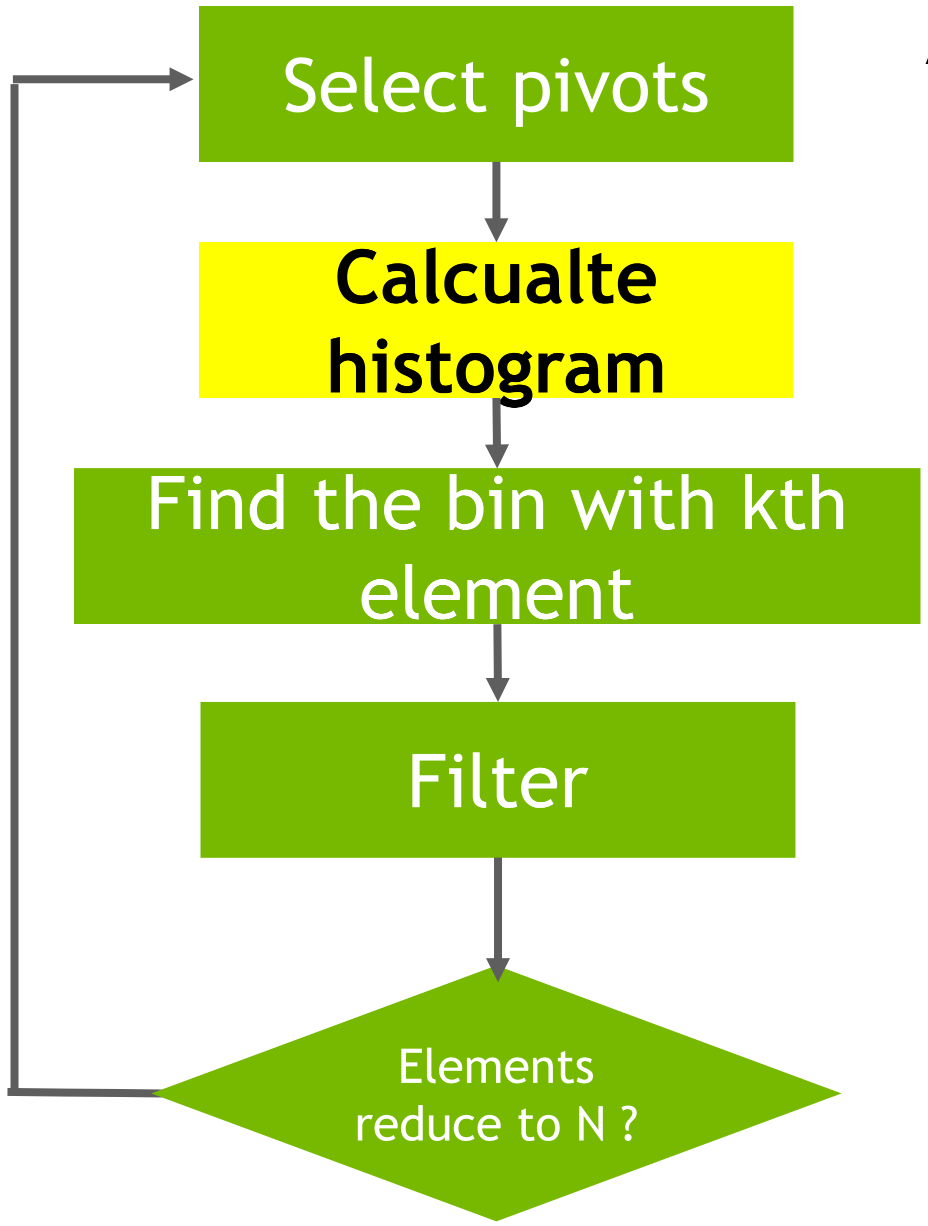
```
atomicAdd(&(b_hist[idx_digit]), 1);
```

Add to the global of histogram

```
atomicAdd(&(g_hist[idx_digit]), b_hist[idx_digit]);
```

PARALLEL IMPLEMENTATION

Calculate histogram



As computed in CUB

Digit value

```
mask = float_flip((unsigned int&)in_buf[index]);
mask = mask>>(32- BIT_HIST*(digit+1));
idx_digit = mask&(0X00ff);
```

1	2	3	4	5	7	9	2	3	4	5	6	4	2	2	3	4	4	4	5	5	6	6	7	8	9	0	1	2	2	3	4

Peer mask

```
peer_mask=match_any(digit)
```

For idx_digit=1

1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0

For idx_digit=2

0	1	0	0	0	0	0	1	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0

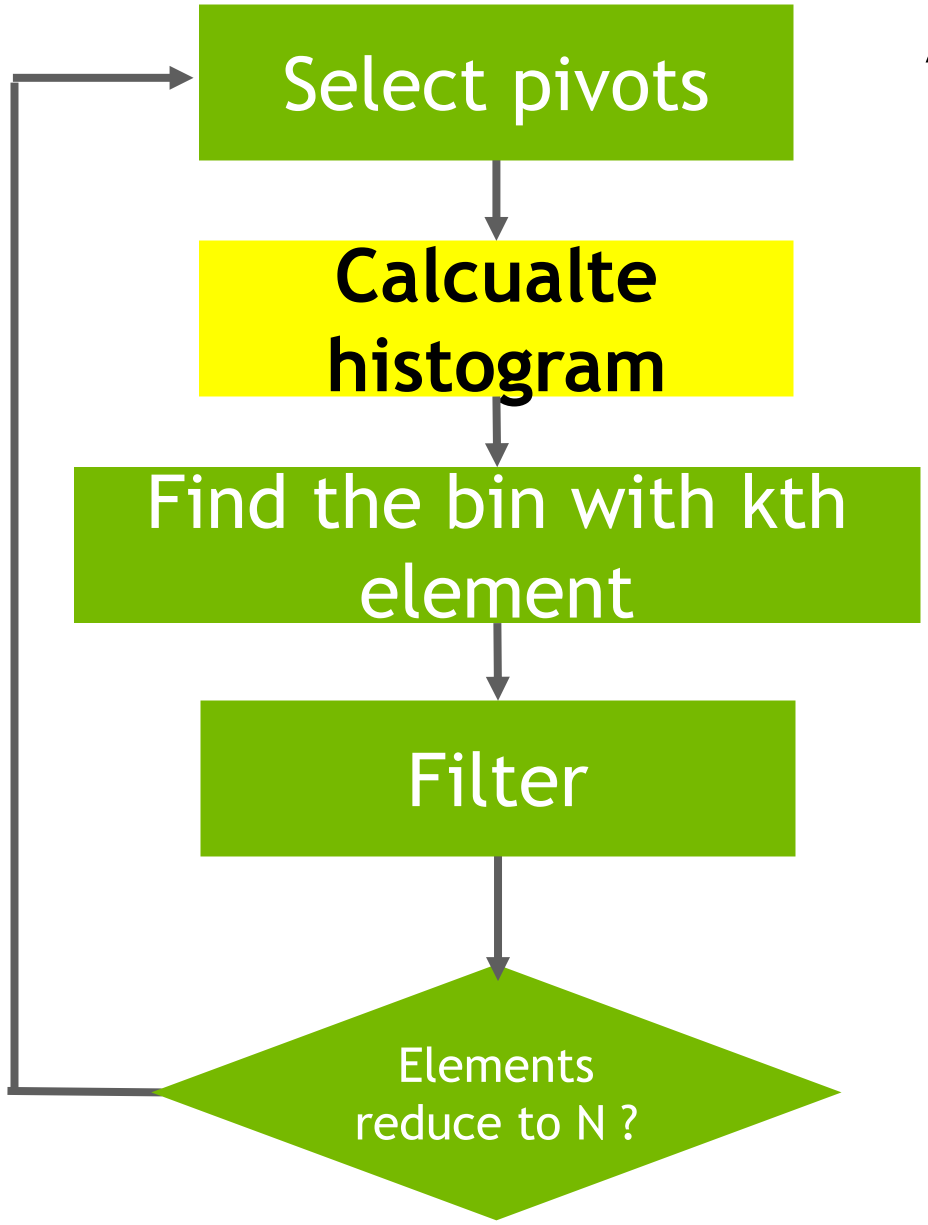
```
count=__popc(peer_mask)
```

6

S21572, Andy Adinets, A FASTER RADIX SORT IMPLEMENTATION
https://developer.nvidia.com/blog/cuda-pro-tip-optimized-filtering-warp-aggregated-atomics/

PARALLEL IMPLEMENTATION

Calculate histogram



As computed in CUB

0 1 0 0 0 0 0 1 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0

count=__popc(peer_mask)

count: 6

For lane_id=1

`LaneMaskLt()`

1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

`pre_count=__popc(peer_mask & LaneMaskLt())`

pre_count: 0

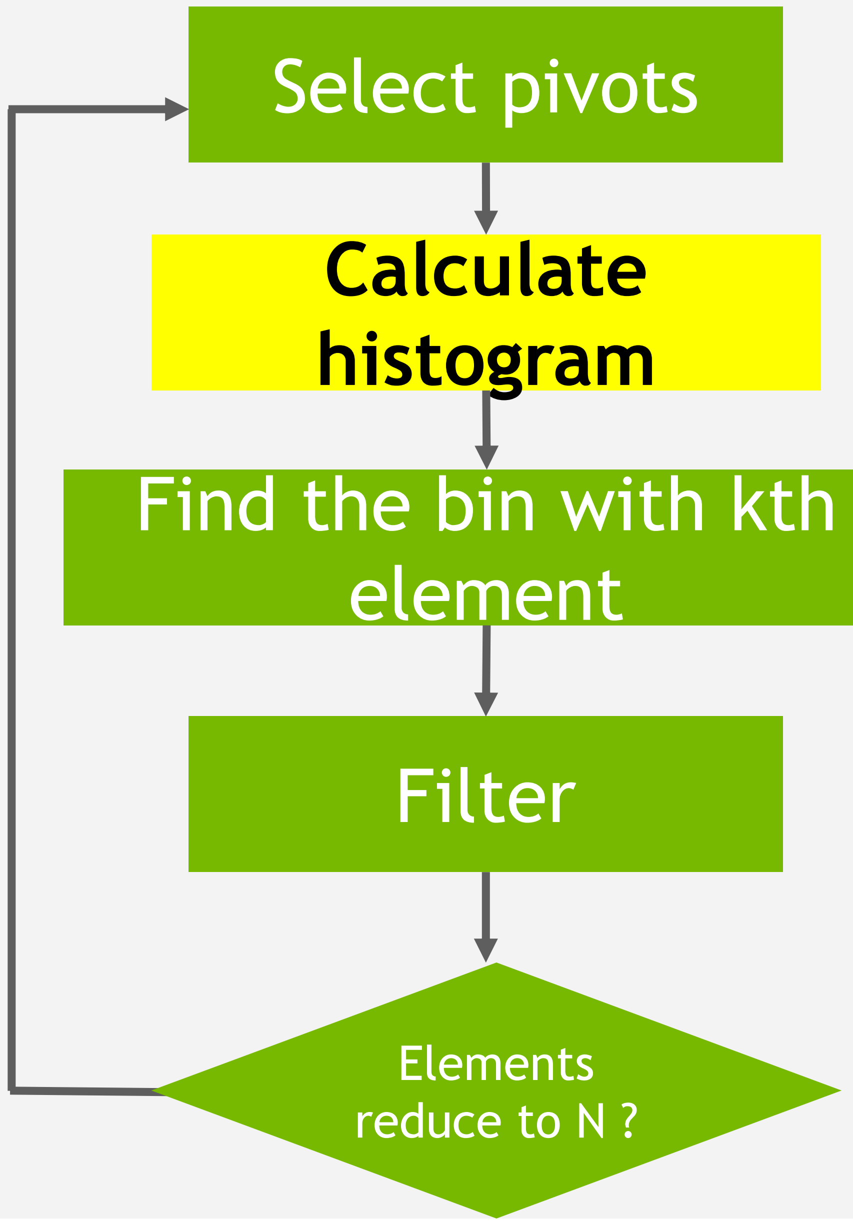
if(pre_count==0) add to histogram

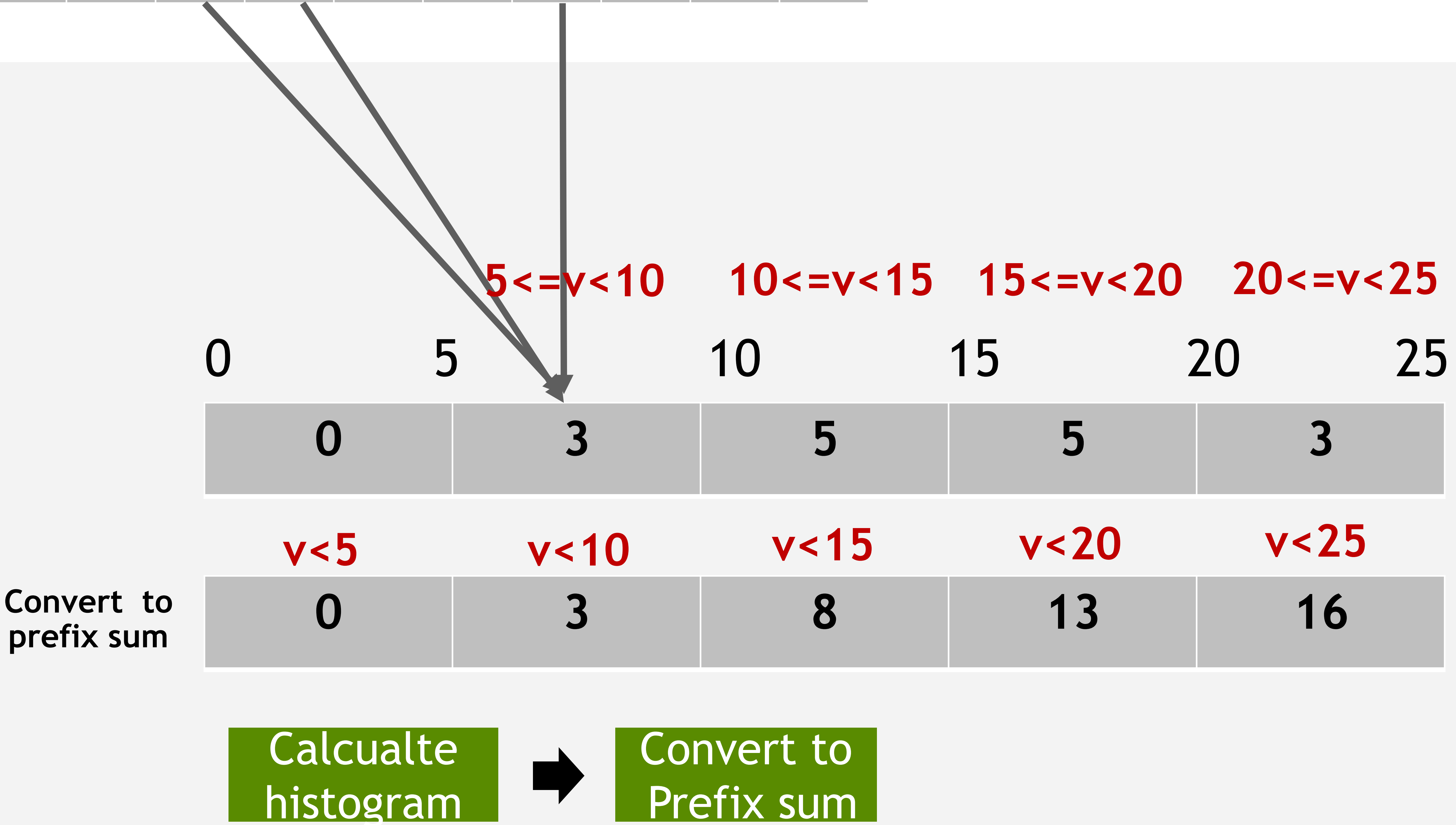
```
atomicAdd(&(b_hist[idx_digit]), 1);
```

BUCKET/RANDOM /RADIX SELECT

Id	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Arr	11	15	13	12	18	16	14	10	9	8	20	17	7	23	22	19

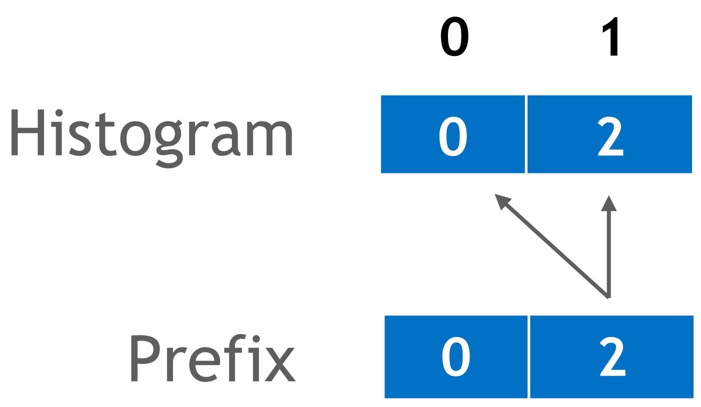
k	7

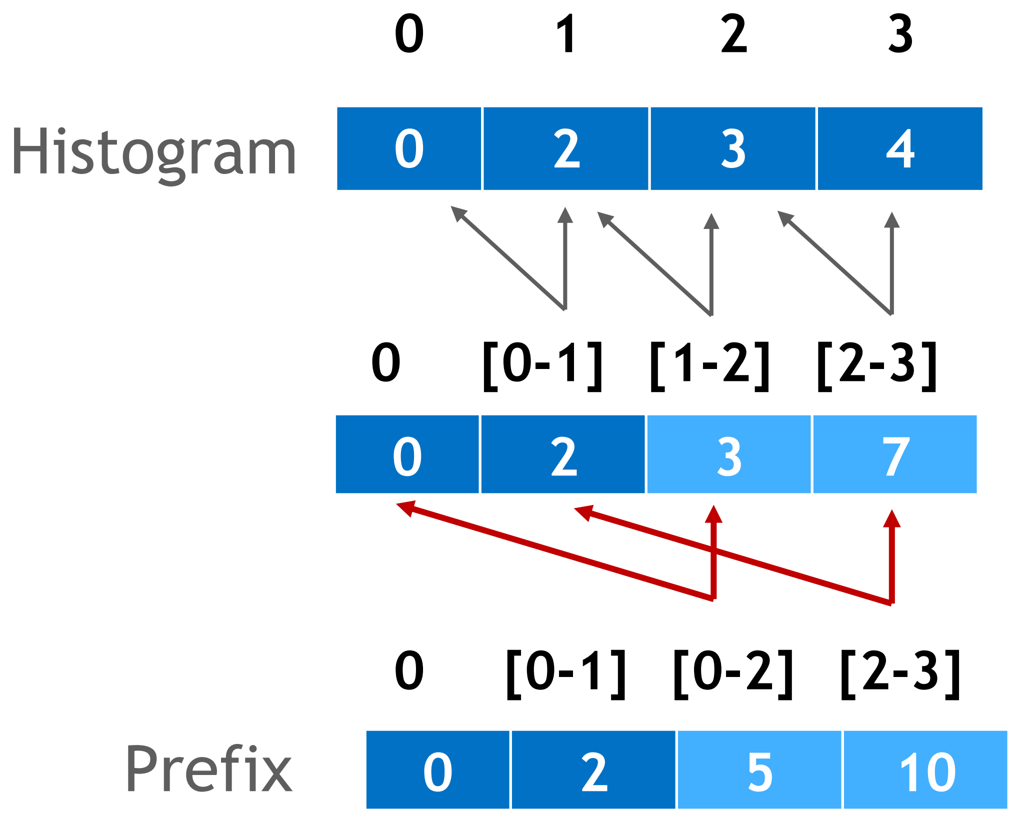




PARALLEL IMPLEMENTATION

Calculate histogram



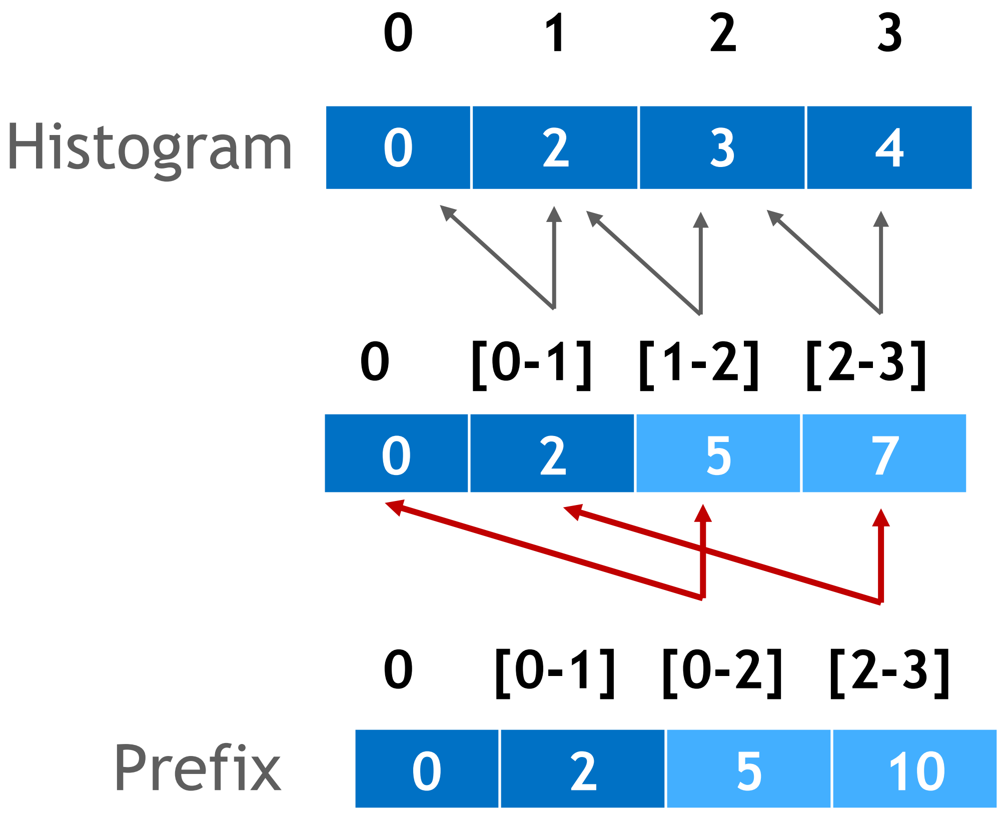


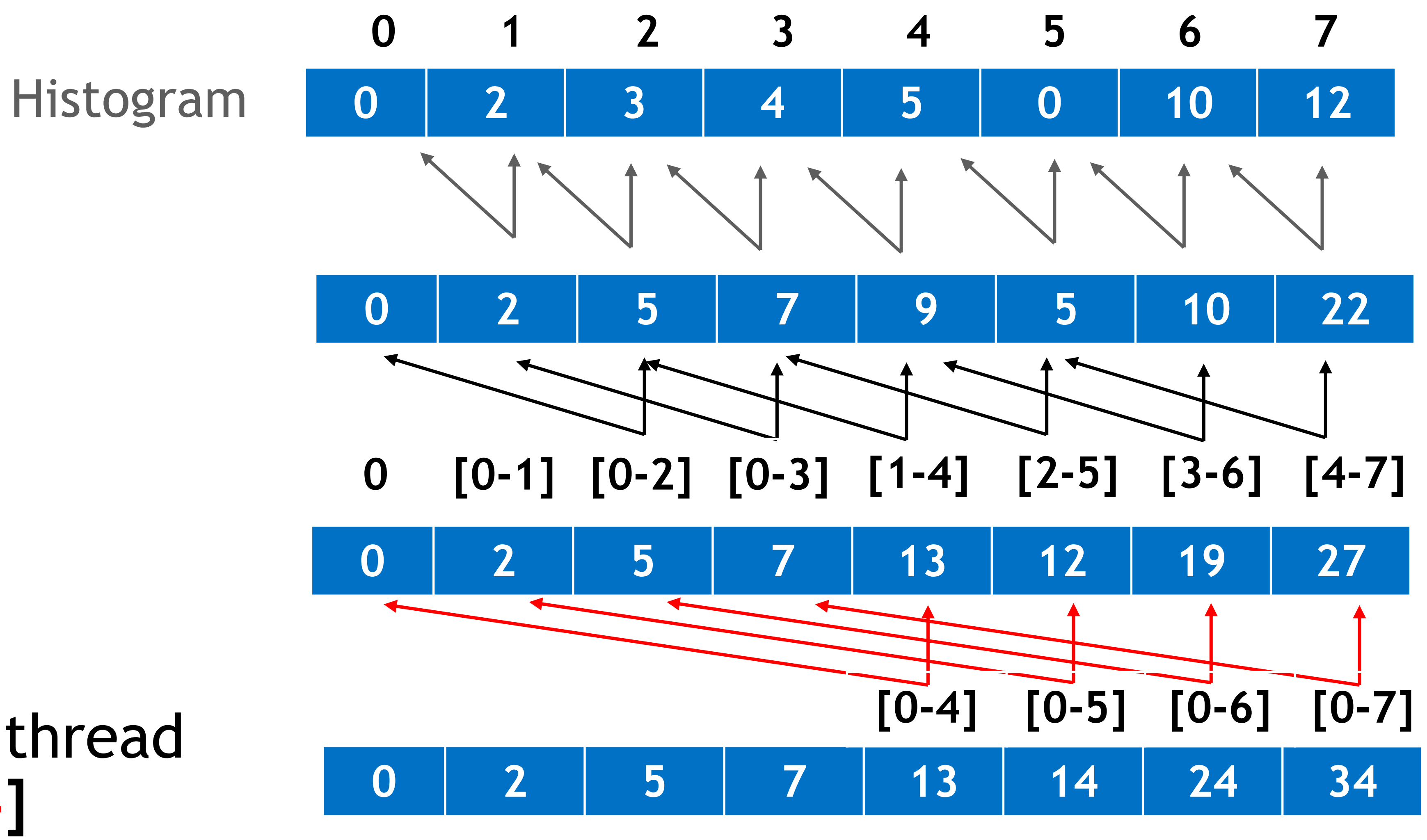
Except the first thread, each thread adds its element with its **previous** one

Except the first 2 thread, each thread adds its element **h[tid] with h[tid-2]**

PARALLEL IMPLEMENTATION

Calculate histogram

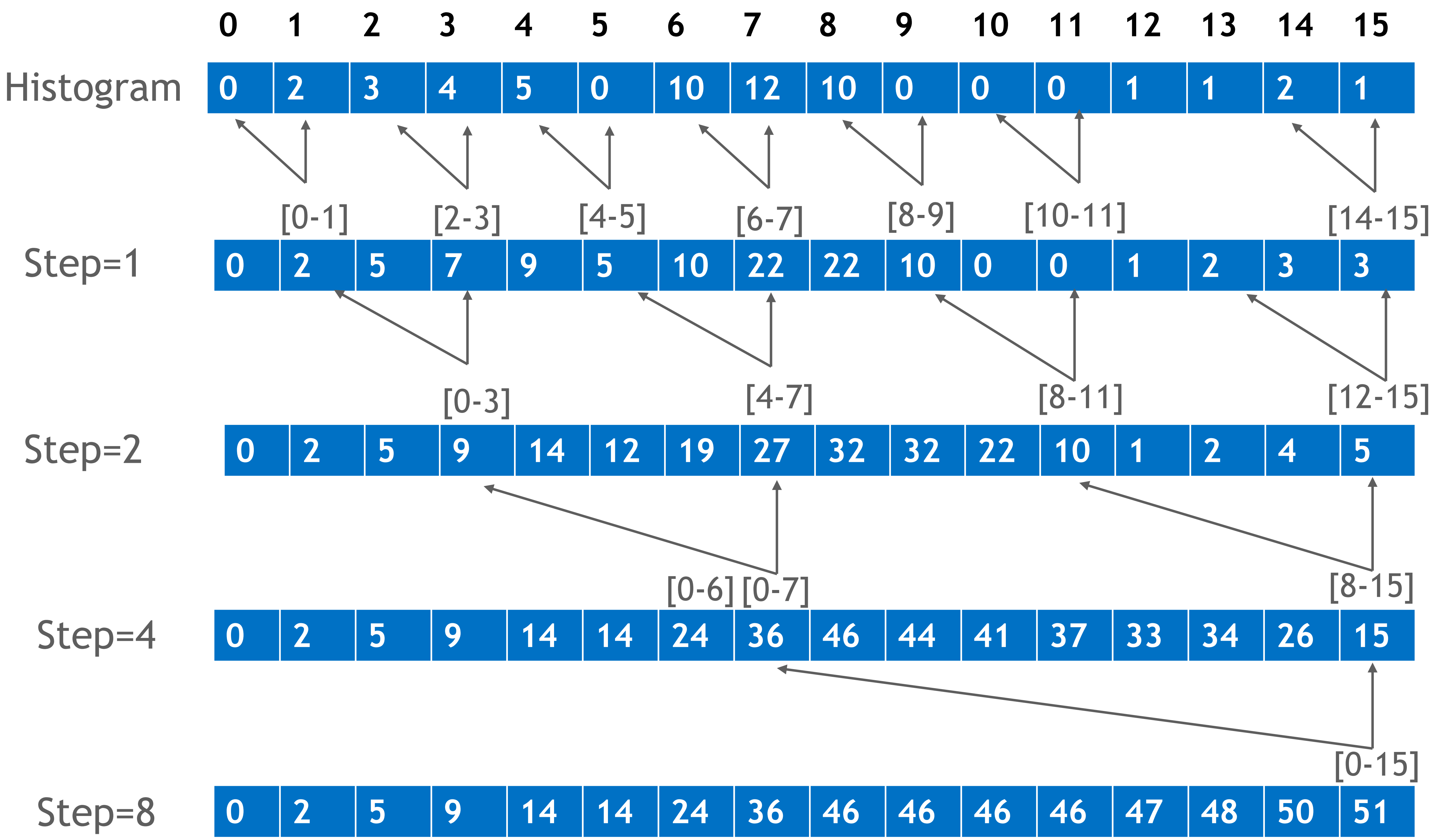




Except the first 4 thread, each thread adds its element **h[tid] with h[tid-4]**

PARALLEL IMPLEMENTATION

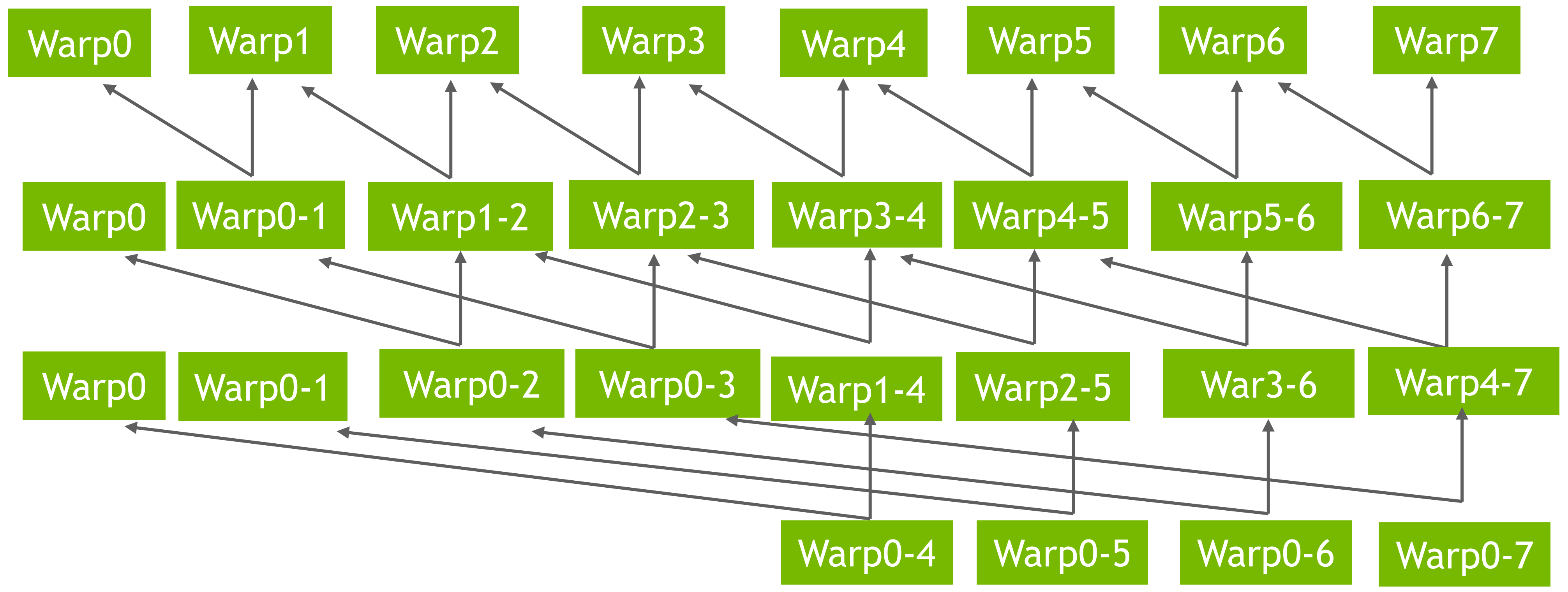
Calculate histogram




```
for (int i = 1; i < WARP_SIZE; i *= 2) {
    n = __shfl_up_sync(WARP_MASK,
         value, i,WARP_SIZE);
    if (index >= i) value += n;
}
```

PARALLEL IMPLEMENTATION

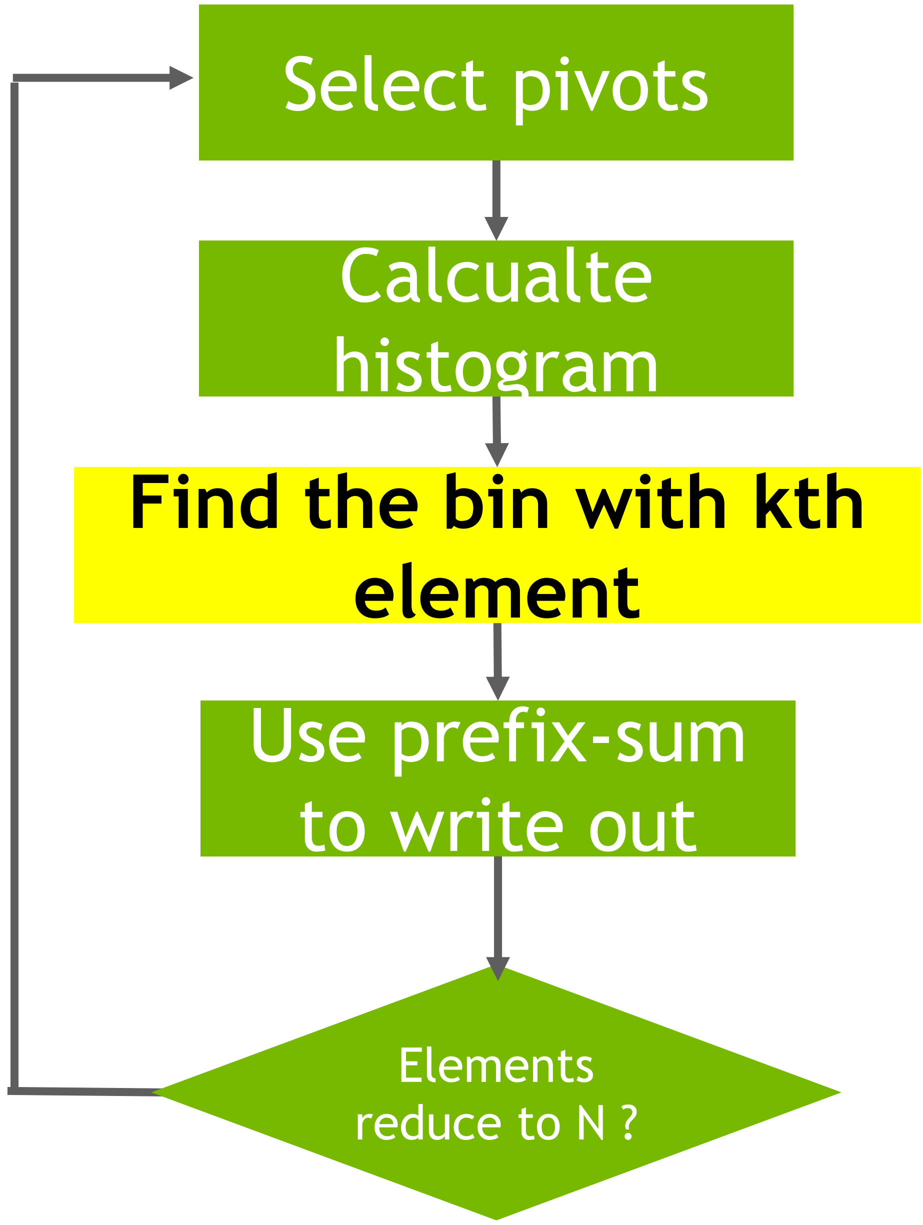
Calculate histogram

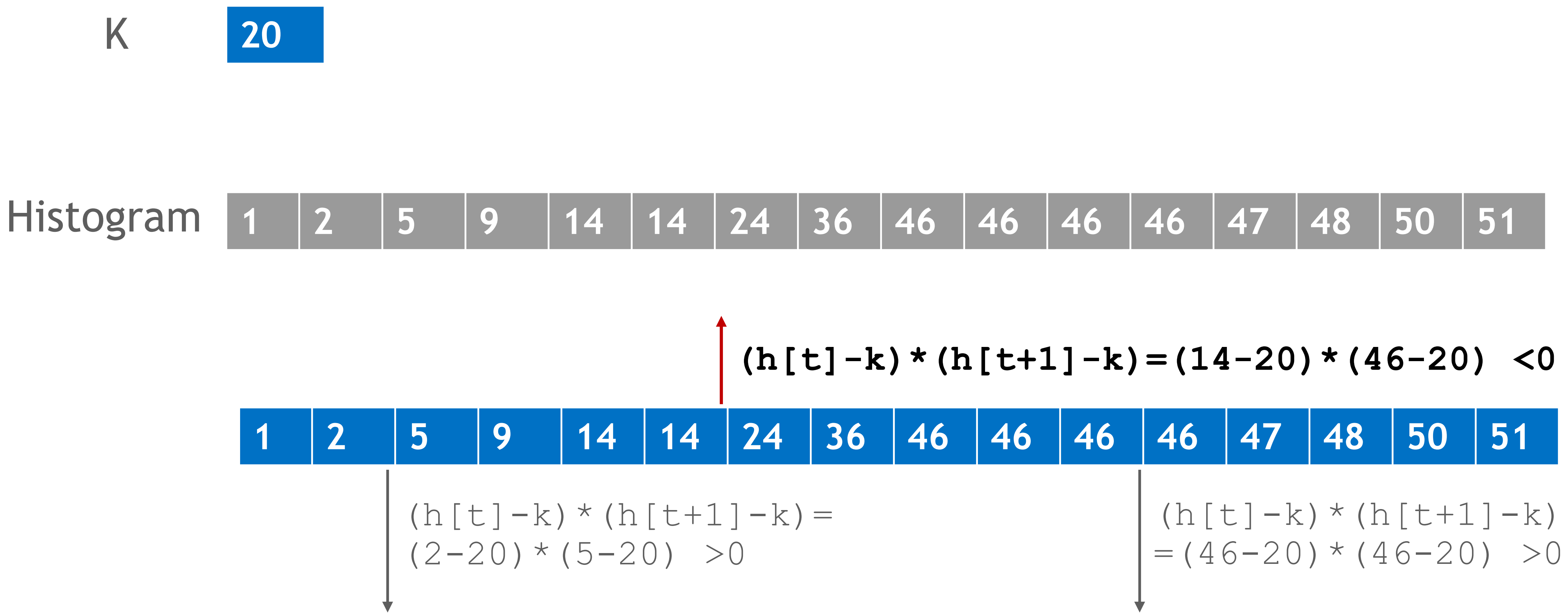



```
for(int i=1;i<NUM_WARP;i*=2){
      n = __shfl_up_sync(0xffffffff,
            value, i,NUM_WARP);
      if(thread_id>=i&&thread_id<NUM_WARP)
           value +=n;
}
```

PARALLEL IMPLEMENTATION

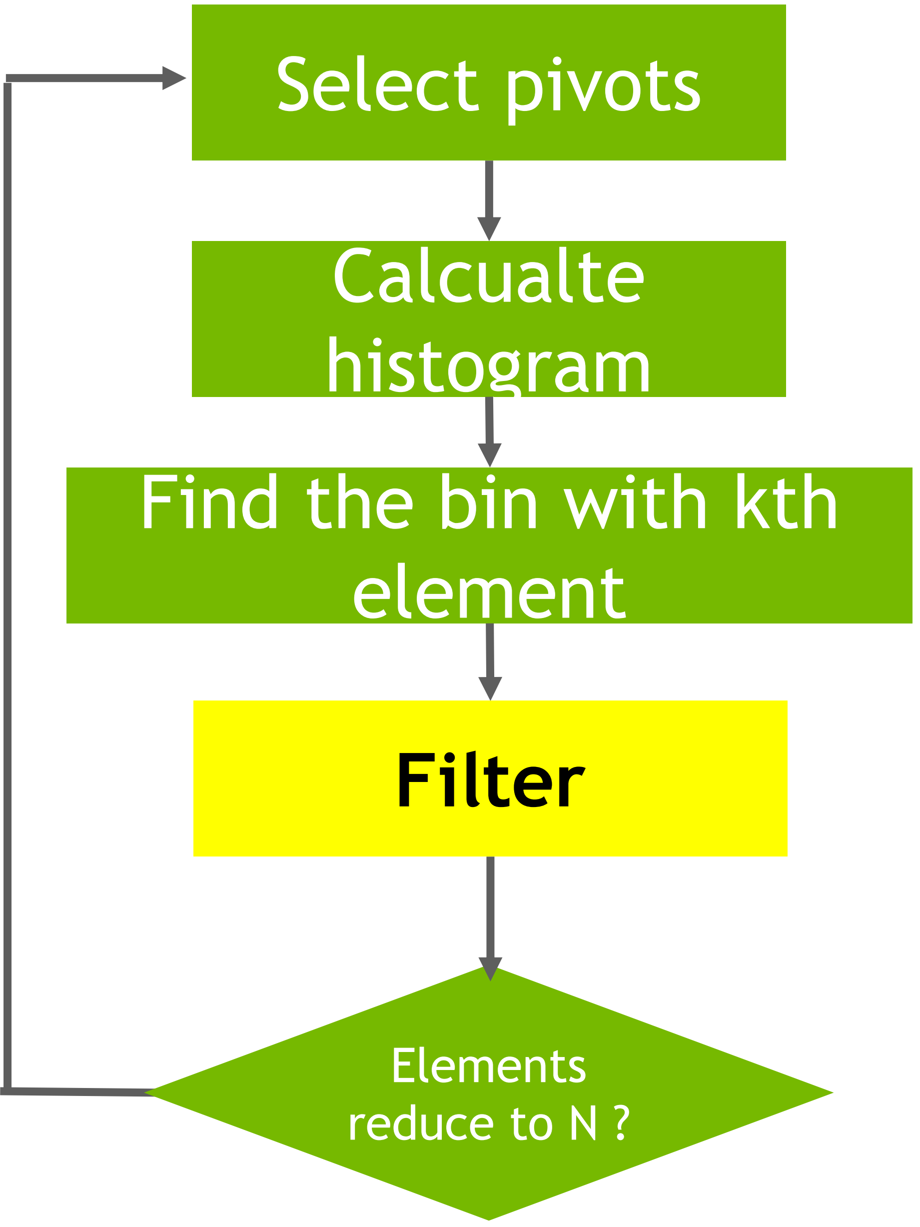
Calculate histogram





PARALLEL IMPLEMENTATION

Filter



data

40	30	21	9	41	42	53	56	58	70	15	13	12	7	8	16

In the bin or not

0	0	0	1	0	0	0	0	0	0	0	0	0	1	1	0

block_pos

0

Update pos &write to block cache

```
unsigned old_pos = atomicAdd(&block_pos, (unsigned)1);
buf[old_pos] = val;
```

block cache

9	7	8

global cache

```
out_pos = atomicAdd(&(global_pos), block_pos);
if (tid < block_pos) {
    out[out_pos + tid] = buf[tid];
}
```

6	4	3	…	9	7	8	…

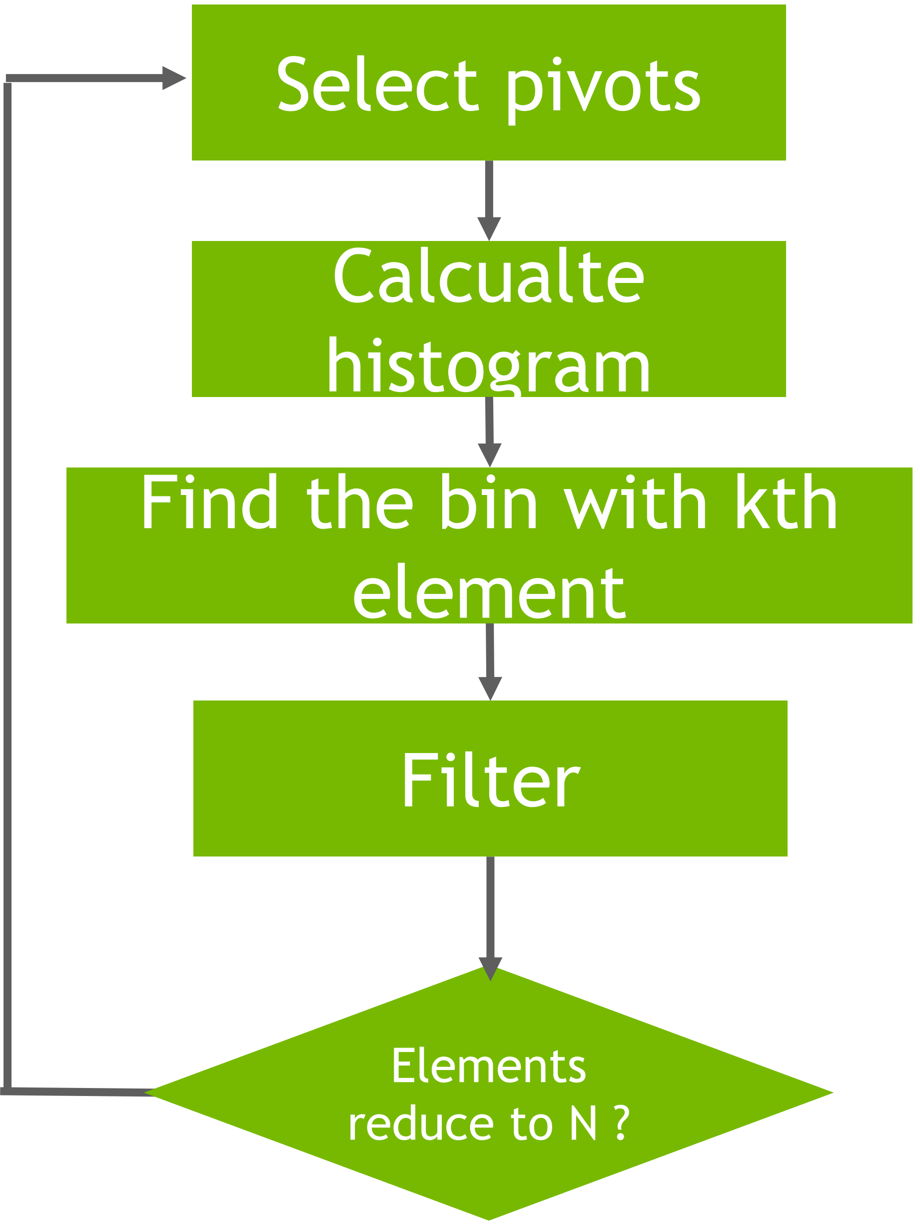
OUTLINE

Accelerate Radix Select with CUDA

- Radix select introduction
- Parallel implementation
- Implementation for different data size

IMPLEMENTATION FOR DIFFERENT DATA SIZE

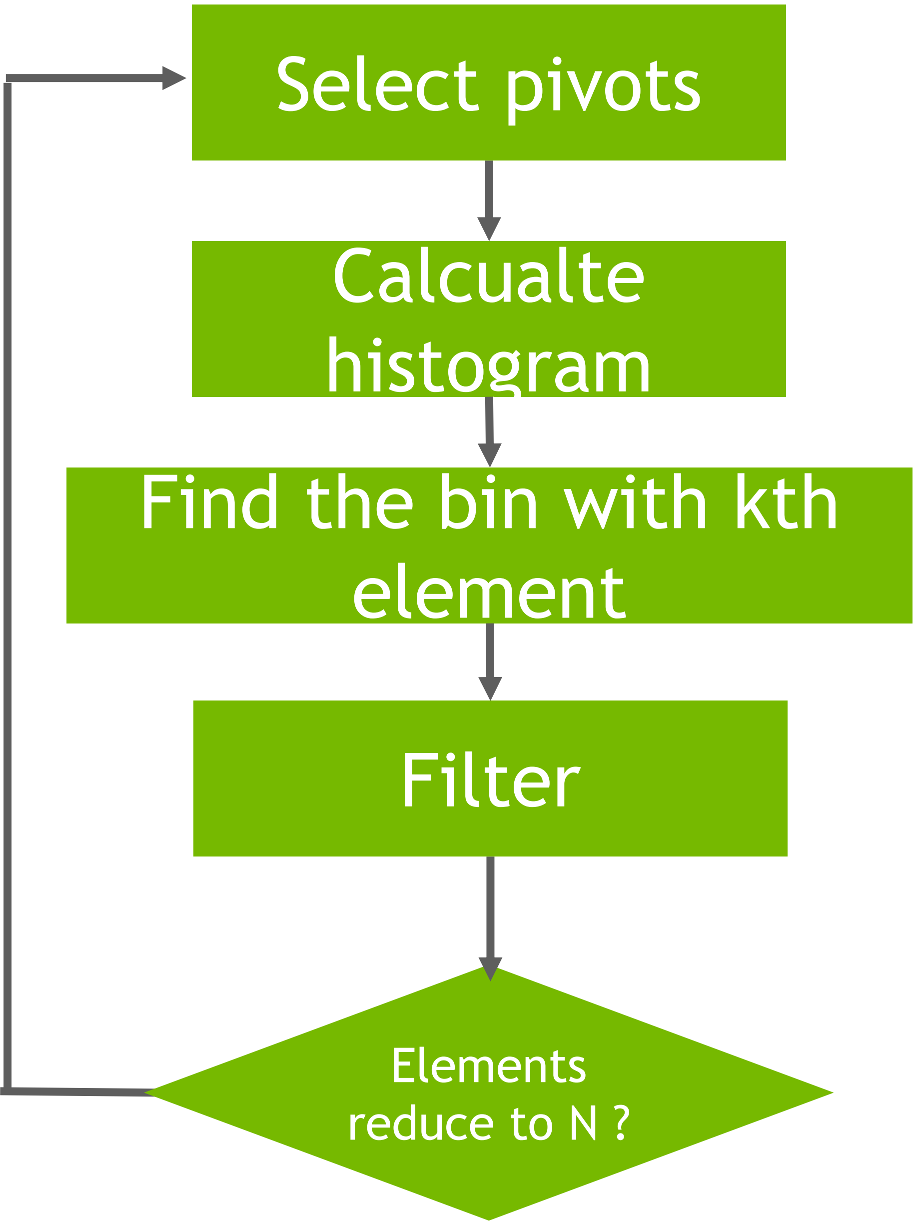
Device-wide




```
for (unsigned int digit = 0; digit < NUM_DIGIT; digit++) {
    //Step 1 Generate Histogram
    dim3 hist_blocks(batch_size, (len-1) / ITEM_PER_BLOCK +1);
    radix_histogram<T,idxT><<<hist_blocks, BLOCK_DIM,0, stream>>>(in_buf,d_info,digit,len,greater);
    CUDA_CHECK(cudaDeviceSynchronize());

    //Step 2 Select the Kth bucket and reduce candidates
    dim3 locate_blocks(batch_size,1);
    contract_interval_kernel<<<locate_blocks, NUM_HIST,0, stream>>>(d_info,greater);
    CUDA_CHECK(cudaDeviceSynchronize());

    //Step 3 Reduce data to Kth bucket
    dim3 filter_blocks(batch_size,(len-1) / ITEM_PER_BLOCK +1);
    filter_kernel<<<filter_blocks, NUM_HIST,0, stream>>>(out_buf,in_buf,d_info, d_k_value,digit,len,greater);
    CUDA_CHECK(cudaDeviceSynchronize());
    CUDA_CHECK(cudaMemcpyAsync(
                h_info, d_info, sizeof(StatInfo<T,idxT>)*batch_size, cudaMemcpyDeviceToHost, stream));


    in_buf=(digit%2==0)?d_buf1: d_buf2;
    out_buf=(digit%2==0)?d_buf2: d_buf1;
}//end got digit
```

IMPLEMENTATION FOR DIFFERENT DATA SIZE

Device-wide

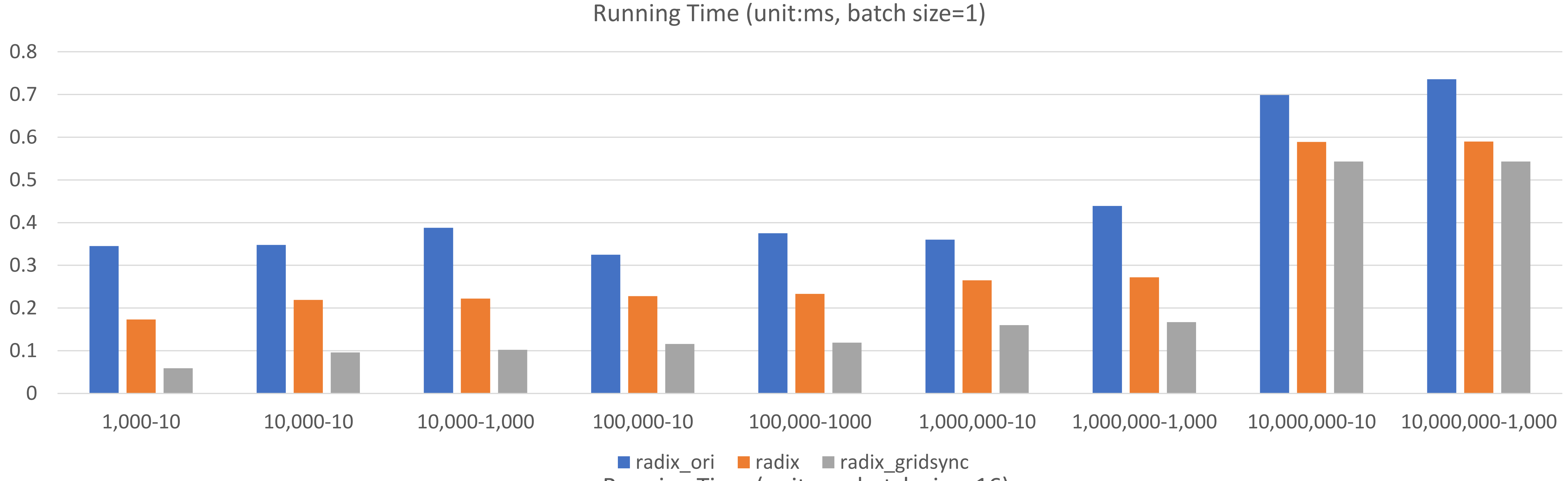


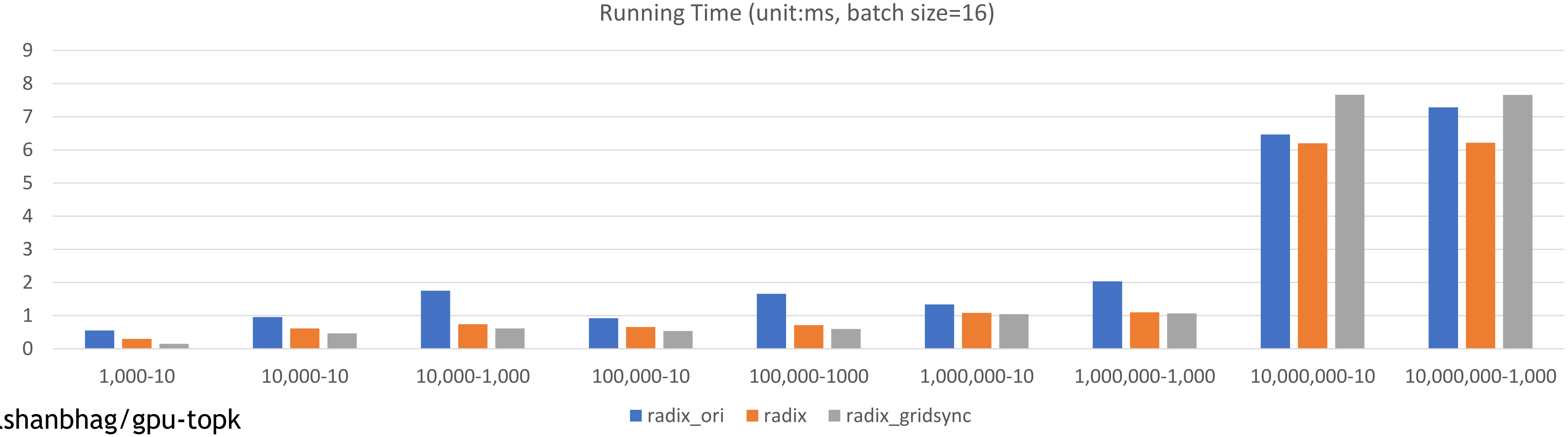

```
for (unsigned int digit = 0; digit < NUM_DIGIT; digit++) {
    //Step 1 Generate Histogram
    histogram(block_histo,in_buf,stat_info,digit,len,greater);
    grid.sync();

    //Step 2 Select the Kth bucket and reduce candidates
    if(blockIdx.y==0){
        contract_interval(stat_info,greater);
    }
    grid.sync();

    //Step 3 Reduce data to Kth bucket
    filter(filter_buf,warp_pos,out_buf,in_buf,stat_info,k_value,digit,len,greater);
    grid.sync();

    in_buf=(digit%2==0)?d_buf1: d_buf2;
    out_buf=(digit%2==0)?d_buf2: d_buf1;
}//for end digit
```

IMPLEMENTATION FOR DIFFERENT DATA SIZE

Device-wide

N-k :1,000-10 indicates find the smallest k elements from 1000 elements

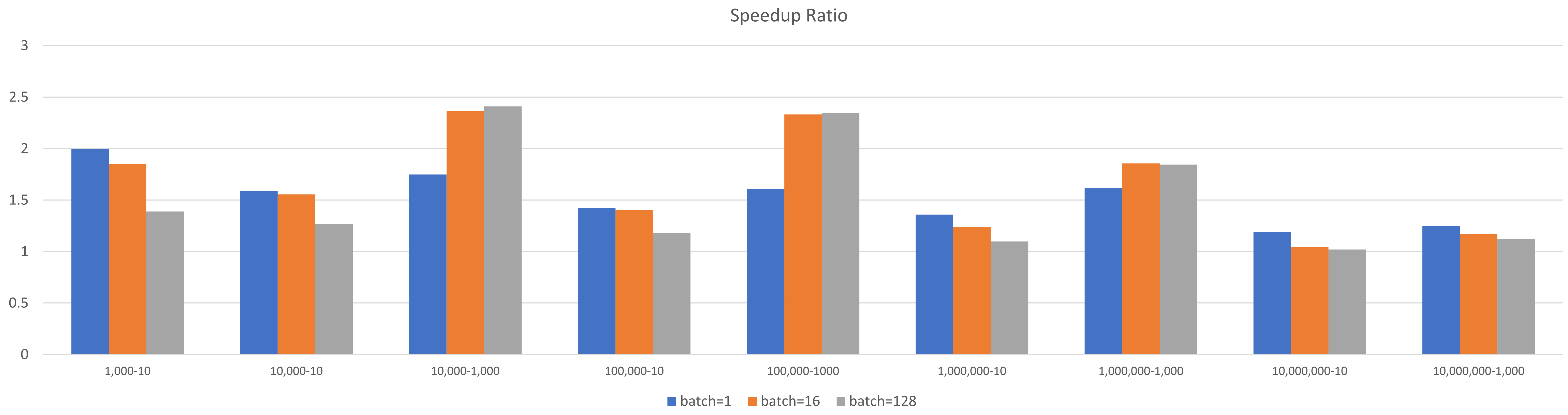




https://github.com/anilshanbhag/gpu-topk

IMPLEMENTATION FOR DIFFERENT DATA SIZE

Device-wide

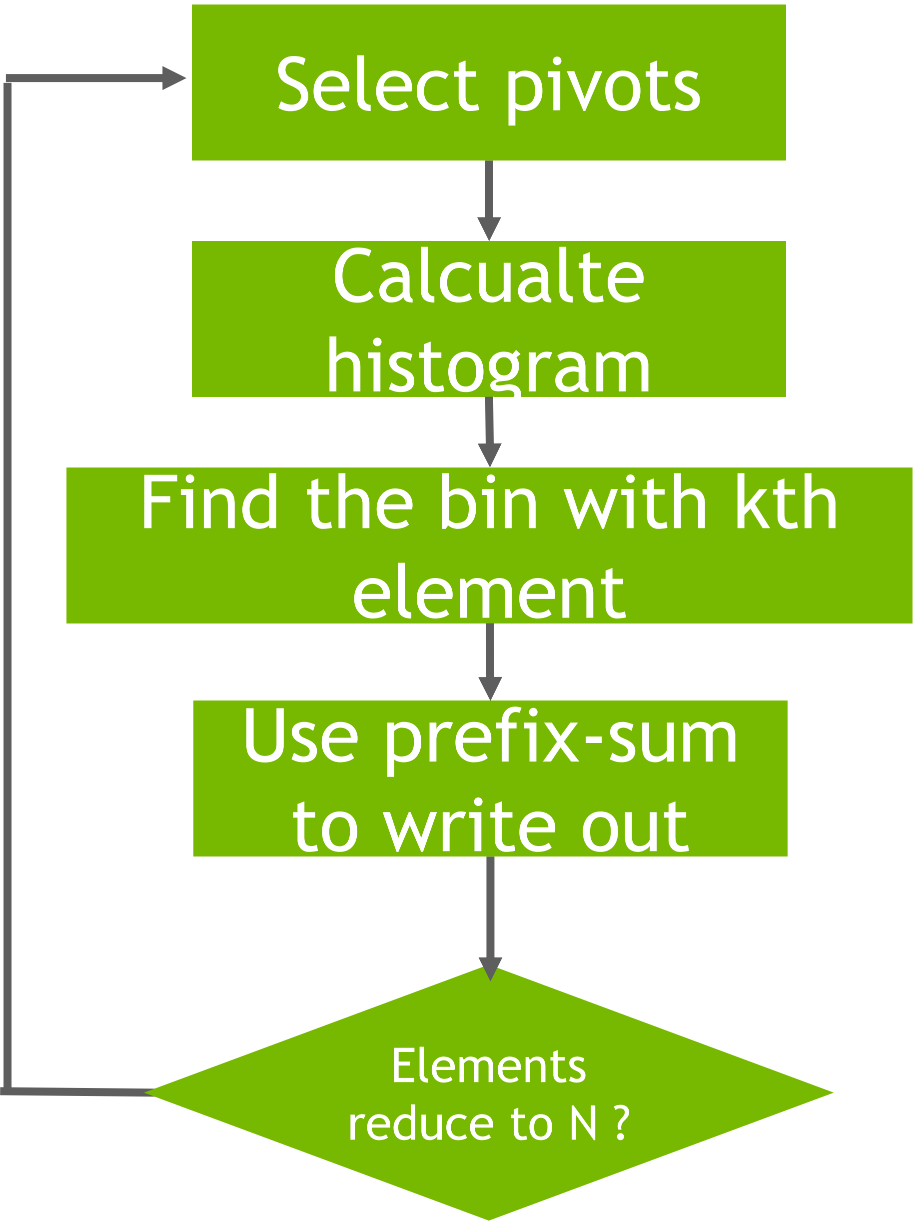


For large K value:

Experiement			Speedup Ratio	
batch	len	k	radix_ori/radix	radix_ori/radix_gridsync
1	1,000,000	100,000	26.70	42.82
1	10,000,000	100,000	12.89	13.73
1	100,000,000	100,000	2.77	2.36
16	1,000,000	100,000	88.75	91.18
16	10,000,000	100,000	16.80	14.15
128	1,000,000	100,000	90.75	72.15
128	10,000,000	100,000	16.75	10.53

IMPLEMENTATION FOR DIFFERENT DATA SIZE

Block-wide

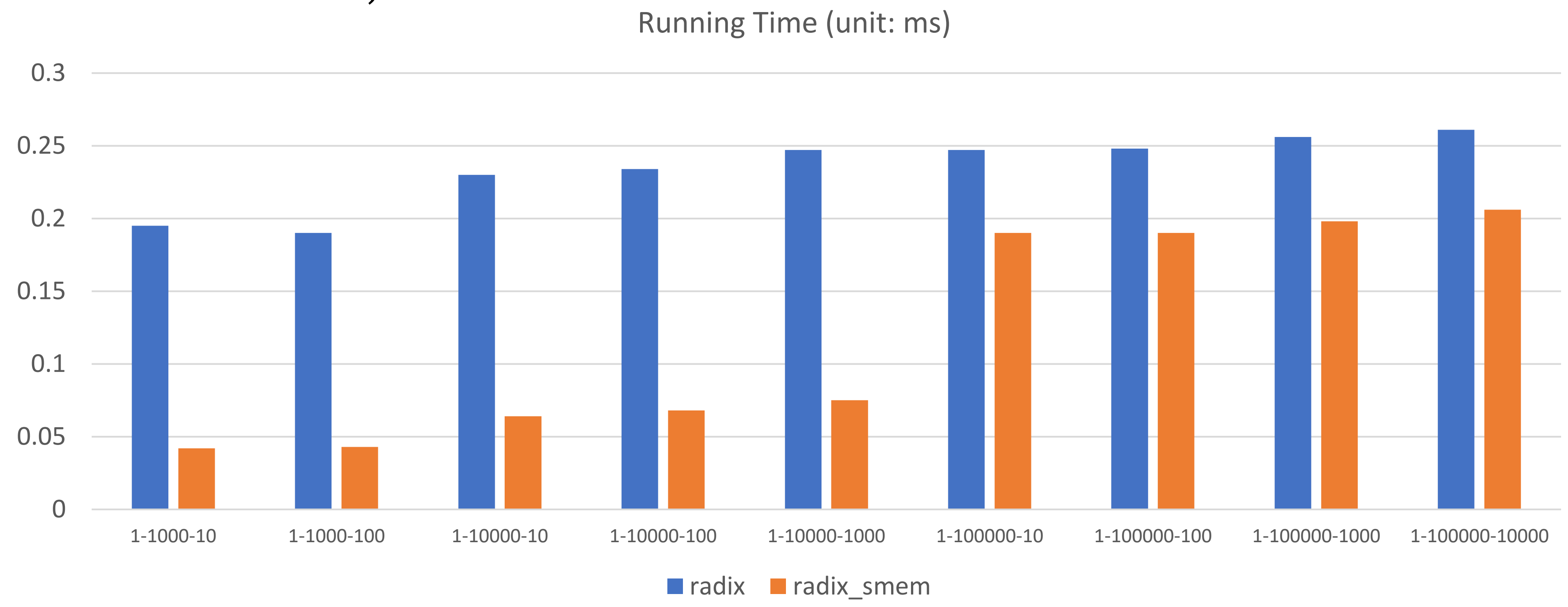


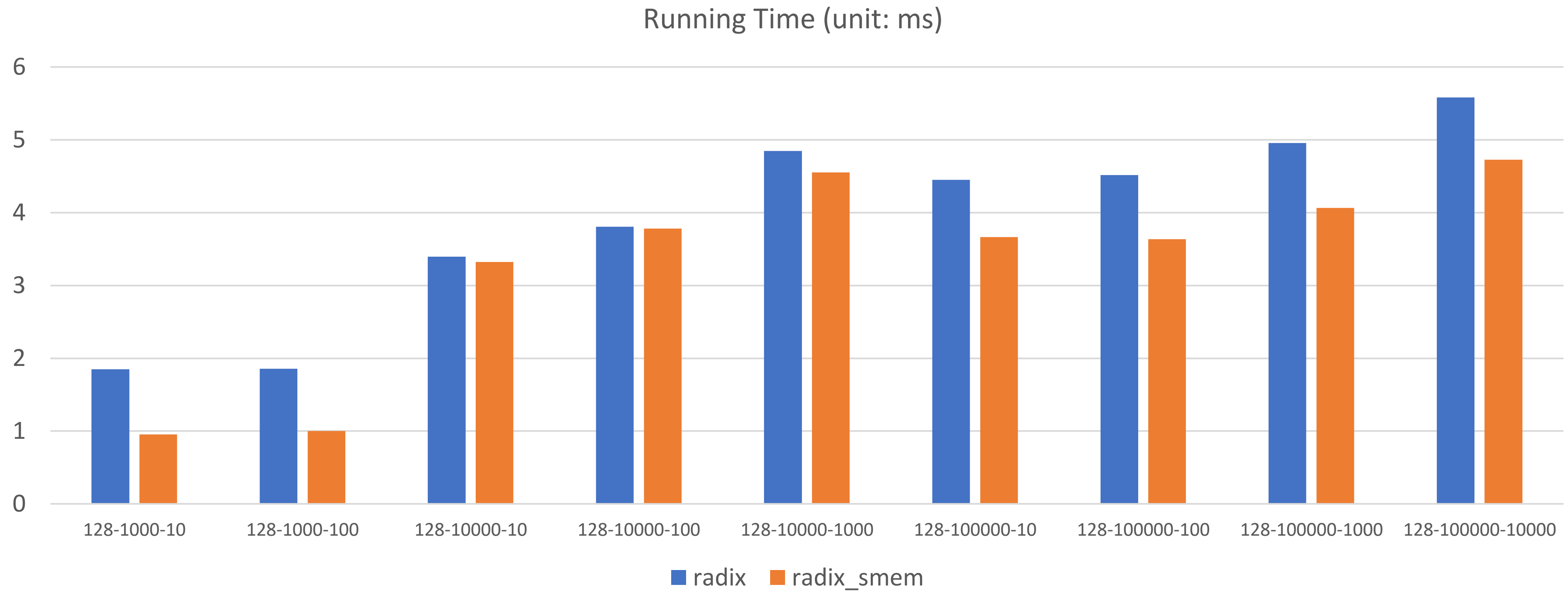

```
#pragma unroll
for (unsigned int digit = 0; digit < NUM_DIGIT; digit++) {
    //Step 1 Generate Histogram
    histogram_smem(index,in_buf,&info.hist[0],info.len,info.previous_len,digit,greater);

    //Step 2 Select the Kth bucket and reduce candidates
    const idxT k=info.k;
    __syncthreads();
    contract_interval_smem(k, first, second,info,greater);

    //Step 3 Reduce data to Kth bucket
    filter_smem(digit,&warp_pos[0],k_value,in_buf,out_buf,info);

    //Step 4 Switch between in and out
    in_buf=(digit%2==0)?buf1: buf2;
    out_buf=(digit%2==0)?buf2: buf1;
}//for (unsigned int digit = 0; digit < NUM_DIGIT; digit++)
```

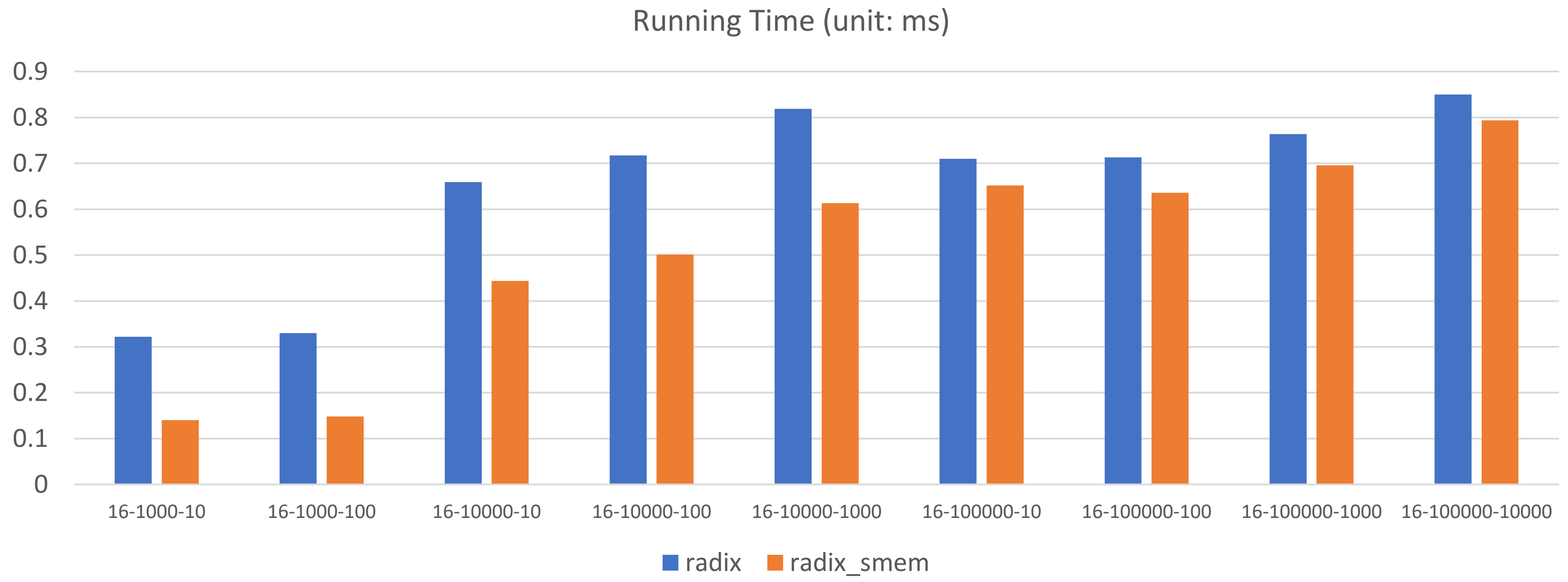
IMPLEMENTATION FOR DIFFERENT DATA SIZE

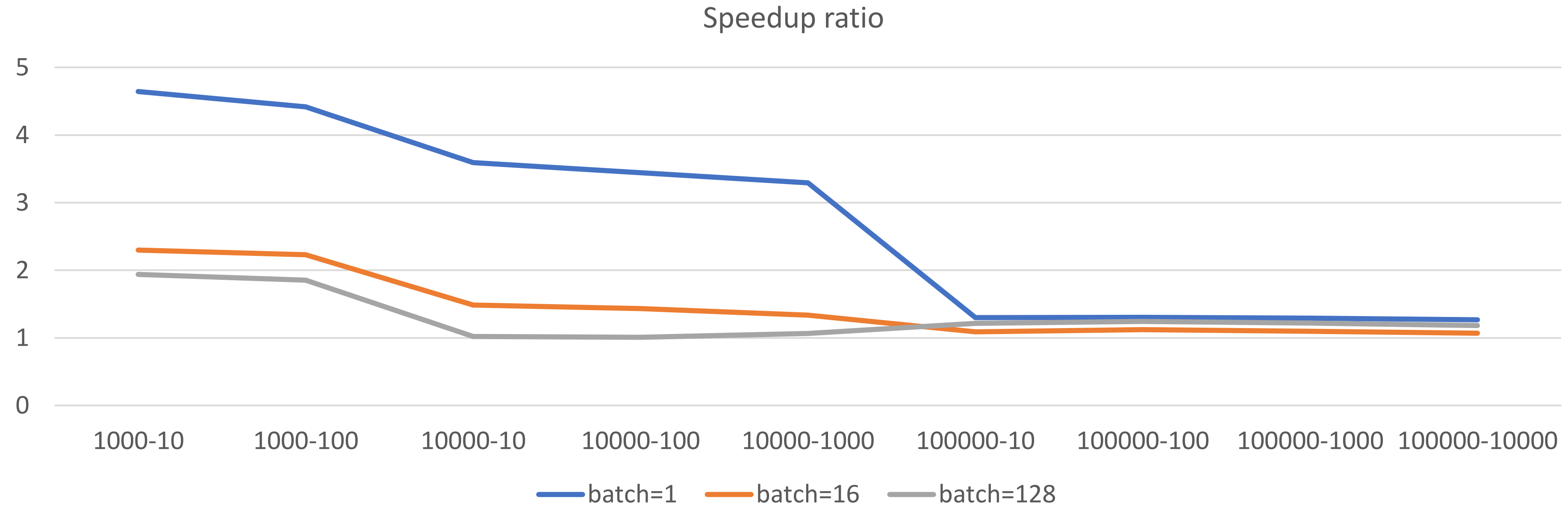
Block-wide

Batch-N-k :16-1,000-10 indicates find the smallest k elements from 1000 elements with batch size 16

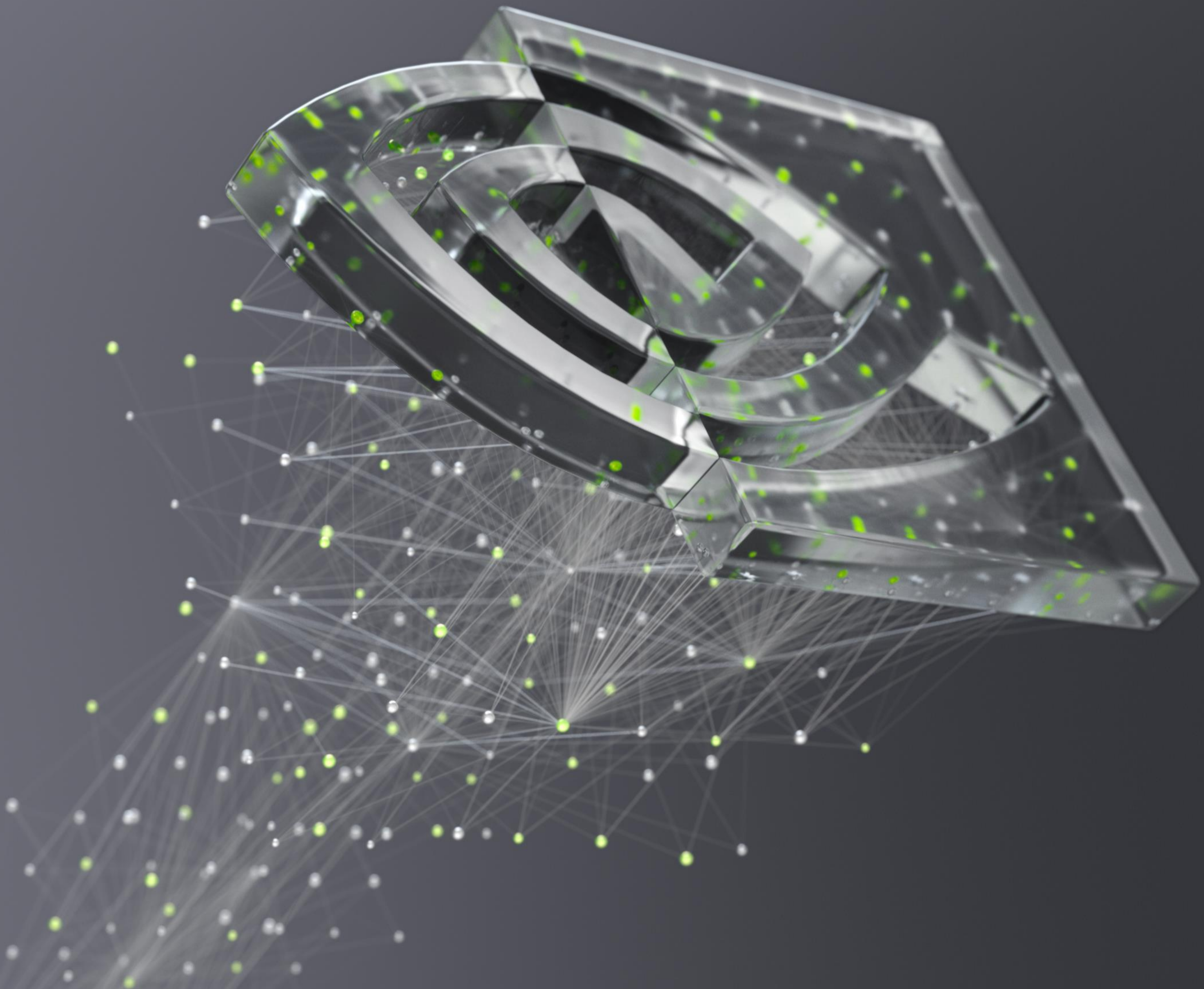








https://github.com/anilshanbhag/gpu-topk



NVIDIA