GPU Radix Sort

Yue Zhang

Exclusive Prefix Sum

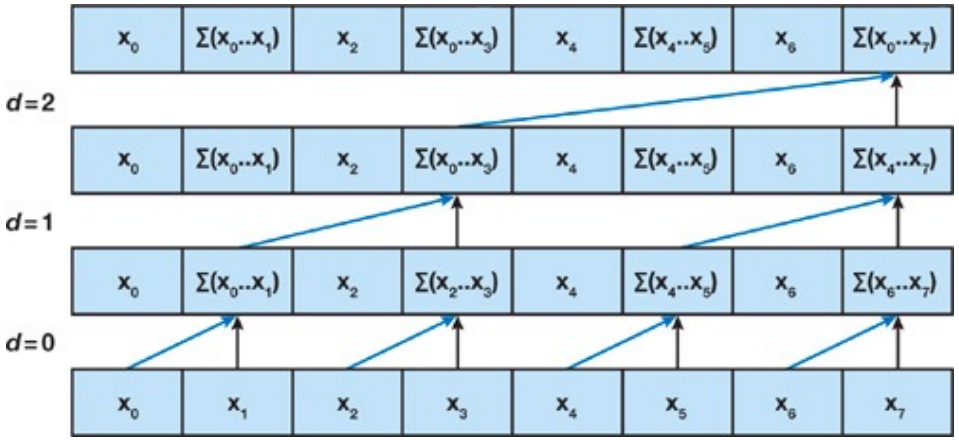
- Definition: The sum of elements before it(not include itself)
  - E.g. input 1,2,3,4 -> 0,1,2,3
- Use exclusive prefix sum in directly computing the offset in each round's output
  - E.g 1^{st} round digit: 21,11,28,15; how 15,11 know its offset in output?
  - 15's offset=1's queue length(2) + 15's position in 5's queue (0)=2
  - 11's offset=0 + 11's position in 1's queue (1)=1
  - Create a mask for digit 1, 1 for end with 1, 0 otherwise: 1, 1 ,0 ,0
  - Exclusive prefix sum of 1, 1 ,0 ,0 = 0,1,2,2
  - 1's queue length = last element in exclusive prefix sum(2) + 1

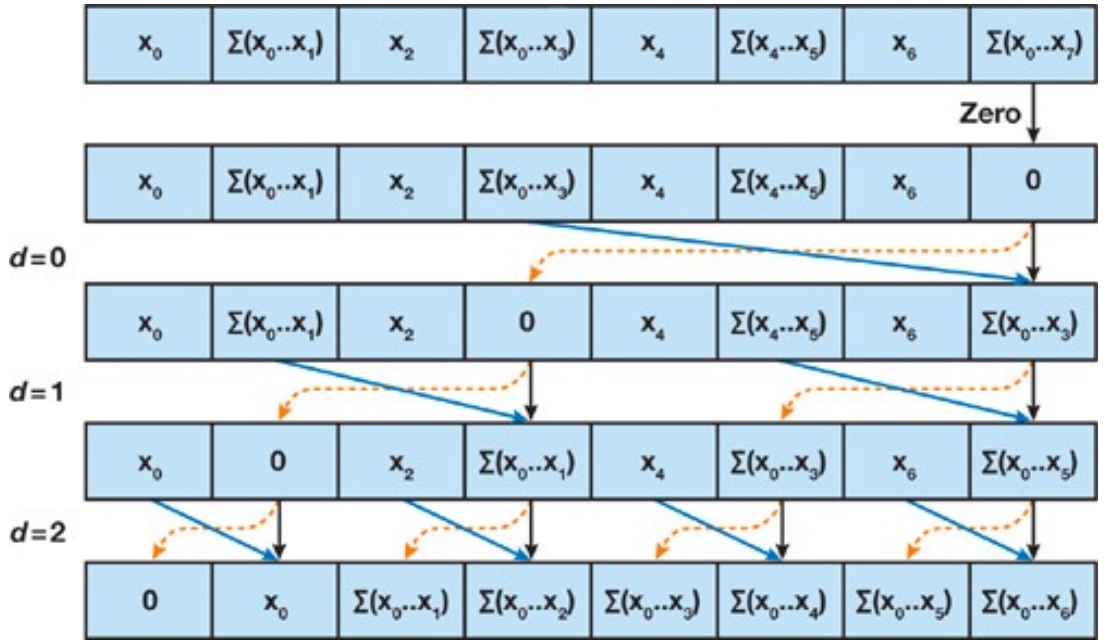
Blelloch algorithm

- Input: 1,2,3,4,5,6,7,8
- Expect: 0,1,3,6,10,15,21,28
- Adding: 3,7,11,15
- Prefix sum: 0, 3, 10, 21
- Replace even: 0, 2, 3, 4, 10, 6, 21, 8
- Add odd: 1, 3, 5, 7
- Result: 0, 1, 3, 6, 10, 15,21,28

- Idea: divide and conquer
- Subproblem: add two adjacent number, forming a new array which is half the size
- If we can solve the prefix sum of subproblem, we can solve the original problem by replacing and adding
- Recursive termination: when input is one element array return [0]

Blelloch algorithm

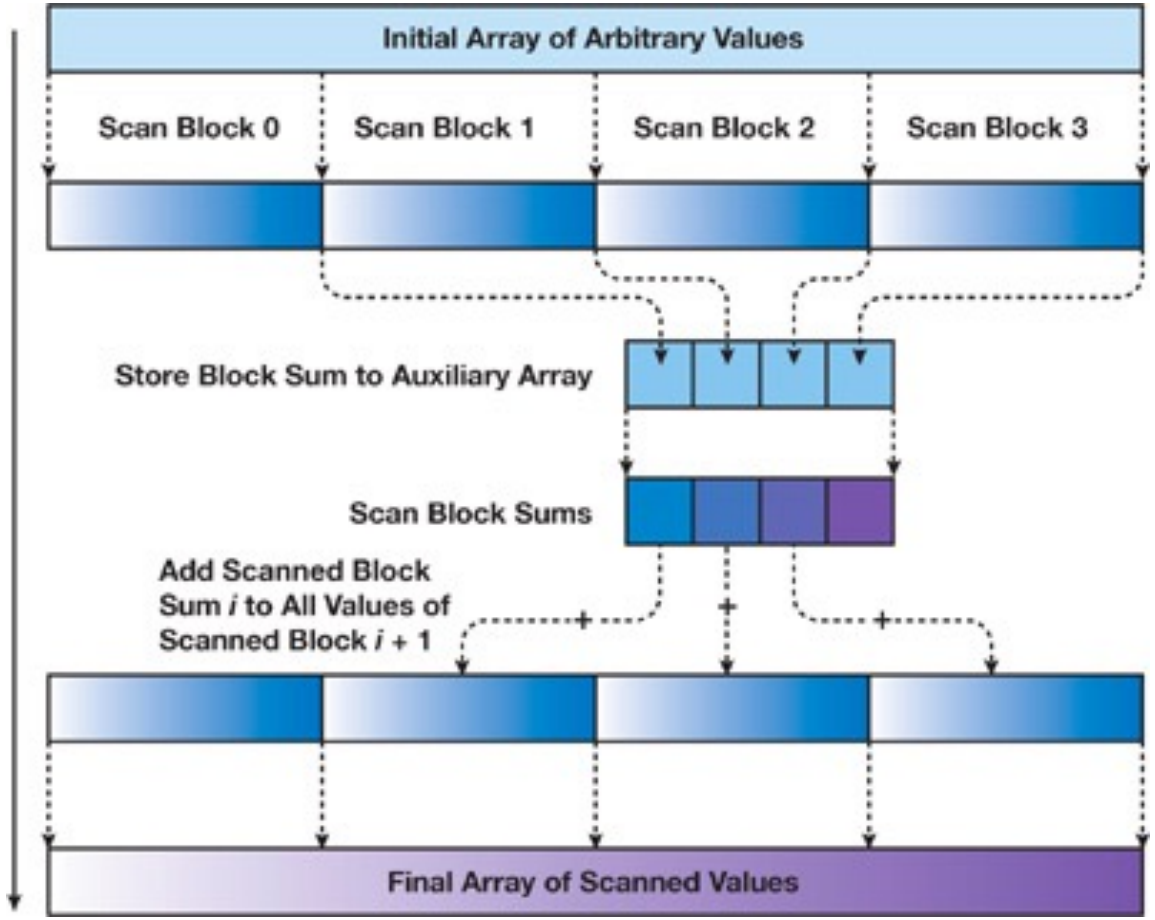





```
for (int d = n>>1; d > 0; d >>= 1 {
   __syncthreads();
   if (thid < d) {
      int ai = offset*(2*thid+1)-1;
      int bi = offset*(2*thid+2)-1;
      temp[bi] += temp[ai]; }
   offset *= 2;}
```

```
temp[n-1]=0
for (int d = 1; d < n; d *= 2) {
   offset >>= 1;
    __syncthreads();
   if (thid < d) {
      int ai = offset*(2*thid+1)-1;
      int bi = offset*(2*thid+2)-1;
      float t = temp[ai];
      temp[ai] = temp[bi];
      temp[bi] += t;}}
```

Inter-block Prefix Sum



- Blelloch algorithm require syncthread
- In grids level, we don’t have synchronization
- Local prefix sum + sum of block sums before it = global prefix sum
- Store each block’s total sum
- Add each intra-block prefix sum with one element in auxiliary array

Avoid bank conflict

- Blelloch algorithm has stride memory access
  - 1st 0, 2…16…30 thread 8 bank conflict with thread 0 => 2-degree bank conflict
  - 2nd 1, 5…17…33…49…61=> 4-degree bank conflict
  - 3rd 1, 9…17…33…49…65…81…97…113…121=> 8-degree bank conflict
- Padding 1 element every 16 element
  - 1st 0, 2…17…31
  - Bank 0,2,4,….1,3,15
  - 2nd 1, 5…18…35…52…64
  - Bank 1,5,….2…3…4…0
  - 3rd 1, 9…18…35…52…69…86…103…120…128
  - Bank 1,9,….2…3…4…5…6…7…8…0

global memory coalescing

- Input:32,23,56,37,89,41……
- Digit: 2,3,6,7,9,1
- Assume 0-9 counter are all 100
- Offset:200,300,600,700,900,100
- Very sparse global memory access
- Sort in-block data using in-block offset, write back to global memory
- Use prefix sum to get final offset, move data to final position
- When do the final move:
  - 10,20,30,40……maps to continuous global memory

Implementation

- Most of my idea comes from
  - https://developer.nvidia.com/gpugems/GPUGems3/gpugems3_ch39.html
  - https://github.com/mark-poscablo/gpu-radix-sort
    - Correctness checking code
- I use QueryPerformanceCounter to measure time
- Sorting 32M random integer(0–1024) the speed up of using GPU instead of CPU is 3-4 times