Mutual Information In CUDA

Yan Ge

Entropy

$$H(A) = -\int p_A(a) \log p_A(a)\, da$$

A	**Random Variable**
$p_A(a)$	**Probability of event a**
H(A)	**Entropy**

Image Entropy

$$H(A) = -\int p_A(a) \log p_A(a)\, da$$

A	**Image A**
$p_A(a)$	**The Probability of Image Intensity a appeared in A i.e. the frequency of one pixel value a in image A**
H(A)	**Image Entropy**

Image Entropy comes from Information theory, Shannon Entropy.

Image Entropy measures the information of an image including spatial location of pixels.

Image Entropy does NOT tell you anything about spatial arrangement of pixels.

Image Entropy tells you Statistical Distribution of the image intensities.

The probability of picking one pixel associated with intensity i, is p_i

How much information do you gain when you pick any pixel and that pixel happens to be i?

$$-\log_2(p_i)$$

You could have picked any pixel,

$$H = -\sum_i p_i \cdot \log_2(p_i)$$

The probability of picking one pixel associated with intensity i, is p_i

How much information do you gain when you pick any pixel and that pixel happens to be i?

$$-\log_2(p_i)$$

You could have picked any pixel,

$$H = -\sum_i p_i \cdot \log_2(p_i)$$

How to calculate Image Entropy?

$$H = -\sum_{i} p_i \cdot \log_2 (p_i)$$

Simply build a histogram.
P is the frequency of the i-th bin of that histogram.

Mutual Information

Mutual Information of two discrete random variables X and Y can be defined as:

$$I(X;Y) = \sum_{y \in Y} \sum_{x \in X} p(x,y) \log \left(\frac{p(x,y)}{p(x)\,p(y)} \right)$$

P(x) and P(y) are the marginal probability distribution functions of X and Y respectively. P(x, y) is the joint probability distribution function of X and Y.

Mutual Information In terms of Entropy

$$I(A,B) = H(A) + H(B) - H(A,B)$$

H(A): Entropy of Image A
H(B): Entropy of Image B
H(A,B): Joint Entropy of Image A and B

How to calculate Mutual Information?

Method 1: Entropy

$$I(A,B) = H(A) + H(B) - H(A,B)$$

Calculate 3 terms:

1. Entropy of image A
2. Entropy of image B
3. Joint Entropy of Image A and B

Weakness of Method 1

- To find Entropy, one have to build histogram first.
- Building histogram is exhausting using CPU
  - For loop iteration over all pixels of image
  - Not just one histogram, but THREE! THREE! THREE!
- What about GPU?
  - We will see shortly.

How to calculate Mutual Information?

Method 2: Probability Density

$$I(X;Y) = \sum_{y \in Y} \sum_{x \in X} p(x,y) \log \left(\frac{p(x,y)}{p(x)\,p(y)} \right)$$

Calculate 3 terms:

1. Marginal probability distribution of x
2. Marginal probability distribution of y
3. Joint probability distribution of x and y

Mattes Mutual Information
Parzen Windowing

$$p(a) \approx P^*(a) = \frac{1}{N} \sum_{s_j \in S} K(a - s_j)$$

Learning resource on Parzen Window, from Prof. Olga Veksler
http://www.csd.uwo.ca/~olga/Courses/CS434a_541a/Lecture6.pdf

D. Mattes, D.R. Haynor, H. Vesselle, T. Lewellen, and W. Eubank. "Non-rigid multimodality image registration." (Proceedings paper).Medical Imaging 2001: Image Processing. SPIE Publications, 3 July 2001. pp. 1609–1620.

CUDA Histogram

From Victor Podlozhnyuk, “Histogram Calculation in CUDA”

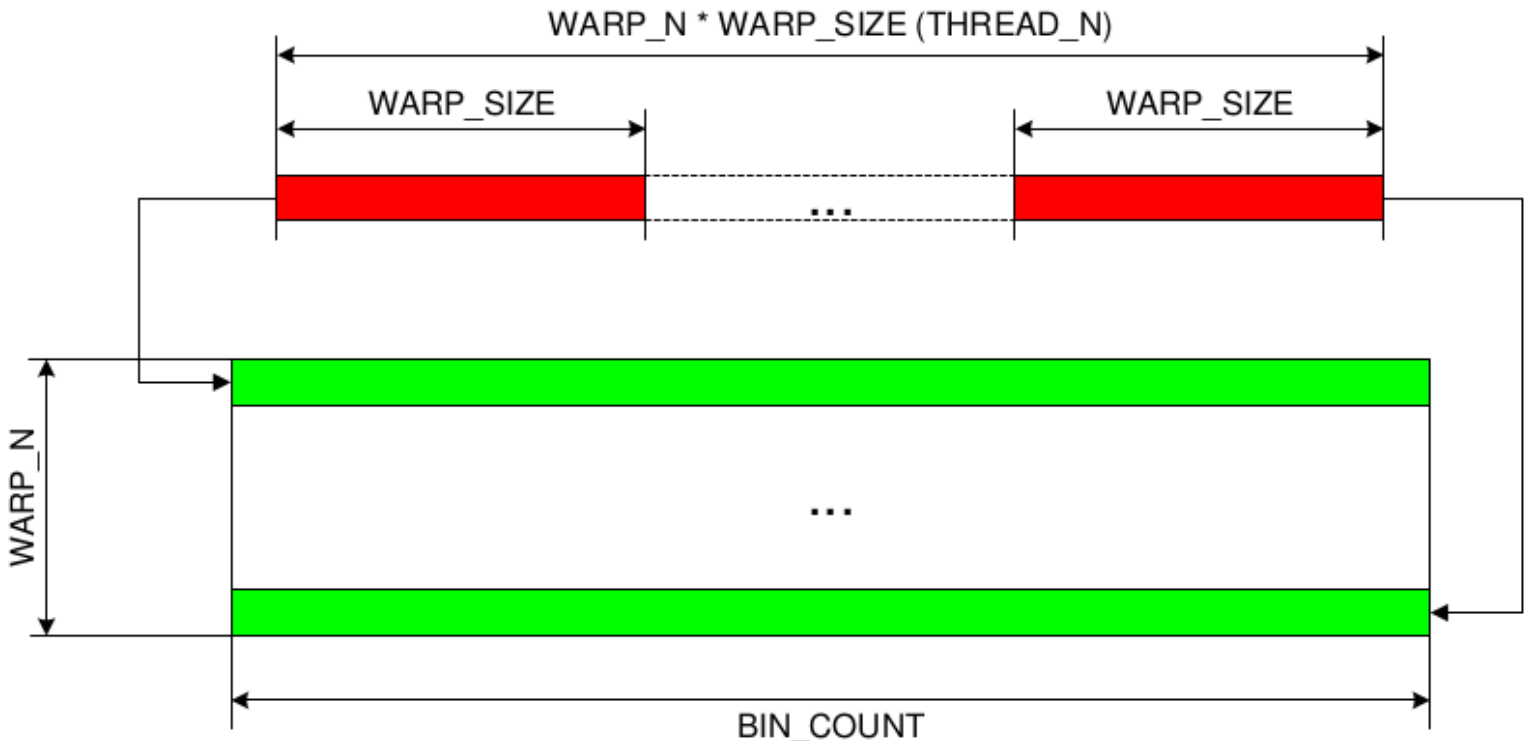

Figure 3. `s_Hist[]` layout for histogram256.

How to avoid bank conflict

The shared memory bank number is equal to (threadPos + data * THREAD_N / 4) % 16.

If we just set threadPos equal to threadIdx.x, all threads within a half-warp will access its own byte “lane”, but these lanes will map to only 4 banks, thus introducing 4-way bank conflicts.

So we shuffle [5:0] bit rangs of threadIdx.x to make all threads within each warp to access the same byte within 4-byte words, stored in different banks, thus completely avoiding bank conflict.

Avoiding bank conflict

```
const uint threadPos =
    ((threadIdx.x & ~(SHARED_MEMORY_BANKS * 4 - 1)) << 0) |
    ((threadIdx.x & (SHARED_MEMORY_BANKS     - 1)) << 2) |
    ((threadIdx.x & (SHARED_MEMORY_BANKS * 3)) >> 4);
```

```
threadIdx: 0threadPos: 0
threadIdx: 1threadPos: 4
threadIdx: 2threadPos: 8
threadIdx: 3threadPos: 12
threadIdx: 4threadPos: 16
threadIdx: 5threadPos: 20
threadIdx: 6threadPos: 24
threadIdx: 7threadPos: 28
threadIdx: 8threadPos: 32
threadIdx: 9threadPos: 36
threadIdx: 10threadPos: 40
threadIdx: 11threadPos: 44
threadIdx: 12threadPos: 48
threadIdx: 13threadPos: 52
threadIdx: 14threadPos: 56
threadIdx: 15threadPos: 60
threadIdx: 16threadPos: 1
threadIdx: 17threadPos: 5
threadIdx: 18threadPos: 9
threadIdx: 19threadPos: 13
threadIdx: 20threadPos: 17
threadIdx: 21threadPos: 21
threadIdx: 22threadPos: 25
threadIdx: 23threadPos: 29
threadIdx: 24threadPos: 33
threadIdx: 25threadPos: 37
threadIdx: 26threadPos: 41
threadIdx: 27threadPos: 45
threadIdx: 28threadPos: 49
threadIdx: 29threadPos: 53
threadIdx: 30threadPos: 57
threadIdx: 31threadPos: 61
```

How much shared memory to allocate?

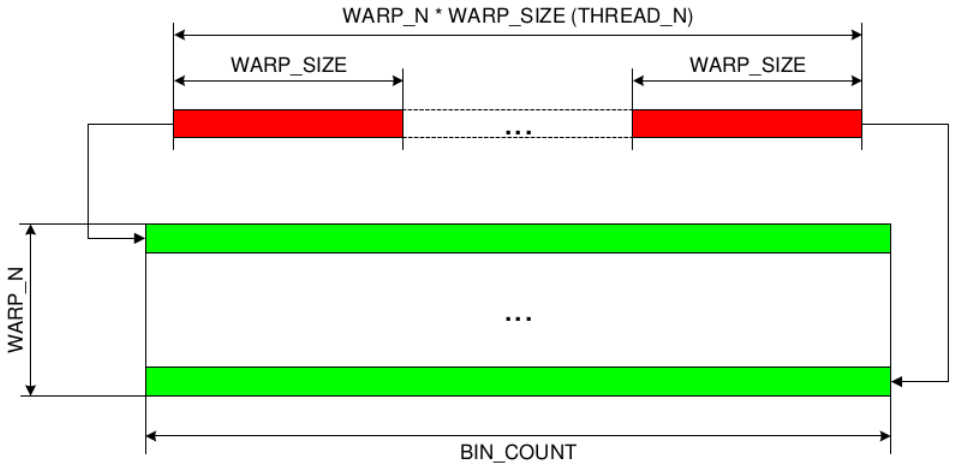

Figure 3. `s_Hist[]` layout for histogram256.

```
// <<<240 PARTIAL_HISTOGRAM64_COUNT   ,  warp_count x warp_size>>>
__global__ void histogram64Kernel(uint *d_PartialJointHistograms,
                                  uint *d_PartialHistograms1,
                                  uint *d_PartialHistograms2,
                                  uint *d_Data1,
                                  uint *d_Data2,
                                  uint dataCount)
{
  __shared__ uint s_Hist1[HISTOGRAM64_THREADBLOCK_MEMORY]; //  warp_count * 64
  __shared__ uint s_Hist2[HISTOGRAM64_THREADBLOCK_MEMORY]; //  warp_count * 64
  __shared__ uint s_JointHist[JOINT_HISTOGRAM64_THREADBLOCK_MEMORY]; // 64 * 64
```

```
/deviceQuery Starting...

 CUDA Device Query (Runtime API) version (CUDART static linking)

)etected 1 CUDA Capable device(s)

)evice 0: "Quadro M1000M"
  CUDA Driver Version / Runtime Version          8.0 / 8.0
  CUDA Capability Major/Minor version number:    5.0
  Total amount of global memory:                 4042 MBytes (4238540800 bytes)
  ( 4) Multiprocessors, (128) CUDA Cores/MP:     512 CUDA Cores
  GPU Max Clock rate:                            1072 MHz (1.07 GHz)
  Memory Clock rate:                             2505 Mhz
  Memory Bus Width:                              128-bit
  L2 Cache Size:                                 2097152 bytes
  Maximum Texture Dimension Size (x,y,z)         1D=(65536), 2D=(65536, 65536),
  Maximum Layered 1D Texture Size, (num) layers  1D=(16384), 2048 layers
  Maximum Layered 2D Texture Size, (num) layers  2D=(16384, 16384), 2048 layers
  Total amount of constant memory:               65536 bytes
  Total amount of shared memory per block:       49152 bytes
  Total number of registers available per block: 65536
  Warp size:                                     32
  Maximum number of threads per multiprocessor:  2048
  Maximum number of threads per block:           1024
  Max dimension size of a thread block (x,y,z): (1024, 1024, 64)
  Max dimension size of a grid size    (x,y,z): (2147483647, 65535, 65535)
  Maximum memory pitch:                          2147483647 bytes
```

What is per-warp histogram and partial histogram in the code?

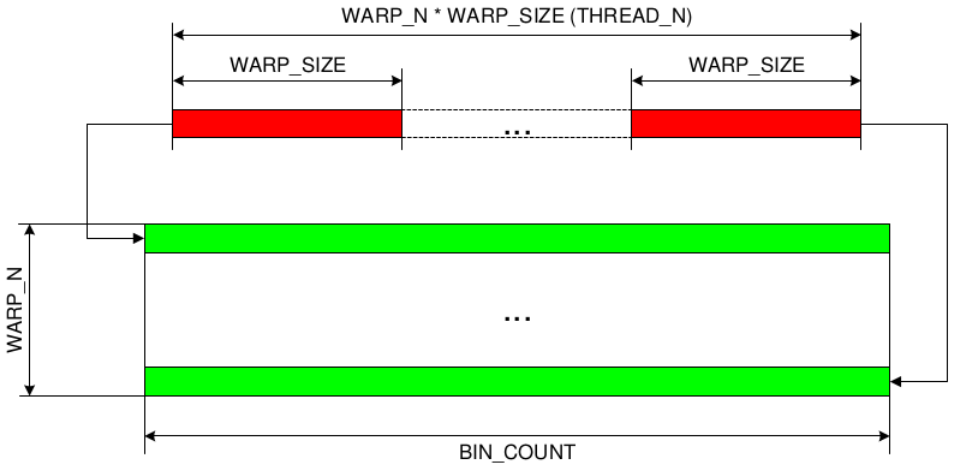

Figure 3. `s_Hist[]` layout for histogram256.

Each WARP corresponds to one per-warp histogram.
Each thread block has WARP_N per-warp histograms.
When all threads within one block complete their job, reduce per-block histograms into one histogram.
That one histogram is then mapped into partial histograms. (it is a very long array depending on how many blocks you allocated.)
Finally, we reduce this partial histograms into one single histogram.

How to calculate Joint histogram?

In a similar fasion like this, but a little different.

```
inline __device__ void addByte(uint *s_JointHist, uint *s_WarpHist1, uint *s_WarpHist2, uint data1, uint data2)
{
  uint d1 = data1 / 4;
  uint d2 = data2 / 4;
  atomicAdd(s_WarpHist1 + d1, 1); // divide 4 for 64-bin histogram
  atomicAdd(s_WarpHist2 + d2, 1);
  atomicAdd(s_JointHist + d1 * HISTOGRAM64_BIN_COUNT + d2, 1);
}
```

```
// store shared Joint histogram into global.
for (uint i = 0; i < (JOINT_HISTOGRAM64_THREADBLOCK_MEMORY / HISTOGRAM64_THREADBLOCK_SIZE); i++)
  {
    d_PartialJointHistograms[blockIdx.x * JOINT_HISTOGRAM64_BIN_COUNT + threadIdx.x + i * HISTOGRAM64_THREADBLOCK_SIZE]
      = s_JointHist[threadIdx.x + i * HISTOGRAM64_THREADBLOCK_SIZE];
  }
```

```
#define JOINT_HISTOGRAM64_THREADBLOCK_MEMORY (HISTOGRAM64_BIN_COUNT * HISTOGRAM64_BIN_COUNT)
```

With all the histograms, we can calculate entropy, and finally, mutual information.

```
__global__ void
entropy_kernel(unsigned int *g_idata, double *g_odata, unsigned int n)
{
  double *sdata = SharedMemory<double>();

    // load shared mem
    unsigned int tid = threadIdx.x;
    unsigned int i = blockIdx.x*blockDim.x + threadIdx.x;


    if(g_idata[i] && i < n){
      double prob = (double)g_idata[i]/n;
      sdata[tid] = -prob * std::log(prob) / LOG_2;
    }else
      sdata[tid] = 0.0;

    __syncthreads();

    // do reduction in shared mem
    for (unsigned int s=1; s < blockDim.x; s *= 2)
    {
        // modulo arithmetic is slow!
        if ((tid % (2*s)) == 0)
        {
            sdata[tid] += sdata[tid + s];
        }

        __syncthreads();
    }

    // write result for this block to global mem
    if (tid == 0){
      g_odata[blockIdx.x] = sdata[0];
      printf("g_odata[%d] = %f\n", blockIdx.x, g_odata[blockIdx.x]);
    }
}
```

My project is far from finishing!!!

- We are still counting all the pixels within two image. Mattes Mutual Information uses Parzen windowing and a kernel function (e.g. cubic BSpline interpolation) to estimate PDF without knowing all the pixels, only sample of them. And it gives accurate, reliable result in practice.
- This is a memory bounded CUDA program, i.e. It does not have much of computation on each thread, beside the log. We should put more work in each thread. Anyone give me some clue?
- My bin count is fixed to product of 32. Why? Because of warp size is 32. What if user like to have 50 bins, not a multiple of 32, what am I gonna do? All threads blocks, memory allocation, are trash code!
- Imaging scientists don’t know CUDA, how to wrap my code into a library so their life can be easier. I am thinking CUDA Thrust! A C++ templated library.

You can throw questions at me now
If I cannot answer it, I promise you I will find solution to your question.

Thank you all and have a wonderful SUMMER!