Tối ưu Radix sort trên GPU

Hoàng Minh Thanh (18424062)
Nguyễn Mạnh Tấn (18424060)

KHOA CÔNG NGHỆ THÔNG TIN
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

Nội dung trình bày

- Cài đặt radix sort tuần tự
- Song song tính hist
- Song song 2 bước hist và scan
- Thuật toán Radix Sort song song với k = 1
- Thuật toán Radix Sort tuần tự theo hướng dẫn
- Cải tiến Song song 2 bước tính hist và scan
- Cải tiến cài đặt preScatter và scatter song song
- Cải tiến cài Scatter song song
- Bản cuối cùng
- Profiling

Kết quả

TRÌNH TỰ

STT	Yêu cầu	Thời gian (ms)	File (*.cu)	Hoàn thành
1	Thuật toán Radix Sort tuần tự	1108.569	RadixSort-Base1	Tấn
2	Song song tính hist	3131.499	RadixSort-Base2.1	Tấn
3	Song song 2 bước hist và scan	2766.765	RadixSort-Base2.2.cu	Tấn
4	Thuật toán Radix Sort song song với k = 1	820.311	RadixSort-Base3.cu	Thanh
5	Thuật toán Radix Sort tuần tự theo hướng dẫn	6017.512	RadixSort-Base4.1.cu	Thanh
6	Cải tiến Song song 2 bước tính hist và scan	783.338	RadixSort-Base4.2	Thanh
7	Cải tiến cài đặt preScatter và scatter song song	345.774	RadixSort-Base4.3.cu	Thanh
8	Cải tiến cài Scatter song song	326.793	RadixSort-Base4.4	Thanh
9	Bản cuối cùng	137.674	RadixSort-Final.cu	Thanh
10	Cài đặt bằng thrust	64.244	Thrust	Tấn

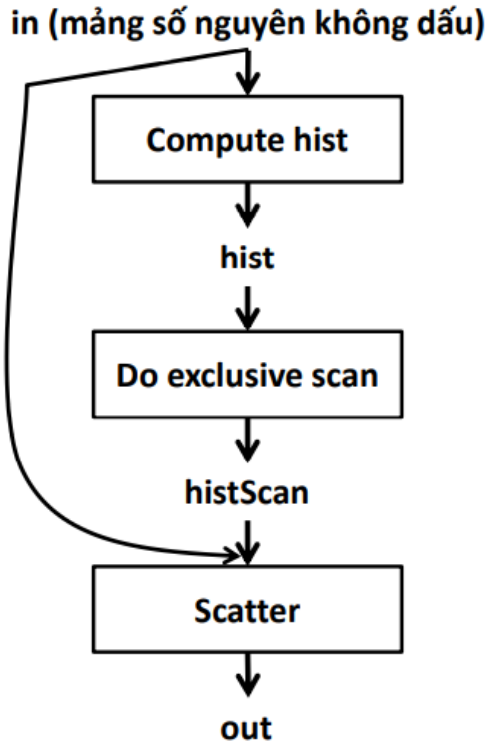
Đường dẫn colab

- Để dễ theo dõi : mong thầy vào link colab :

Public : https://drive.google.com/file/d/1WVOGq5s0kQCv0x9l81ifpJYFmQqOCiYL/view?usp=sharing

Radix sort tuần tự

- Cầu trúc chương trình radix sort tuần tự



Radix sort tuần tự

- Mã nguồn : Có sẵn

```
// (using STABLE counting sort)
for (int bit = 0; bit < sizeof(uint32_t) * 8; bit += nBits)
{
    // TODO: Compute histogram
    memset(hist, 0, nBins * sizeof(int));
    for (int i = 0; i < n; i++)
    {
        int bin = (src[i] >> bit) & (nBins - 1);
        hist[bin]++;
    }

    // TODO: Scan histogram (exclusively)
    histScan[0] = 0;
    for (int bin = 1; bin < nBins; bin++)
        histScan[bin] = histScan[bin - 1] + hist[bin - 1];

    // TODO: Scatter elements to correct locations
    for (int i = 0; i < n; i++)
    {
        int bin = (src[i] >> bit) & (nBins - 1);
        dst[histScan[bin]] = src[i];
        histScan[bin]++;
    }

    // Swap src and dst
    uint32_t * temp = src;
    src = dst;
    dst = temp;
}
```

Radix sort tuần tự

- Kết quả

```
**********GPU info**********
Name: Tesla P100-PCIE-16GB
Compute capability: 6.0
Num SMs: 56
Max num threads per SM: 2048
Max num warps per SM: 64
GMEM: 17071734784 byte
SMEM per SM: 65536 byte
SMEM per block: 49152 byte
*****************************

Input size: 16777217

Radix Sort by host
Time: 1214.182 ms
```

Song song 2 bước hist và scan

- Cài đặt histogram kernel

```
// Histogram kernel
__global__ void computeHistogram(uint32_t * in, int n, int * hist, int nBins, int bit)
{
    // Each block compute its local hist using atomic on SMEM
    extern __shared__ int s_bin[];
    int i = blockIdx.x * blockDim.x + threadIdx.x;
    int delta = (nBins - 1) / blockDim.x + 1;
    for (int i = 0; i < delta; i++){
        int id = threadIdx.x + i * blockDim.x;
        if (id < nBins){
            s_bin[id] = 0;
        }
    }
    __syncthreads();

    if (i < n){
        int bin = (in[i] >> bit) & (nBins - 1);
        atomicAdd(&s_bin[bin], 1);
    }
    __syncthreads();

    // Each block adds its local hist to global hist using atomic on GMEM
    for (int i = 0; i < delta; i++)
    {
        int id = threadIdx.x + i * blockDim.x;
        if (id < nBins){
            atomicAdd(&hist[id], s_bin[id]);
        }
    }
}
```

Thuật toán Radix Sort song song k=1

- Cài đặt scan kernel

```
// scan kernel
__global__ void scanExclusiveBlk(int * in, int n, int * out, int * blkSums)
{
    extern __shared__ int s_data[];
    int i = blockIdx.x * blockDim.x + threadIdx.x;
    if (i > 0 && i < n){
        s_data[threadIdx.x] = in[i - 1];
    }
    else{
        s_data[threadIdx.x] = 0;
    }
    __syncthreads();

    for (int stride = 1; stride < blockDim.x; stride *= 2)
    {
        int val = 0;
        if (threadIdx.x >= stride){
            val = s_data[threadIdx.x - stride];
        }
        __syncthreads();

        s_data[threadIdx.x] += val;
        __syncthreads();
    }

    if (i < n){
        out[i] = s_data[threadIdx.x];
    }
    if (threadIdx.x == 0 && blkSums != NULL){
        blkSums[blockIdx.x] = s_data[blockDim.x - 1];
    }
}
```

Radix sort với k = 1

- Cài đặt radix sort với k = 1

```
for (int bit = 0; bit < sizeof(uint32_t) * 8; bit += nBits)
{
    CHECK(cudaMemset(d_hist, 0, nBins * sizeof(int)));

    // Compute histogram
    computeHistogram<<<gridSize, blockSize, smemHistBytes>>>(d_src, n, d_hist, nBins, bit);
    cudaDeviceSynchronize();
    CHECK(cudaMemcpy(hist, d_hist, nBins * sizeof(int), cudaMemcpyDeviceToHost));

    // Scan exclusive only its block
    scanExclusiveBlk<<<gridSize, blockSize, smemScanBytes>>>(d_hist, nBins, d_histScan, d_blkSums);
    cudaDeviceSynchronize();
    CHECK(cudaMemcpy(blkSums, d_blkSums, gridSize.x * sizeof(int), cudaMemcpyDeviceToHost));

    // Sum scan result of its previous block
    for (int i = 1; i < gridSize.x; i++){
        blkSums[i] += blkSums[i-1];
    }
    CHECK(cudaMemcpy(d_blkSums, blkSums, gridSize.x * sizeof(int), cudaMemcpyHostToDevice));
    computeHistScan<<<gridSize, blockSize>>>(d_histScan, nBins, d_blkSums);
    cudaDeviceSynchronize();

    CHECK(cudaMemcpy(histScan, d_histScan, nBins * sizeof(int), cudaMemcpyDeviceToHost));

    // Scatter
    for (int i = 0; i < n; i++)
    {
        int bin = (src[i] >> bit) & (nBins - 1);
        dst[histScan[bin]] = src[i];
        histScan[bin]++;
    }
    uint32_t * temp = src;
    src = dst;
    dst = temp;
}
```

Cải tiến Song song 2 bước tính hist và scan

```
for (int bit = 0; bit < sizeof(uint32_t) * 8; bit += nBits)
{
    // compute historgram
    CHECK(cudaMemcpy(d_src, src, n * sizeof(uint32_t), cudaMemcpyHostToDevice));
    CHECK(cudaMemset(d_hist, 0, nBins * gridHistSize.x * sizeof(int)));
    computeHistogram<<<gridHistSize, blockSize, smemHistBytes>>>(d_src, n, d_hist, nBins, bit);
    cudaDeviceSynchronize();

    // compute scan
    scanExclusiveBlk<<<gridScanSize, blockSize, smemScanBytes>>>(d_hist, nBins * gridHistSize.x, d_scan, d_blkSums);
    cudaDeviceSynchronize();

    CHECK(cudaMemcpy(blkSums, d_blkSums, gridScanSize.x * sizeof(int), cudaMemcpyDeviceToHost));
    for (int i = 1; i < gridScanSize.x; i++){
        blkSums[i] += blkSums[i - 1];
    }
    CHECK(cudaMemcpy(d_blkSums, blkSums, gridScanSize.x * sizeof(int), cudaMemcpyHostToDevice));
    computeHistScan<<<gridScanSize, blockSize>>>(d_scan, nBins * gridHistSize.x, d_blkSums);
    cudaDeviceSynchronize();
    CHECK(cudaMemcpy(scan, d_scan, nBins * gridHistSize.x * sizeof(int), cudaMemcpyDeviceToHost));

    // Scatter
    for (int i = 0; i < n ; i++)
    {
        int bin = i / blockSize.x + ((src[i] >> bit) & (nBins - 1)) * gridHistSize.x;
        dst[scan[bin]] = src[i];
        scan[bin]++;
    }
    uint32_t * temp = src;
    src = dst;
    dst = temp;       You, a few seconds ago • Uncommitted changes
}
```

Cải tiến song song hist

- Bản đơn giản

```
// Histogram kernel
__global__ void computeHistogram(uint32_t * in, int n, int * hist, int nBins, int bit)
{
    // TODO
    int i = blockIdx.x * blockDim.x + threadIdx.x;
    if (i < n)
    {
        int bin = (in[i] >> bit) & (nBins - 1);
        atomicAdd(&hist[bin * gridDim.x + blockIdx.x], 1);
    }       You, 4 hours ago • update
}
```

Sử dụng SMEM trong tính hist

```cpp
// Histogram kernel
__global__ void computeHistogram(uint32_t * in, int n, int * hist, int nBins, int bit)
{
    // TODO
    // Each block computes its local hist using atomic on SMEM
    extern __shared__ int s_bin[];
    int i = blockIdx.x * blockDim.x + threadIdx.x;

    s_bin[threadIdx.x] = 0;
    __syncthreads();
    if (i < n)
    {
        int bin = (in[i] >> bit) & (nBins - 1);      You, 4 hours ago • update
        atomicAdd(&s_bin[bin], 1);
    }
    __syncthreads();
    if (threadIdx.x < nBins){
        hist[threadIdx.x * gridDim.x + blockIdx.x] += s_bin[threadIdx.x];
    }
}
```

Song song hóa scan

```cpp
// scan kernel
__global__ void scanExclusiveBlk(int * in, int n, int * out, int * blkSums)
{
    // TODO
    extern __shared__ int s_data[];
    int i = blockIdx.x * blockDim.x + threadIdx.x;
    if (i > 0 && i < n){
        s_data[threadIdx.x] = in[i - 1];
    }
    else{
        s_data[threadIdx.x] = 0;
    }
    __syncthreads();

    for (int stride = 1; stride < blockDim.x; stride *= 2)
    {
        int val = 0;
        if (threadIdx.x >= stride){
            val = s_data[threadIdx.x - stride];
        }
        __syncthreads();

        s_data[threadIdx.x] += val;
        __syncthreads();
    }
    if (i < n){
        out[i] = s_data[threadIdx.x];
    }
    if (blkSums != NULL){
        blkSums[blockIdx.x] = s_data[blockDim.x - 1];
    }       You, 4 hours ago • update
}
```

Song song hóa scan

```
__global__ void computeHistScan(int * in, int n, int* blkSums)
{
    int i = blockIdx.x * blockDim.x + threadIdx.x;
    if (i < n && blockIdx.x > 0)
        in[i] += blkSums[blockIdx.x - 1];
}
```

Cải tiến cài đặt preScatter và scatter song song

```
for (int bit = 0; bit < sizeof(uint32_t) * 8; bit += nBits)
{
    // Compute "hist" of the current digit
    CHECK(cudaMemset(d_scan, 0, nBins * gridHistSize.x * sizeof(int)));
    computeHistogram<<<gridHistSize, blockSize, smemBytes>>>(d_src, n, d_scan, nBins, bit);
    cudaDeviceSynchronize();

    // Scan
    scanExclusiveBlk<<<gridScanSize, blockSize, smemBytes>>>(d_scan, nBins * gridHistSize.x, d_scan, d_blkSums);
    cudaDeviceSynchronize();
    CHECK(cudaMemcpy(blkSums, d_blkSums, gridScanSize.x * sizeof(int), cudaMemcpyDeviceToHost));
    for (int i = 1; i < gridScanSize.x; i++){
        blkSums[i] += blkSums[i - 1];
    }
    CHECK(cudaMemcpy(d_blkSums, blkSums, gridScanSize.x * sizeof(int), cudaMemcpyHostToDevice));
    computeHistScan<<<gridScanSize, blockSize>>>(d_scan, nBins * gridHistSize.x, d_blkSums);
    cudaDeviceSynchronize();
    CHECK(cudaMemcpy(scan, d_scan, sizeof(int)* nBins * gridHistSize.x, cudaMemcpyDeviceToHost));

    // Scatter
    preScatter<<<gridScatterSize, blockSize, smemScatterBytes>>>(d_src, n, nBits, bit, nBins, d_preRank);
    cudaDeviceSynchronize();

    scatter<<<gridScatterSize, blockSize>>>(d_src, d_preRank, bit, d_scan, n, nBins, d_dst);      You, 3 hours ag
    cudaDeviceSynchronize();

    // Swap "src" and "dst"
    uint32_t * temp = d_src;
    d_src = d_dst;
    d_dst = temp;
}
// Copy "d_src" to "out"
```

Cải tiến cài Scatter song song

```
for (int bit = 0; bit < sizeof(uint32_t) * 8; bit += nBits)
{
    // Compute "hist" of the current digit
    CHECK(cudaMemset(d_scan, 0, nBins * gridHistSize.x * sizeof(int)));
    computeHistogram<<<gridHistSize, blockSize, smemBytes>>>(d_src, n, d_scan, nBins, bit);
    cudaDeviceSynchronize();
          You, 3 hours ago • update
    // Scan
    scanExclusiveBlk<<<gridScanSize, blockSize, smemBytes>>>(d_scan, nBins * gridHistSize.x, d_scan, d_blkSums);
    cudaDeviceSynchronize();
    CHECK(cudaMemcpy(blkSums, d_blkSums, gridScanSize.x * sizeof(int), cudaMemcpyDeviceToHost));
    for (int i = 1; i < gridScanSize.x; i++){
        blkSums[i] += blkSums[i - 1];
    }
    CHECK(cudaMemcpy(d_blkSums, blkSums, gridScanSize.x * sizeof(int), cudaMemcpyHostToDevice));
    computeHistScan<<<gridScanSize, blockSize>>>(d_scan, nBins * gridHistSize.x, d_blkSums);
    cudaDeviceSynchronize();
    CHECK(cudaMemcpy(scan, d_scan, sizeof(int)* nBins * gridHistSize.x, cudaMemcpyDeviceToHost));

    // Scatter
    scatter<<<gridScatterSize, blockSize, smemScatterBytes>>>(d_src, n, nBits, bit, nBins, d_scan, d_dst);
    cudaDeviceSynchronize();

    // Swap "src" and "dst"
    uint32_t * temp = d_src;
    d_src = d_dst;
    d_dst = temp;
}
```

Bản clean code

```
Input size: 16777217

Radix Sort by host
Time: 1110.266 ms

Baseline Radix Sort (highlight)
Time: 6040.603 ms
CORRECT :)

Radix Sort by device
Time: 326.793 ms
CORRECT :)
```

Kết quả tốt nhất

```
**********GPU info**********
Name: Tesla P100-PCIE-16GB
Compute capability: 6.0
Num SMs: 56
Max num threads per SM: 2048
Max num warps per SM: 64
GMEM: 17071734784 byte
SMEM per SM: 65536 byte
SMEM per block: 49152 byte
****************************

Input size: 16777217
Block size : 512

Radix Sort by host
Time: 1157.686 ms

Baseline Radix Sort (highlight)
Time: 6223.896 ms
CORRECT :)

Radix Sort by device
Time: 107.973 ms
CORRECT :)

Radix Sort with thrust
Time: 61.450 ms
CORRECT :)
```

Kết quả chạy với các blockSize

- 128

```
1 !./run_radix_final 128
```

```
**********GPU info**********
Name: Tesla P100-PCIE-16GB
Compute capability: 6.0
Num SMs: 56
Max num threads per SM: 2048
Max num warps per SM: 64
GMEM: 17071734784 byte
SMEM per SM: 65536 byte
SMEM per block: 49152 byte
*****************************

Input size: 16777217
Block size : 128

Radix Sort by host
Time: 1187.524 ms

Baseline Radix Sort (highlight)
Time: 6604.675 ms
CORRECT :)

Radix Sort by device
Time: 171.408 ms
CORRECT :)

Radix Sort with thrust
Time: 62.409 ms
CORRECT :)
```

Kết quả chạy với các blockSize

- 256

```
1 !./run_radix_final 256
```

```
**********GPU info**********
Name: Tesla P100-PCIE-16GB
Compute capability: 6.0
Num SMs: 56
Max num threads per SM: 2048
Max num warps per SM: 64
GMEM: 17071734784 byte
SMEM per SM: 65536 byte
SMEM per block: 49152 byte
**************************

Input size: 16777217
Block size : 256

Radix Sort by host
Time: 1179.641 ms

Baseline Radix Sort (highlight)
Time: 6525.257 ms
CORRECT :)

Radix Sort by device
Time: 164.896 ms
CORRECT :)

Radix Sort with thrust
Time: 64.943 ms
CORRECT :)
```

Kết quả chạy với các blockSize

- 512

```
1 !./run_radix_final 512
```

```
**********GPU info**********
Name: Tesla P100-PCIE-16GB
Compute capability: 6.0
Num SMs: 56
Max num threads per SM: 2048
Max num warps per SM: 64
GMEM: 17071734784 byte
SMEM per SM: 65536 byte
SMEM per block: 49152 byte
**************************

Input size: 16777217
Block size : 512

Radix Sort by host
Time: 1187.559 ms

Baseline Radix Sort (highlight)
Time: 6496.068 ms
CORRECT :)

Radix Sort by device
Time: 176.900 ms
CORRECT :)

Radix Sort with thrust
Time: 67.799 ms
CORRECT :)
```

Kết quả chạy với các blockSize

- 1024

```
1 !./run_radix_final 1024
```

```
**********GPU info**********
Name: Tesla P100-PCIE-16GB
Compute capability: 6.0
Num SMs: 56
Max num threads per SM: 2048
Max num warps per SM: 64
GMEM: 17071734784 byte
SMEM per SM: 65536 byte
SMEM per block: 49152 byte
****************************

Input size: 16777217
Block size : 1024

Radix Sort by host
Time: 1188.351 ms

Baseline Radix Sort (highlight)
Time: 6543.773 ms
CORRECT :)

Radix Sort by device
Time: 171.959 ms
CORRECT :)

Radix Sort with thrust
Time: 61.078 ms
CORRECT :)
```

Profiling

```
ime: 63.225 ms
ORRECT :)
=230== Profiling application: ./build
=230== Profiling result:
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
GPU activities:   65.51%  87.110ms         6  14.518ms  6.2720us  43.572ms  [CUDA memcpy DtoH]
                  22.25%  29.592ms         6  4.9320ms  8.5110us  14.789ms  [CUDA memcpy HtoD]
                   7.77%  10.335ms         4  2.5838ms  2.5509ms  2.6823ms  scatter(unsigned int*, int, int, int, int, int*, unsigned int*)
                   1.69%  2.2434ms         4  560.84us  535.21us  571.88us  computeHistogram(unsigned int*, int, int*, int, int)
                   0.54%  719.43us         3  239.81us  163.90us  278.33us  void thrust::cuda_cub::cub::DeviceRadixSortDownsweepKernel<thrus
                   0.54%  714.05us         4  178.51us  178.23us  178.78us  scanExclusiveBlk(int*, int, int*, int*)
                   0.50%  659.72us         4  164.93us  164.06us  166.14us  void thrust::cuda_cub::cub::DeviceRadixSortDownsweepKernel<thrus
                   0.34%  457.49us         7  65.355us  64.605us  66.429us  void thrust::cuda_cub::cub::RadixSortScanBinsKernel<thrust::cuda_
                   0.27%  360.31us         4  90.076us  89.597us  90.524us  void thrust::cuda_cub::cub::DeviceRadixSortUpsweepKernel<thrust:
                   0.26%  347.41us         4  86.853us  86.653us  87.037us  computeHistScan(int*, int, int*)
                   0.20%  264.21us         3  88.071us  86.045us  91.197us  void thrust::cuda_cub::cub::DeviceRadixSortUpsweepKernel<thrust:
                   0.13%  172.03us         1  172.03us  172.03us  172.03us  void thrust::cuda_cub::core::_kernel_agent<thrust::cuda_cub::__p
                   0.00%  6.2080us         4  1.5520us  1.5040us  1.6320us  [CUDA memset]
     API calls:   79.21%  546.72ms         8  68.340ms  1.0420us  546.68ms  cudaEventCreate
                   8.62%  59.525ms        10  5.9525ms  18.187us  44.315ms  cudaMemcpy
                   8.58%  59.214ms         2  29.607ms  15.004ms  44.209ms  cudaMemcpyAsync
                   2.37%  16.352ms        17  961.89us  92.984us  2.6881ms  cudaDeviceSynchronize
                   0.72%  4.9781ms         6  829.68us  180.11us  2.1380ms  cudaFree
                   0.22%  1.5394ms         6  256.57us  92.357us  371.84us  cudaMalloc
                   0.06%  441.10us         1  441.10us  441.10us  441.10us  cuDeviceTotalMem
                   0.05%  358.27us        38  9.4280us  5.3060us  46.566us  cudaLaunchKernel
                   0.04%  296.87us        97  3.0600us     143ns  147.36us  cuDeviceGetAttribute
                   0.04%  268.02us         1  268.02us  268.02us  268.02us  cudaGetDeviceProperties
                   0.02%  135.08us         8  16.884us  3.9290us  35.728us  cudaEventRecord
                   0.01%  95.837us         4  23.959us  12.452us  55.663us  cudaMemset
                   0.01%  89.830us         2  44.915us  6.2910us  83.539us  cudaStreamSynchronize
                   0.01%  78.773us         8  9.8460us  7.9720us  12.138us  cudaEventSynchronize
                   0.01%  40.372us         1  40.372us  40.372us  40.372us  cuDeviceGetName
```

Profiling

Time	Calls	Avg	Min	Max	Name
65.51%	87.110ms	6	14.518ms	6.2720us	43.572ms	[CUDA memcpy DtoH]
22.25%	29.592ms	6	4.9320ms	8.5110us	14.789ms	[CUDA memcpy HtoD]
7.77%	10.335ms	4	2.5838ms	2.5509ms	2.6823ms	scatter(unsigned int*, int, int, int, int, int*, unsigr
1.69%	2.2434ms	4	560.84us	535.21us	571.88us	computeHistogram(unsigned int*, int, int*, int, int)
0.54%	719.43us	3	239.81us	163.90us	278.33us	void thrust::cuda_cub::cub::DeviceRadixSortDownsweepKer
0.54%	714.05us	4	178.51us	178.23us	178.78us	scanExclusiveBlk(int*, int, int*, int*)
0.50%	659.72us	4	164.93us	164.06us	166.14us	void thrust::cuda_cub::cub::DeviceRadixSortDownsweepKer
0.34%	457.49us	7	65.355us	64.605us	66.429us	void thrust::cuda_cub::cub::RadixSortScanBinsKernel<thr
0.27%	360.31us	4	90.076us	89.597us	90.524us	void thrust::cuda_cub::cub::DeviceRadixSortUpsweepKerne
0.26%	347.41us	4	86.853us	86.653us	87.037us	computeHistScan(int*, int, int*)
0.20%	264.21us	3	88.071us	86.045us	91.197us	void thrust::cuda_cub::cub::DeviceRadixSortUpsweepKerne
0.13%	172.03us	1	172.03us	172.03us	172.03us	void thrust::cuda_cub::core::_kernel_agent<thrust::cuda
0.00%	6.2080us	4	1.5520us	1.5040us	1.6320us	[CUDA memset]

Nhận xét

- Có thể thấy tốc độ của hàm scatter là chậm nhất.
- Sau đó đến hàm tính histogram