

OpenCL Benchmark Progress Report (2016.11.14)

1. The image format is incorrect, according to the table below, CL_INTENSITY cannot be used with CL_UNSIGNED_INT8, but it still works on target or Nvidia, not on Intel, so the performance may be undefined:

Format	Description
CL_R, or CL_A	
CL_INTENSITY	This format can only be used if channel data type = CL_UNORM_INT8, CL_UNORM_INT16, CL_SNORM_INT8, CL_SNORM_INT16, CL_HALF_FLOAT, or CL_FLOAT.
CL_LUMINANCE	This format can only be used if channel data type = CL_UNORM_INT8, CL_UNORM_INT16, CL_SNORM_INT8, CL_SNORM_INT16, CL_HALF_FLOAT, or CL_FLOAT.
CL_RG, or CL_RA	
CL_RGB	This format can only be used if channel data type = CL_UNORM_SHORT_565, CL_UNORM_SHORT_555 or CL_UNORM_INT101010.
CL_RGBA	
CL_ARGB, CL_BGRA	This format can only be used if channel data type = CL_UNORM_INT8, CL_SNORM_INT8, CL_SIGNED_INT8 or CL_UNSIGNED_INT8.

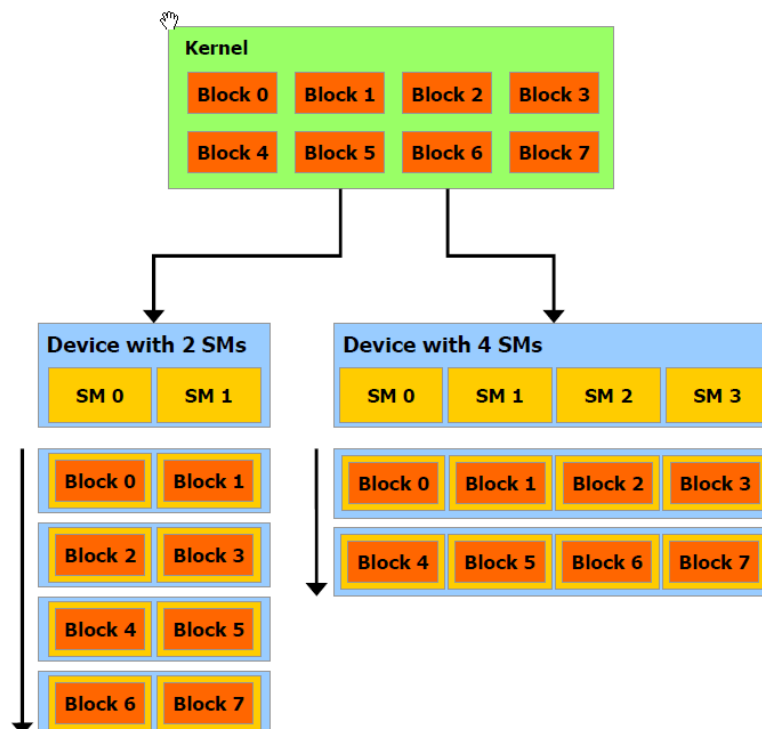
`cl_image_format image_format;`

`image_format.image_channel_data_type = CL_UNSIGNED_INT8;`

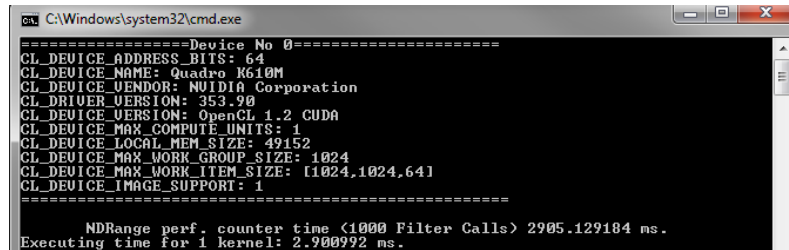
`image_format.image_channel_order = CL_INTENSITY;`

However, For the CL_UNORM_INT8 type, you will use the `read_imagef` and `write_imagef` functions to access to the image. These functions will return (or accept) a normalised floating point value, in the range 0.0f - 1.0f. Some devices (e.g. most GPUs) have hardware support for normalising texture values, so the conversion between integer and normalised floating point values will be very efficient.

2. For every Compute Unit on GPU, there is a limit number of workgroups can be launched simultaneously(eg. Max 8 for one CU in NVIDIA), which means the less workgroups the better performance, but the total number of CUs should be the factor of the total number of workgroups in order to avoid resource waste. And the requirement of private memory and local memory also decide how many workgroups can be launched at once.

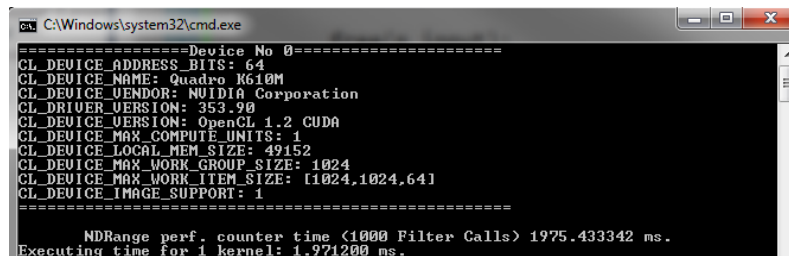


3. int vs float, the reason for using float data form just to speed up the calculation process(float point operation) and may also to avoid type conversion when calculate the $I_g/16$. However, the time cost of these is negligible, while the input data from uchar(8bit) to float(32bit) will increase the burden of fetching data from the global memory.



```
C:\Windows\system32\cmd.exe
=====Device No 0=====
CL_DEVICE_ADDRESS_BITS: 64
CL_DEVICE_NAME: Quadro K610M
CL_DEVICE_VENDOR: NVIDIA Corporation
CL_DRIVER_VERSION: 353.90
CL_DEVICE_VERSION: OpenCL 1.2 CUDA
CL_DEVICE_MAX_COMPUTE_UNITS: 1
CL_DEVICE_LOCAL_MEM_SIZE: 49152
CL_DEVICE_MAX_WORK_GROUP_SIZE: 1024
CL_DEVICE_MAX_WORK_ITEM_SIZE: [1024,1024,64]
CL_DEVICE_IMAGE_SUPPORT: 1
=====
NDRange perf. counter time <1000 Filter Calls> 2905.129184 ms.
Executing time for 1 kernel: 2.900992 ms.
```

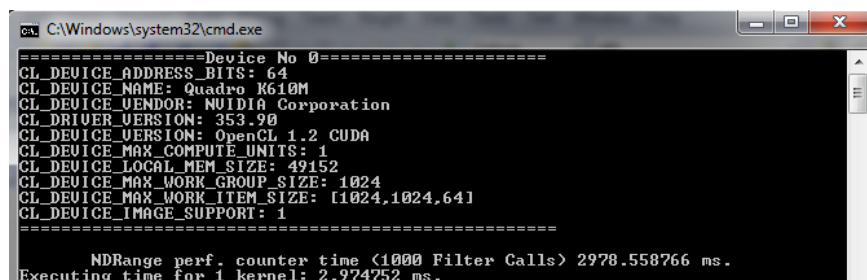
Fig1. Result for float(32bit) input using buffer



```
C:\Windows\system32\cmd.exe
=====Device No 0=====
CL_DEVICE_ADDRESS_BITS: 64
CL_DEVICE_NAME: Quadro K610M
CL_DEVICE_VENDOR: NVIDIA Corporation
CL_DRIVER_VERSION: 353.90
CL_DEVICE_VERSION: OpenCL 1.2 CUDA
CL_DEVICE_MAX_COMPUTE_UNITS: 1
CL_DEVICE_LOCAL_MEM_SIZE: 49152
CL_DEVICE_MAX_WORK_GROUP_SIZE: 1024
CL_DEVICE_MAX_WORK_ITEM_SIZE: [1024,1024,64]
CL_DEVICE_IMAGE_SUPPORT: 1
=====
NDRange perf. counter time <1000 Filter Calls> 1975.433342 ms.
Executing time for 1 kernel: 1.971200 ms.
```

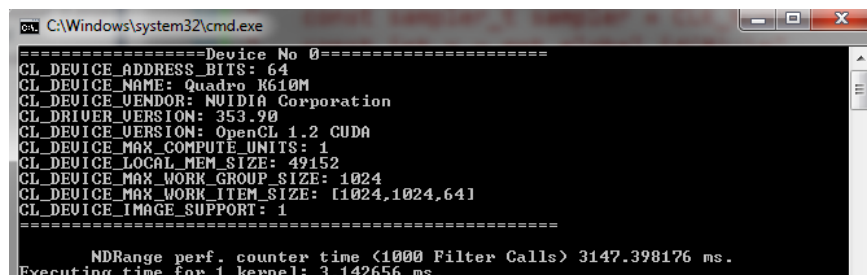
Fig2. Result for unsigned char(8bit) input using buffer

4. The advantage of using image method is the texture lookup and cache, which means one fetch from global memory will also the values of the neighbor area. However, the speed up may be not that obvious, and it quite depends on the hardware implementation.



```
C:\Windows\system32\cmd.exe
=====Device No 0=====
CL_DEVICE_ADDRESS_BITS: 64
CL_DEVICE_NAME: Quadro K610M
CL_DEVICE_VENDOR: NVIDIA Corporation
CL_DRIVER_VERSION: 353.90
CL_DEVICE_VERSION: OpenCL 1.2 CUDA
CL_DEVICE_MAX_COMPUTE_UNITS: 1
CL_DEVICE_LOCAL_MEM_SIZE: 49152
CL_DEVICE_MAX_WORK_GROUP_SIZE: 1024
CL_DEVICE_MAX_WORK_ITEM_SIZE: [1024,1024,64]
CL_DEVICE_IMAGE_SUPPORT: 1
=====
NDRange perf. counter time <1000 Filter Calls> 2978.558766 ms.
Executing time for 1 kernel: 2.974752 ms.
```

Fig3. Results when reach=1, image method.



```
C:\Windows\system32\cmd.exe
=====Device No 0=====
CL_DEVICE_ADDRESS_BITS: 64
CL_DEVICE_NAME: Quadro K610M
CL_DEVICE_VENDOR: NVIDIA Corporation
CL_DRIVER_VERSION: 353.90
CL_DEVICE_VERSION: OpenCL 1.2 CUDA
CL_DEVICE_MAX_COMPUTE_UNITS: 1
CL_DEVICE_LOCAL_MEM_SIZE: 49152
CL_DEVICE_MAX_WORK_GROUP_SIZE: 1024
CL_DEVICE_MAX_WORK_ITEM_SIZE: [1024,1024,64]
CL_DEVICE_IMAGE_SUPPORT: 1
=====
NDRange perf. counter time <1000 Filter Calls> 3147.398176 ms.
Executing time for 1 kernel: 3.142656 ms.
```

Fig4. Results when reach=10, image method.

And 10 is obviously out of cache range for texture cache.

5. In the buffer method for the edge pixels, the index of `pSrc[iPrev - reach]` will be negative and the results are undefined. While in image method, it just use `CLK_ADDRESS_CLAMP`, and return (0.0f, 0.0f, 0.0f, 0.0f) when coordinate out of range.

For next step:

1. Figure out the implementation of `read_image` and `write_image` function, the way "`read_imagef()`" is converted into code may not be as simple or as cheap as we would all like.
2. Try to compare one-channel image and 4-channel image.
3. Hypothesis: image object will store every channel value in 32 bit float form.