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 $N = 8$, $k = 3$:
TWO USEFUL ENHANCEMENTS IN PRACTICE

batch processing

<table>
<thead>
<tr>
<th>3</th>
<th>4</th>
<th>6</th>
<th>1</th>
<th>5</th>
<th>8</th>
<th>2</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>2</td>
<td>6</td>
<td>1</td>
<td>8</td>
<td>3</td>
<td>7</td>
<td>5</td>
</tr>
<tr>
<td>8</td>
<td>6</td>
<td>4</td>
<td>5</td>
<td>2</td>
<td>7</td>
<td>3</td>
<td>1</td>
</tr>
</tbody>
</table>

return also the index

<table>
<thead>
<tr>
<th>3</th>
<th>4</th>
<th>6</th>
<th>1</th>
<th>5</th>
<th>8</th>
<th>2</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>index 0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
</tbody>
</table>
ALGORITHM 1: SORTING

input 3 4 6 1 5 8 2 7

sorted 1 2 3 4 5 6 7 8

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.

(input) 3 4 6 1 5 8 2 7

(selection) 3 4 6 1 5 8 2 7

(filter by) <= 3

3 1 2

(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:

<table>
<thead>
<tr>
<th>sorting</th>
<th>selection</th>
</tr>
</thead>
<tbody>
<tr>
<td>quicksort</td>
<td>quick select</td>
</tr>
<tr>
<td>samplesort</td>
<td>sample select</td>
</tr>
<tr>
<td>radix sort</td>
<td>radix select</td>
</tr>
</tbody>
</table>

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
**WARPSSELECT**

_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

<table>
<thead>
<tr>
<th>#thread</th>
<th>intermediate memory</th>
<th>sync cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>warp</td>
<td>32  register (lowest latency)</td>
<td>low</td>
</tr>
<tr>
<td>block</td>
<td>[32, 1024] shared memory (low latency)</td>
<td>medium (__syncthreads())</td>
</tr>
<tr>
<td>multi-block</td>
<td>can be large global memory (high latency)</td>
<td>high (e.g. extra kernel launch)</td>
</tr>
</tbody>
</table>

- 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 Tesla 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

Select pivots

Calculate histogram

Find the bin with kth element

Filter

For example:

\[ 0 < 5 < 10 \]

<table>
<thead>
<tr>
<th>Id</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arr</td>
<td>11</td>
<td>15</td>
<td>13</td>
<td>12</td>
<td>18</td>
<td>16</td>
<td>14</td>
<td>10</td>
<td>9</td>
<td>8</td>
<td>20</td>
<td>17</td>
<td>7</td>
<td>23</td>
<td>22</td>
<td>19</td>
</tr>
</tbody>
</table>

BUCKET/RANDOM /RADIX SELECT

Select pivots

Calculate histogram

Find the bin with kth element

Filter

Elements reduce to N?

<table>
<thead>
<tr>
<th>Id</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arr</td>
<td>11</td>
<td>15</td>
<td>18</td>
<td>12</td>
<td>16</td>
<td>14</td>
<td>10</td>
<td></td>
<td>9</td>
<td>8</td>
<td>20</td>
<td>17</td>
<td>7</td>
<td>23</td>
<td>22</td>
<td>19</td>
</tr>
</tbody>
</table>

\( k \) = 7

<table>
<thead>
<tr>
<th>Bin</th>
<th>0</th>
<th>5</th>
<th>10</th>
<th>15</th>
<th>20</th>
<th>25</th>
</tr>
</thead>
<tbody>
<tr>
<td>(&lt;5)</td>
<td>0</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(&lt;10)</td>
<td>0</td>
<td>3</td>
<td>8</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(&lt;15)</td>
<td>0</td>
<td>3</td>
<td>8</td>
<td>13</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(&lt;20)</td>
<td>0</td>
<td>3</td>
<td>8</td>
<td>13</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>(&lt;25)</td>
<td>0</td>
<td>3</td>
<td>8</td>
<td>13</td>
<td>16</td>
<td></td>
</tr>
</tbody>
</table>

5\(\leq v<10\) 10\(\leq v<15\) 15\(\leq v<20\) 20\(\leq v<25\)
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

Select pivots
Calculate histogram
Find the bin with kth element
Filter

Elements reduce to N?

0  5  10  15  20  25
0  3  5  5  3
0 < 7 3 < 7 7 < 8 7 < 13 7 < 16

Find the bin with kth element
Convert to prefix sum

Select pivots
BUCKET/RANDOM /RADIX SELECT

<table>
<thead>
<tr>
<th>Id</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arr</td>
<td>11</td>
<td>15</td>
<td>13</td>
<td>12</td>
<td>18</td>
<td>16</td>
<td>14</td>
<td>10</td>
<td>9</td>
<td>8</td>
<td>20</td>
<td>17</td>
<td>7</td>
<td>23</td>
<td>22</td>
<td>19</td>
</tr>
</tbody>
</table>

Select pivots
Calculate histogram
Find the bin with kth element
Filter

Elements reduce to 1 ?

<table>
<thead>
<tr>
<th>0</th>
<th>5</th>
<th>10</th>
<th>15</th>
<th>20</th>
<th>25</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>3</td>
<td>8</td>
<td>13</td>
<td>16</td>
<td></td>
</tr>
</tbody>
</table>

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

k 4
BUCKET/RANDOM /RADIX SELECT

Select pivots

Calculate histogram

Find the bin with kth element

Filter

Elements reduce to 1?

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

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

Select pivots

Calculate histogram

Find the bin with kth element

Filter

Elements reduce to 1?
BUCKET/RANDOM /RADIX SELECT

Select pivots
Calculate histogram
Find the bin with kth element
Filter

Select pivots
Calculate histogram
Find the bin with kth element
Filter

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

k' | 4

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

10 11 12 13 14 15

1 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
For example:
32 bit unsigned int

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 0 0

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

```
0 0 0 0 0
1 fff ffff
2 fff ffff
3 fff ffff
4 fff ffff
5 fff ffff
...
15 fff ffff
```

```
3000 0000
31ff ffff
32ff ffff
33ff ffff
34ff ffff
35ff ffff
...
3fff ffff
```

```
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
```

```
3400 0000
341 ffff
342 ffff
343 ffff
344 ffff
345 ffff
...
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.

```c
__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

Select pivots

Calculate histogram

Find the bin with kth element

Filter

Elements reduce to N?

...
PARALLEL IMPLEMENTATION

Calculate histogram

- Select pivots
- Calculate histogram
- Find the bin with kth element
- Filter
- Elements reduce to N?

Digit value

```c
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
PARALLEL IMPLEMENTATION

**Calculate histogram**

1. Select pivots
2. Calculate histogram
3. Find the bin with kth element
4. Filter
5. Add to the block of histogram
6. Add to the global of histogram

Digit value:

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

Add to the block of histogram:

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

Add to the global of histogram:

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

Calculate histogram

Select pivots

Calculate histogram

Find the bin with kth element

Filter

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

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

Digit value

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>4</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

Peer mask

For `idx_digit`=1

```c
1 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

```c
0 1 0 0 0 0 1 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0
```

count=__popc(peer_mask)

6
PARALLEL IMPLEMENTATION

Calculate histogram

Select pivots →
Calculuate histogram →
Find the bin with kth element →
Filter

As computed in CUB

```
0 1 0 0 0 0 0 1 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

For lane_id = 1

```c
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
```

pre_count = __popc(peer_mask & LaneMaskLt())

0

if(pre_count == 0)

add to histogram

```c
atomicAdd(&b_hist[idx_digit]), 1);
```
### BUCKET/RANDOM / RADIX SELECT

<table>
<thead>
<tr>
<th>Id</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arr</td>
<td>11</td>
<td>15</td>
<td>13</td>
<td>12</td>
<td>18</td>
<td>16</td>
<td>14</td>
<td>10</td>
<td>9</td>
<td>8</td>
<td>20</td>
<td>17</td>
<td>7</td>
<td>23</td>
<td>22</td>
<td>19</td>
</tr>
</tbody>
</table>

#### Select pivots

#### Calculate histogram

#### Find the bin with kth element

#### Filter

#### Elements reduce to N?

#### Convert to prefix sum

#### Calculate histogram

#### Convert to Prefix sum

**Find the bin with kth element**

- **Select pivots**
- **Calculate histogram**
- **Find the bin with kth element**
- **Filter**
- **Elements reduce to N?**

**Convert to prefix sum**

<table>
<thead>
<tr>
<th>Bin</th>
<th>0</th>
<th>5</th>
<th>10</th>
<th>15</th>
<th>20</th>
<th>25</th>
</tr>
</thead>
<tbody>
<tr>
<td>Count</td>
<td>0</td>
<td>3</td>
<td>5</td>
<td>5</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>Elements Histogram</td>
<td>v&lt;5</td>
<td>v&lt;10</td>
<td>v&lt;15</td>
<td>v&lt;20</td>
<td>v&lt;25</td>
<td></td>
</tr>
<tr>
<td>Prefix Sum</td>
<td>0</td>
<td>3</td>
<td>8</td>
<td>13</td>
<td>16</td>
<td></td>
</tr>
</tbody>
</table>
PARALLEL IMPLEMENTATION

Calculate histogram

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

Except the first 2 threads, 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

<table>
<thead>
<tr>
<th>Step = 1</th>
<th>0</th>
<th>2</th>
<th>5</th>
<th>7</th>
<th>9</th>
<th>5</th>
<th>10</th>
<th>22</th>
<th>22</th>
<th>10</th>
<th>0</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Step = 2</td>
<td>0</td>
<td>2</td>
<td>5</td>
<td>9</td>
<td>14</td>
<td>12</td>
<td>19</td>
<td>27</td>
<td>32</td>
<td>32</td>
<td>22</td>
<td>10</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>Step = 4</td>
<td>0</td>
<td>2</td>
<td>5</td>
<td>9</td>
<td>14</td>
<td>14</td>
<td>24</td>
<td>36</td>
<td>46</td>
<td>44</td>
<td>41</td>
<td>37</td>
<td>33</td>
<td>34</td>
<td>26</td>
<td>15</td>
</tr>
<tr>
<td>Step = 8</td>
<td>0</td>
<td>2</td>
<td>5</td>
<td>9</td>
<td>14</td>
<td>14</td>
<td>24</td>
<td>36</td>
<td>46</td>
<td>46</td>
<td>46</td>
<td>46</td>
<td>46</td>
<td>47</td>
<td>48</td>
<td>50</td>
</tr>
</tbody>
</table>

```c
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

```c
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

Select pivots

Calculate histogram

Find the bin with kth element

Use prefix-sum to write out

Elements reduce to N?

\[
\text{Histogram: } 1 \ 2 \ 5 \ 9 \ 14 \ 14 \ 24 \ 36 \ 46 \ 46 \ 46 \ 46 \ 47 \ 48 \ 50 \ 51
\]

\[
K = 20
\]

\[
(h[t] - k) \cdot (h[t+1] - k) = (14 - 20) \cdot (46 - 20) > 0
\]

\[
(h[t] - k) \cdot (h[t+1] - k) = (14 - 20) \cdot (46 - 20) > 0
\]

\[
(h[t] - k) \cdot (h[t+1] - k) = (14 - 20) \cdot (46 - 20) > 0
\]
PARALLEL IMPLEMENTATION

Filter

Select pivots
Calcualte histogram
Find the bin with kth element

Filter

Elements reduce to N?

<table>
<thead>
<tr>
<th>data</th>
<th>40 30 21 9 41 42 53 56 58 70 15 13 12 7 8 16</th>
</tr>
</thead>
<tbody>
<tr>
<td>In the bin or not</td>
<td>0 0 0 1 0 0 0 0 0 0 0 0 0 1 1 0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>block_pos</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Update pos &amp; write to block cache</td>
<td>unsigned old_pos = atomicAdd(&amp;block_pos, (unsigned)1); buf[old_pos] = val;</td>
</tr>
<tr>
<td>block cache</td>
<td>9 7 8</td>
</tr>
<tr>
<td>global cache</td>
<td>out_pos = atomicAdd(&amp;(global_pos), block_pos);</td>
</tr>
</tbody>
</table>
| | if (tid < block_pos) {
| | out[out_pos + tid] = buf[tid];
| | }
| | out[out_pos + tid] = buf[tid];
| | 6 4 3 ... 9 7 8 ... |
Accelerate Radix Select with CUDA

- Radix select introduction
- Parallel implementation
- Implementation for different data size
IMPLEMENTATION FOR DIFFERENT DATA SIZE

Device-wide

- Select pivots
- Calculate histogram
- Find the bin with kth element
- Filter

```
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

Select pivots

Calculate histogram

Find the bin with \( k \)th element

Filter

Elements reduce to \( N \)?

---

```c
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(blockId.x==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:

<table>
<thead>
<tr>
<th>Experiment</th>
<th>Speedup Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>batch</td>
<td>len</td>
</tr>
<tr>
<td>1</td>
<td>1,000,000</td>
</tr>
<tr>
<td>1</td>
<td>10,000,000</td>
</tr>
<tr>
<td>1</td>
<td>100,000,000</td>
</tr>
<tr>
<td>16</td>
<td>1,000,000</td>
</tr>
<tr>
<td>16</td>
<td>10,000,000</td>
</tr>
<tr>
<td>128</td>
<td>1,000,000</td>
</tr>
<tr>
<td>128</td>
<td>10,000,000</td>
</tr>
</tbody>
</table>

https://github.com/anilshanbhag/gpu-topk
IMPLEMENTATION FOR DIFFERENT DATA SIZE

Block-wide

Select pivots
Calcualte histogram
Find the bin with kth element
Use prefix-sum to write out

Elements reduce to N?

#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