Using zero-copy buffers on integrated GPUs

BRIAN KLOPPENBORG / APRIL 9, 2015 /
C/C++ (HTTPS://ARRAYFIRE.COM/CATEGORY/CC/), OPENCL (HTTPS://ARRAYFIRE.COM/CATEGORY/OPENCL/) /
1 COMMENT (HTTPS://ARRAYFIRE.COM/ZERO-COPY-ON-INTEGRATED-GPUS/#COMMENTS)

One of the most powerful aspects of parallel program on integrated GPUs is taking advantage of shared memory and caches. The best example of this is sharing common data between the CPU and GPU via. zero-copy buffers. This technique permits your program to avoid the O(N) cost of copying data to/from the GPU. This feature is particularly useful for applications that deal with real-time data streams, like video processing.

Creating zero copy buffers is not that difficult, but there is one caveat when it comes to using the buffers. First, lets consider how to make zero-copy buffers:

Method 1: OpenCL allocation of zero-copy buffers

The first method of allocating zero-copy buffers is quite simple: let OpenCL do it for you. To create a buffer in this fashion simply specify the `CL_MEM_ALLOC_HOST_PTR` argument to the `cl::Buffer` constructor (or `clCreateBuffer` if you are using the C API):

```
1
2 // Instruct OpenCL to allocate host memory
3 ::size_t size = N * sizeof(float)
4 Buffer d_results(context, CL_MEM_ALLOC_HOST_PTR, size);
5
6 ... launch kernels, etc
7
```

Method 2: Programmer-allocated aligned memory

In situations where you are streaming data or doing DMA between devices, it may be necessary to use aligned memory. In this situation, Intel OpenCL devices require that data be allocated with 4k alignment and be a multiple of 64 bytes (a cache line). At present AMD APUs follow this same alignment convention. Aligned allocation requires calls to OS-specific functions, after the memory is allocated, you can use create a buffer using CL_MEM_USE_HOST_PTR function:

```
1
2  ::size_t size = N * sizeof(float); // size must be a multiple of 64
3  ::size_t alignment = 4096;
4  // Allocate aligned memory on the host. The particular function is OS-dependent.
5  //  memalign(size_t alignment, size_t size) on Linux, release memory with free(...)
6  //  _aligned_malloc(size_t size, size_t alignment) on Windows, release with _aligned_free(...)
7  //  posix_memalign(void ** ptr, size_t alignment, size_t size) on OSX, release with free(...)
8  float * h_results = (float*) memalign(alignment, size);
9
10 // Create an OpenCL buffer using the host pointer
11 Buffer d_results = cl::Buffer(context, CL_MEM_USE_HOST_PTR, size, t_results);
12
```

Accessing zero-copy memory buffers

After your buffers are created, how do we access the data contained in them? Given method 2, one might think your code could directly access the elements of "h_results", but this probably won't work. Perhaps the most misunderstood aspect of zero-copy buffers in OpenCL relates to the state of the buffer after being used in a kernel invocation. It is often thought that the memory in the buffers will be updated immediately after the kernel

completes; however, the OpenCL specification **does not guarantee this behavior**. Instead, it is up to the programmer to ensure that the buffer is correctly updated prior to accessing the data. To do so, simply use the OpenCL map/unmap buffer commands:

```
1
2 // Get exclusive access to the buffer
3 float * temp = (float*) queue.enqueueMapBuffer(d_results, CL_TRUE, CL_MAP_READ, 0, size);
4
5 ... use the mapped buffer
6
7 // release the buffer back to OpenCL's control
8 queue.enqueueUnmapMemObject(d_results, temp);
```

Note that you must unmap the buffer after use to inform OpenCL that it has been given control over the memory location.

Further reading

Getting the Most from OpenCL™ 1.2: How to Increase Performance by Minimizing Buffer Copies on Intel® Processor Graphics (https://software.intel.com/en-us/articles/getting-the-most-from-opencl-12-how-to-increase-performance-by-minimizing-buffer-copies-on-intel-processor-graphics)

AMD OpenCL Optimization Guide: Mapping and zero-copy buffers (http://developer.amd.com/tools-and-sdks/opencl-zone/amd-accelerated-parallel-processing-app-sdk/opencl-optimization-guide/#50401315_pgfId-503784)

Related Posts:

INTEL (HTTPS://ARRAYFIRE.COM/TAG/INTEL/) INTEL HD GRAPHICS (HTTPS://ARRAYFIRE.COM/TAG/INTEL-HD-GRAPHICS/) INTEL IRIS GRAPHICS (HTTPS://ARRAYFIRE.COM/TAG/INTEL-IRIS-GRAPHICS/) OPENCL (HTTPS://ARRAYFIRE.COM/TAG/OPENCL/)

- Florian Hoenig

  Great article. You just saved me a lot of time reading individual vendor docs. Thank you!

Search

Follow this blog:

Subscribe via RSS (http://feeds.feedburner.com/arrayfire)

Recent Posts

ArrayFire v3.5 Official Release (https://arrayfire.com/arrayfire-v3-5-official-release/)

Visit ArrayFire at GTC 2017 (https://arrayfire.com/visit-arrayfire-at-gtc-2017/)

Thrombotherm: Analyzing Blood Platelets with ArrayFire (https://arrayfire.com/thrombotherm-analyzing-blood-platelets-with-arrayfire/)

ArrayFire at SC16 (https://arrayfire.com/arrayfire-at-sc16/)

ArrayFire v3.4 Official Release (https://arrayfire.com/arrayfire-v3-4-official-release/)

Categories

Android (https://arrayfire.com/category/android/)

Announcements (https://arrayfire.com/category/announcements/)

ArrayFire (https://arrayfire.com/category/arrayfire/)

Benchmarks (https://arrayfire.com/category/benchmarks-2/)

C/C++ (https://arrayfire.com/category/cc/)

Careers (https://arrayfire.com/category/careers-2/)

Case Studies (https://arrayfire.com/category/case-studies/)

Computer Vision (https://arrayfire.com/category/computer-vision-2/)

Computing Trends (https://arrayfire.com/category/computing-trends/)

CUDA (https://arrayfire.com/category/cuda-2/)

Events (https://arrayfire.com/category/events/)

Fortran (https://arrayfire.com/category/fortran/)