Histogram

1. Short program description.

Histogram kernel is used to calculate how many instances of every number(between 0-4095) there is in our input array. Each block uses shared memory to store number of times each number has been encountered by threads in this block. The kernel uses atomic operations to update this local histograms for each block and, after all threads are synchronized, it uses atomic operations to update globabl histogram. The second kernel saturates the histogram at 256.

2. The most important part for speeding up the algorithm is data privatisation approach for each block of threads. Each block uses shared memory to store its local histogram. Shared memory is on-chip memory which makes it much faster to access than global memory, in this case especially, where we need to update the histogram regularly.

3. There are a few essential steps to using shared memory. The most important one is thread synchronization. In order to avoid so called „race conditions", we need to make sure that no thread tries to read the data from shared memory before other threads write to it first. An example of this is initializing our local histograms. After loop which initializes local histogram to '0' we use __syncthreads(), which prevents threads from proceeding with execution until all other threads reach this __syncthreads().

4. Memory movement in histogram kernel:

Each thread first initializes local histogram to '0'. So there is MAX_BINS writes to shared memory. After synchronizing, we loop trough a number of elements (dependent on data_size and grid configuration) from global input data, reading them and writing to shared memory (which is performer by atomic operations).Then we synchronize threads again. The last step is to update global histogram, which requires writing each element from local histogram to global histogram (also usign atomic operations).

Memory movement in saturation kernel:

In saturation kernel we use just the global memory, because each element of out data (histogram array) is read once and (optionally) written to once.

Michał Orlewski