CSED490C Lab Assignment 05

20232241 Dookyung Kang

Kernel code

```
__global__ void histogram(unsigned int* buffer, unsigned int* bin, int size) {
  int stride = blockDim.x * gridDim.x;
  int i = blockIdx.x * blockDim.x + threadIdx.x;

  __shared__ unsigned int pr_bin[NUM_BINS];
  for (int j = i; j < NUM_BINS; j += stride)
    bin[j] = 0;
  for (int u = threadIdx.x; u < NUM_BINS; u += blockDim.x)
    pr_bin[u] = 0;
  __syncthreads();
  for (int j = i; j < size; j += stride)
    ① atomicAdd(&(pr_bin[buffer[j]]), 1);
  __syncthreads();
  for (int u = threadIdx.x; u < NUM_BINS; u += blockDim.x)
    ② atomicAdd(&bin[u], pr_bin[u]);
  __syncthreads();
}

__global__ void histogramSaturate(unsigned int* bin) {
  int i = blockIdx.x * blockDim.x + threadIdx.x;
  bin[i] = min(bin[i], 127);
}
```

Host code

```
//@@ Allocate GPU memory here
  cudaMalloc((void**)&deviceInput, inputLength * sizeof(unsigned int));
  cudaMalloc((void**)&deviceBins, NUM_BINS * sizeof(unsigned int));
  CUDA_CHECK(cudaDeviceSynchronize());

//@@ Copy memory to the GPU here
  cudaMemcpy(deviceInput, hostInput, inputLength * sizeof(unsigned int), cudaMemcpyHostToDevice);
  cudaMemcpy(deviceBins, hostBins, NUM_BINS * sizeof(unsigned int), cudaMemcpyHostToDevice);
  CUDA_CHECK(cudaDeviceSynchronize());

//@@ Launch kernel
  int blockSize = 256;
  int maxGridSize = 8192;
  int gridSize = min(maxGridSize, (inputLength + blockSize - 1) / blockSize);
  histogram<<<gridSize, blockSize>>>(deviceInput, deviceBins, inputLength);
  int satGridSize = NUM_BINS / blockSize;
  histogramSaturate<<<satGridSize, blockSize>>>(deviceBins);

//@@ Copy the GPU memory back to the CPU here
  cudaMemcpy(hostBins, deviceBins, NUM_BINS * sizeof(unsigned int), cudaMemcpyDeviceToHost);
  CUDA_CHECK(cudaDeviceSynchronize());

//@@ Free the GPU memory here
  cudaFree(deviceBins);
  cudaFree(deviceInput);
```

Problem 1

① atomicAdd is called exactly inputLength(N) time throughout kernels.

② atomicAdd is called NUM_BINS = 4096 times per thread block.

In this case, B = # of thread block is initialized into $\min\left(B_{\max}, \lceil N/T \rceil\right)$ in host code,

where T(blockSize) = 256 and Bmax(maxBlockSize) = 8192 is modifiable.

Therefore, Atomic operations are called $\boldsymbol{N + 4096 \times \min\left(B_{\max}, \lceil N/T \rceil\right)}$ times.

Problem 2

The contention of ② atomicAdd is constant regardless of data.

However, contention of ① atomicAdd will be maximized when every element has the same value.

For each thread block, addition to same shared memory is completely serialized.

If $N \le B_{max} \times T$, the loop of ① called only once, so T of them are serialized.

Otherwise, $B = B_{max}$, then maximum $T \times \lceil N/B_{max}T \rceil$ of ① atomicAdd are serialized.

Problem 3

The more random the data is, especially in 256 chunks, the more different the input values are, the less content is. In this case, ① atomicAdd is closer to full parallel.

Problem 4

Refer to the previous page.

Problem 5

Refer to the below table. Each time is millisecond unit.

N	16	1024	513	511	1	500000	800000	1200000
GPU alloc	0.136262	0.141	0.145751	0.147649	0.14361	0.161956	0.394259	0.332044
H->D	0.044791	0.045716	0.043647	0.043303	0.041688	0.313888	0.449736	0.57497
Kernel	0.024034	0.025683	0.023922	0.025047	0.02356	0.076281	0.11287	0.148933
D->H	0.02734	0.025967	0.029283	0.024645	0.022598	0.019647	0.025409	0.028175
GPU free	0.122255	0.119247	0.118937	0.121361	0.119485	0.14266	0.265325	0.249115