Name

CL_QCOM_ION_HOST_PTR

Name Strings

cl_qcom_ion_host_ptr

Contact

ssusheel at quicinc dot com
dgarcia at qti dot qualcomm dot com

Contributors

Balaji Calidas, QUALCOMM
David Garcia, QUALCOMM
Sushmita Susheelendra, QUALCOMM Innovation Center Inc.

Status

SHIPPING

Number

OpenCL Extension #22

Dependencies

OpenCL 1.1 is required. cl_qcom_ext_host_ptr is required. Android OS is required.

This extension is written against the OpenCL 1.1 specification

Overview

This extension extends the functionality provided by
clCreateBuffer, clCreateImage2D, clCreateImage3D.
It allows applications to pass an Android ION memory allocation to these functions
so that it can be mapped to the device's address space and thus avoid having to
copy data back and forth between the host and the device.

```
IP Status

    No known IP claims.

New Tokens

    Accepted by the <host_ptr> argument of clCreateBuffer, clCreateImage2D and clCreateImage3D:

        typedef struct _cl_mem_ion_host_ptr
        {
            // Type of external memory allocation.
            // Must be CL_MEM_ION_HOST_PTR_QCOM for ION allocations.
            cl_mem_ext_host_ptr   ext_host_ptr;

            // ION file descriptor
            int                   ion_filedesc;

            // Host pointer to the ION allocated memory
            void*                 ion_hostptr;

        } cl_mem_ion_host_ptr;

    Used together with CL_MEM_EXT_HOST_PTR_QCOM:

        CL_MEM_ION_HOST_PTR_QCOM                   0x40A8

Additions to Chapter 5.2.1 of the OpenCL 1.1 Specification
(Creating Buffer Objects)

        When CL_MEM_EXT_HOST_PTR_QCOM is enabled in the <flags> argument, then <host_ptr> is interpreted as a pointer to cl_mem_ext_host_ptr.
        When <host_ptr>->allocation_type is equal to CL_MEM_ION_HOST_PTR_QCOM then <host_ptr> can also be interpreted as
        a pointer to cl_mem_ion_host_ptr.

        In addition to that, the application must also initialize the following struct fields:
        * <host_ptr>->host_cache_policy must be equal to CL_MEM_HOST_WRITEBACK_QCOM if the ion allocation was made with the flag ION_FLAG_CACHED enabled.
        It must be equal to CL_MEM_HOST_UNCACHED_QCOM otherwise.
        * <host_ptr>->ion_filedesc must be the file descriptor of the ION memory allocation that the application
          wants to use as storage bits for the memory object.
        * <host_ptr>->ion_hostptr must be the host virtual pointer associated with the same ION memory allocation.

        Memory specified this way must be aligned to the device's page size. The application can query the device's
        page size by using clGetDeviceInfo(..., CL_DEVICE_PAGE_SIZE_QCOM, ...).
```

    Once the memory object is created, the application must call clEnqueueMapBuffer/clEnqueueMapImage with
    appropriate flags before reading or writing to it on the host. The host unmaps the region when accesses
    (reads and/or writes) to this mapped region by the host are complete. As per the OpenCL 1.2 specification,
    clEnqueueMapBuffer and clEnqueueMapImage act as synchronization points for the region of the buffer
    object being mapped.

Issues

Sample Code

```
    /*    Using the extension for CL buffer objects   */

    cl_mem              buffer_object           = NULL;
    size_t              buffer_size_in_bytes     = 0;
    size_t              buffer_size_with_padding = 0;
    cl_mem_ion_host_ptr  myionmem               = {0};
    size_t              ext_mem_padding_in_bytes = 0;
    size_t              device_page_size        = 0;

    // Query the device's page size and the amount of padding necessary at the end of the buffer.
    clGetDeviceInfo(device, CL_DEVICE_PAGE_SIZE_QCOM, sizeof(device_page_size), &device_page_size, NULL);
    clGetDeviceInfo(device, CL_DEVICE_EXT_MEM_PADDING_IN_BYTES_QCOM, sizeof(ext_mem_padding_in_bytes), &ext_mem_padding_in_bytes, NULL);

    // Compute the desired size for the data in the buffer.
    buffer_size_in_bytes = foobar();

    // Compute amount of memory that needs to be allocated for the buffer including padding.
    buffer_size_with_padding = buffer_size_in_bytes + ext_mem_padding_in_bytes;

    // Make an ION memory allocation of size "buffer_size_with_padding" here.
    // Notice that allocating "buffer_size_in_bytes" instead would be a mistake! It's important to allocate the extra padding.
    // Let's say the parameters of the allocation are stored in a struct named "ion_info" that we will use below.
    // ...

    // Create an OpenCL buffer object that uses "ion_info" as its data store. Notice how the buffer is created with size "buffer_size_in_bytes", not "buffer_size_with_padding".
    myionmem.ext_host_ptr.allocation_type    = CL_MEM_ION_HOST_PTR_QCOM;
    myionmem.ext_host_ptr.host_cache_policy  = CL_MEM_HOST_UNCACHED_QCOM;
    myionmem.ion_filedesc                  = ion_info_fd.file_descriptor;   // the file descriptor for ION
    myionmem.ion_hostptr                   = ion_info.host_virtual_address; // the hostptr returned by ION which is device page size aligned

    if(myionmem.ion_hostptr % device_page_size)
    {
        error("Host pointer must be aligned to device_page_size!");
    }

    buffer_object = clCreateBuffer(context, CL_MEM_USE_HOST_PTR | CL_MEM_EXT_HOST_PTR_QCOM, buffer_size_in_bytes, &myionmem, &errcode);


    /*    Using the extension for CL image objects   */

    cl_mem              image_object            = NULL;
    cl_mem_ion_host_ptr  myionmem               = {0};
    size_t              ext_mem_padding_in_bytes = 0;
    size_t              device_page_size        = 0;
    size_t              row_pitch               = 0;

    // Query the device's page size and the amount of padding necessary at the end of the buffer.
    clGetDeviceInfo(device, CL_DEVICE_PAGE_SIZE_QCOM, sizeof(device_page_size), &device_page_size, NULL);
    clGetDeviceInfo(device, CL_DEVICE_EXT_MEM_PADDING_IN_BYTES_QCOM, sizeof(ext_mem_padding_in_bytes), &ext_mem_padding_in_bytes, NULL);

    // Query the device supported row and slice pitch using clGetDeviceImageInfoQCOM
    // imgw - image width
    // imgh - image height
    // img_fmt - image format
    clGetDeviceImageInfoQCOM(device, imgw, imgh, &img_fmt, CL_IMAGE_ROW_PITCH, sizeof(image_row_pitch), &image_row_pitch, NULL);

    // Use the image height, row pitch obtained above and element size to compute the size of the buffer
    buffer_size_in_bytes = imgh * image_row_pitch;

    // Compute amount of memory that needs to be allocated for the buffer including padding.
    buffer_size_with_padding = buffer_size_in_bytes + ext_mem_padding_in_bytes;

    // Make an ION memory allocation of size "buffer_size_with_padding" here.
    // Notice that allocating "buffer_size_in_bytes" instead would be a mistake! It's important to allocate the extra padding.
    // Let's say the parameters of the allocation are stored in a struct named "ion_info" that we will use below.
    // ...

    // Create an OpenCL image object that uses "ion_info" as its data store.
    myionmem.ext_host_ptr.allocation_type    = CL_MEM_ION_HOST_PTR_QCOM;
    myionmem.ext_host_ptr.host_cache_policy  = CL_MEM_HOST_UNCACHED_QCOM;
    myionmem.ion_filedesc                    = ion_info_fd.file_descriptor;   // the file descriptor for ION
    myionmem.ion_hostptr                     = ion_info.host_virtual_address; // the hostptr returned by ION which is device page size aligned
    if(myionmem.ion_hostptr % device_page_size)
    {
       error("Host pointer must be aligned to device_page_size!");
    }

    // Note that the image_row_pitch obtained by calling clGetDeviceImageInfoQCOM should be passed to clCreateImage2D
    image_object = clCreateImage2D(context, CL_MEM_USE_HOST_PTR|CL_MEM_EXT_HOST_PTR_QCOM, &image_fmt, imgw, imgh, image_row_pitch, &myionmem, &errcode);

    // Call clEnqueueMapImage before filling input image data
    pinput = clEnqueueMapImage(command_queue, image_object, CL_TRUE, CL_MAP_WRITE, origin, region, &row_pitch, NULL,
                               0, NULL, NULL, &errcode);

    // Fill the input image data using the hostptr and row_pitch returned by clEnqueueMapImage
    cl_uchar* inp = pinput;
    memset(inp, 0x0, (row_pitch * imgh));
    for(i = 0; i < (row_pitch * imgh); i+=row_pitch)
    {
       memset(inp+i, 0xff, imgw * element_size);
    }

    errcode =  clEnqueueUnmapMemObject(command_queue, image_object, pinput, 0, NULL, NULL);
```

Revision History

Revision 3, 2013/05/17: Generalized. Cleaned-up for Khronos. Added final token values.
Revision 2, 2012/11/01: Improved sample code.
Revision 1, 2012/10/18: Initial version.