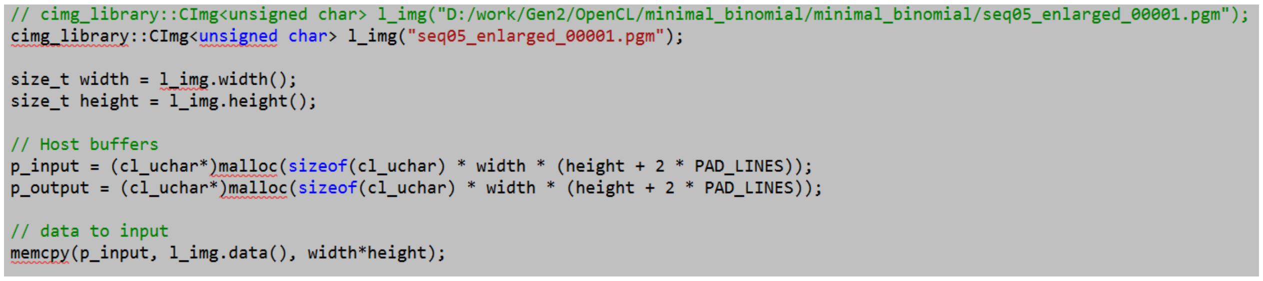
OpenCL Benchmark Analysis (2016.08.11)

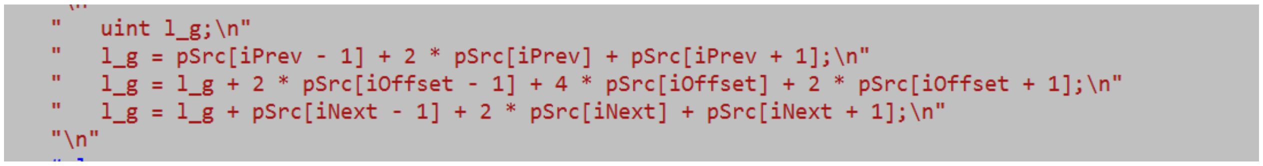
Brief: Buffer process needs to deal with width*height elements while Image process has to deal with width*height*4 elements.

Detail: when loading image file using CImg Library, the p_input is filled with 1920*1080 unsigned char values.



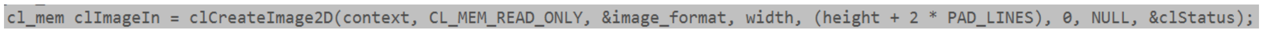
```
// cimg_library::CImg<unsigned char> l_img("D:/work/Gen2/OpenCL/minimal_binomial/minimal_binomial/seq05_enlarged_00001.pgm");
cimg_library::CImg<unsigned char> l_img("seq05_enlarged_00001.pgm");

size_t width = l_img.width();
size_t height = l_img.height();

// Host buffers
p_input = (cl_uchar*)malloc(sizeof(cl_uchar) * width * (height + 2 * PAD_LINES));
p_output = (cl_uchar*)malloc(sizeof(cl_uchar) * width * (height + 2 * PAD_LINES));

// data to input
memcpy(p_input, l_img.data(), width*height);
```

When using Buffer method, p_input will be copied to cl_input_buffer(has the same size), and the kernel set up (width*height) work_items to run the kernel.

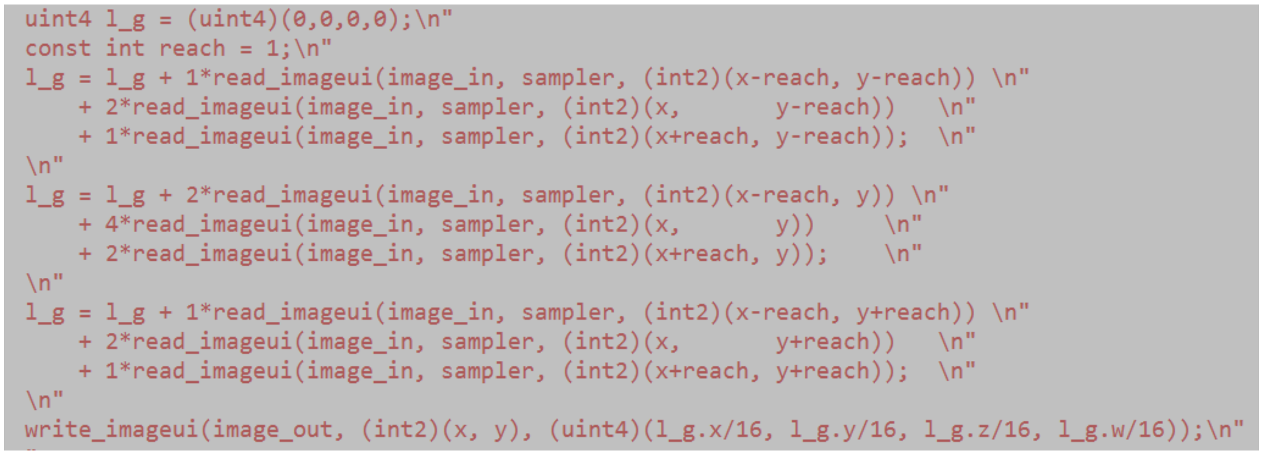


```
"   uint l_g;\n"
"   l_g = pSrc[iPrev - 1] + 2 * pSrc[iPrev] + pSrc[iPrev + 1];\n"
"   l_g = l_g + 2 * pSrc[iOffset - 1] + 4 * pSrc[iOffset] + 2 * pSrc[iOffset + 1];\n"
"   l_g = l_g + pSrc[iNext - 1] + 2 * pSrc[iNext] + pSrc[iNext + 1];\n"
"\n"
```

However, when using Image method, there are 4 values to describe one pixel, thus, every work_item has 4 times workload comparing to Buffer method.



```
cl_mem clImageIn = clCreateImage2D(context, CL_MEM_READ_ONLY, &image_format, width, (height + 2 * PAD_LINES), 0, NULL, &clStatus);
```



```
uint4 l_g = (uint4)(0,0,0,0);\n"
const int reach = 1;\n"
l_g = l_g + 1*read_imageui(image_in, sampler, (int2)(x-reach, y-reach)) \n"
    + 2*read_imageui(image_in, sampler, (int2)(x,       y-reach))   \n"
    + 1*read_imageui(image_in, sampler, (int2)(x+reach, y-reach));  \n"
\n"
l_g = l_g + 2*read_imageui(image_in, sampler, (int2)(x-reach, y)) \n"
    + 4*read_imageui(image_in, sampler, (int2)(x,       y))     \n"
    + 2*read_imageui(image_in, sampler, (int2)(x+reach, y));    \n"
\n"
l_g = l_g + 1*read_imageui(image_in, sampler, (int2)(x-reach, y+reach)) \n"
    + 2*read_imageui(image_in, sampler, (int2)(x,       y+reach))   \n"
    + 1*read_imageui(image_in, sampler, (int2)(x+reach, y+reach));  \n"
\n"
write_imageui(image_out, (int2)(x, y), (uint4)(l_g.x/16, l_g.y/16, l_g.z/16, l_g.w/16));\n"
```

Question: I just wonder that how to divide a 8-bit uchar pixel value into 4 channels? Kind of like 2bits for each channel?

Suggestions: The advantages of Image method show in two way, firstly, Pixel 2D coordinate do not have to convert into linear index in buffer; secondly, if original image pixel format is not float type, Image method do not need the conversion. (In our code we did not consider the second difference in comparing)

However, the implement of "read_image()" function is different on hardwares, and largely related to memory fetch mechanism.(Refering to website: https://forums.khronos.org/showthread.php/8657-Buffer-Vs-Image)