Histogram data access pattern – Accelerating C++ with CUDA advanced class.

07.12.2020 Introduction to CUDA and OpenCL

Task:

The purpose of this lab is to implement an efficient **histogramming** algorithm for an input array of integers within a given range (in principle can be arbitrary large, depending on the memory size in our machine). Each **integer will map into a single bin**, so the values will range from 0 to (MAX_BINS - 1). The histogram bins will use unsigned integer counters that must **be saturated at 256** (no flipping back to zero!). The input length can be assumed to be less than $N = 2^{32}$. MAX_BINS is fixed at 4096 for this task.

Hints:

This algorithm can be implemented as a cascade with two kernels: one that does a histogram without saturation, and a final kernel that cleans up the bins if they are too large, that also give us some flexibility regarding to change in saturation value if necessary. These two stages can also be combined into a single kernel. Use a standard data flow structure as learned during classes!
For speed up purposes we can use **privatisation approach** (each thread should keep its own histogram). Also, use only **atomic operations** to update the histogram entries.

Must be:

In the final solution, we should have in the code:

- A function that checks and handles error queries to the GPU
- Processing grid optimisation taking into account the input data vector size, the amount of shared memory available per SM/processing block
- A wrapper function that handles the kernel launch with appropriate arguments allowing the processing grid to be worked out

When creating the report please include the following answers

- What should be considered the most important part for the speed up of the algorithm?
- What is essential step when using shared memory?
- Describe the memory movement (access and writes into the global memory, take into account both kernels!)

General template:

1. Necessary includes
2. Define the max bin constant
3. Define the error checking/handling function
4. Define the histogram wrapper function
5. Define the main function able to read line arguments for the data size
6. Create single threaded application for histograming

Structure of the main function:

1. Read the input data size
2. Define the data structures using malloc low level functions
3. Allocate necessary memory on host and device
4. Generate data set (for this exercise do not initialise the data on the device!)
5. Create timer object (as you learned from the matrix multiplication algorithm)
6. Transfer the data
7. Runn histogram wrapper
8. Fetch the data back form the GPU
9. Compare the output with the single threaded application

Comment regarding the shared memory

Since the input data size and histogram size are both arguments it is best to use **extern** specifier when declaring the shared memory. Please analyse this example:

```
#include <stdio.h>

__global__ void staticReverse(int *d, int n)
{
  __shared__ int s[64];
  int t = threadIdx.x;
  int tr = n-t-1;
  s[t] = d[t];
  __syncthreads();
  d[t] = s[tr];
}

__global__ void dynamicReverse(int *d, int n)
{
  extern __shared__ int s[];
  int t = threadIdx.x;
  int tr = n-t-1;
  s[t] = d[t];
  __syncthreads();
  d[t] = s[tr];
}
```

```
int main(void)
{
  const int n = 64;
  int a[n], r[n], d[n];

  for (int i = 0; i < n; i++) {
    a[i] = i;
    r[i] = n-i-1;
    d[i] = 0;
  }

  int *d_d;
  cudaMalloc(&d_d, n * sizeof(int));

  // run version with static shared memory
  cudaMemcpy(d_d, a, n*sizeof(int), cudaMemcpyHostToDevice);
  staticReverse<<<1,n>>>(d_d, n);
  cudaMemcpy(d, d_d, n*sizeof(int), cudaMemcpyDeviceToHost);
  for (int i = 0; i < n; i++)
    if (d[i] != r[i]) printf("Error: d[%d]!=r[%d] (%d, %d)\n", i, i, d[i], r[i]);

  // run dynamic shared memory version
  cudaMemcpy(d_d, a, n*sizeof(int), cudaMemcpyHostToDevice);
  dynamicReverse<<<1,n,n*sizeof(int)>>>(d_d, n);
  cudaMemcpy(d, d_d, n * sizeof(int), cudaMemcpyDeviceToHost);
  for (int i = 0; i < n; i++)
    if (d[i] != r[i]) printf("Error: d[%d]!=r[%d] (%d, %d)\n", i, i, d[i], r[i]);
}
```