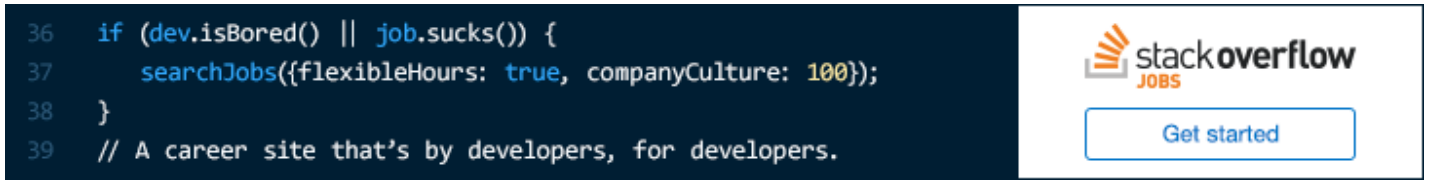
x Dismiss

Join the Stack Overflow Community

Stack Overflow is a community of 7.7 million programmers, just like you, helping each other. Join them; it only takes a minute:

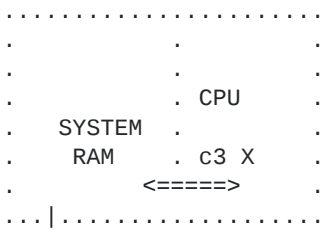
Sign up

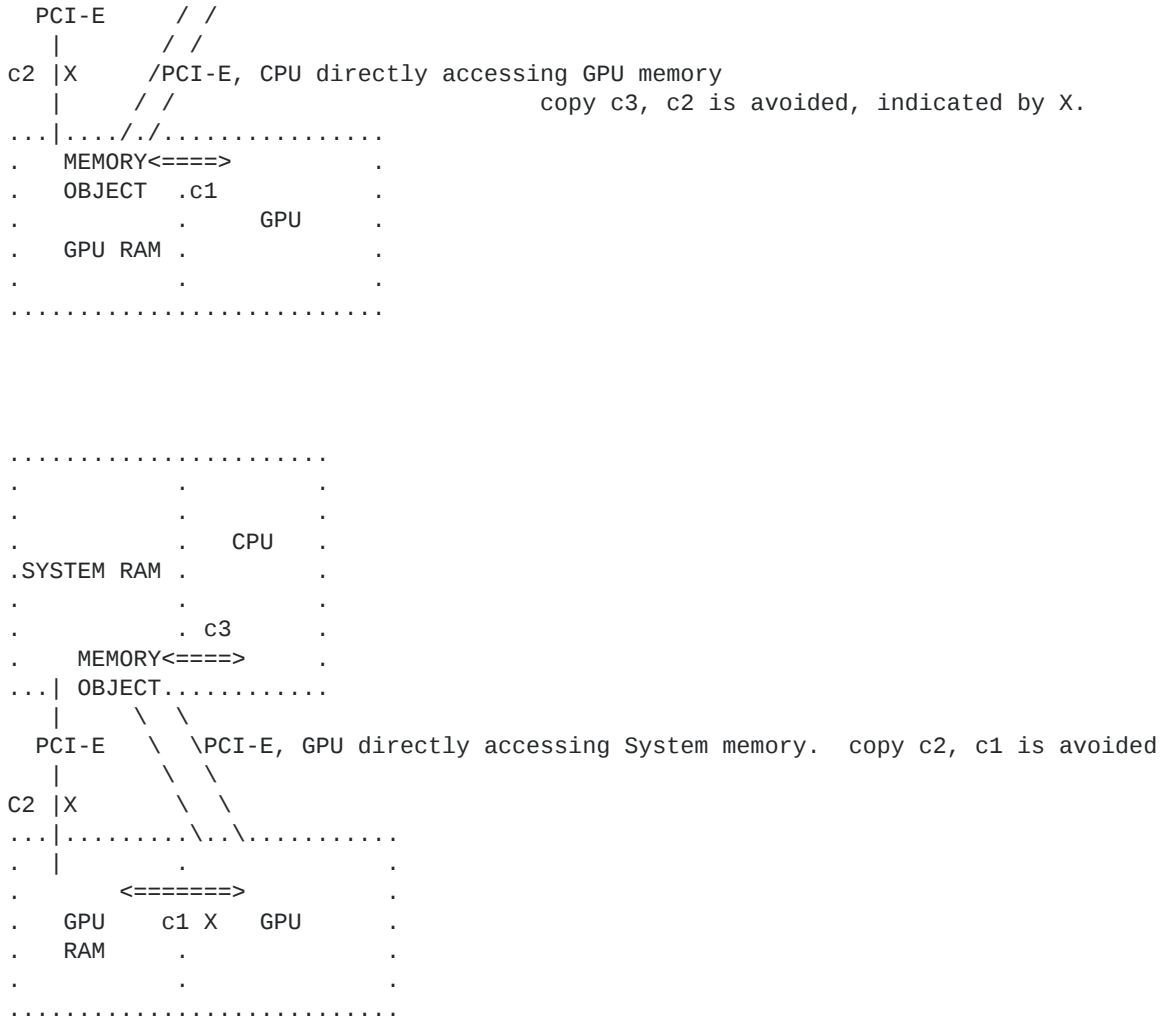
Access Path in Zero-Copy in OpenCL



I am a little bit confused about how exactly zero-copy work.

1- Want to confirm that the following corresponds to zero-copy in opencl.





The GPU/CPU is accessing System/GPU-RAM directly, without explicit copy.

2-What is the advantage of having this? PCI-e is still limiting the over all bandwidth. Or the only advantage is that we can avoid copies c2 & c1/c3 in above situations?

opencl

asked Oct 7 '12 at 6:16

gpuguy
1,649 10 40 87

1 Answer

You are correct in your understanding of how zero-copy works. The basic premise is that you can access either the host memory from the device, or the device memory from the host without needing to do an intermediate buffering step in between.

You can perform zero-copy by creating buffers with the following flags:

```
CL_MEM_AMD_PERSISTENT_MEM //Device-Resident Memory
CL_MEM_ALLOC_HOST_PTR // Host-Resident Memory
```

Then, the buffers can be accessed using memory mapping semantics:

```
void* p = clEnqueueMapBuffer(queue, buffer, CL_TRUE, CL_MAP_WRITE, 0, size, 0,
NULL, NULL, &err);
//Perform writes to the buffer p
err = clEnqueueUnmapMemObject(queue, buffer, p, 0, NULL, NULL);
```

Using zero-copy you could be able to achieve performance over an implementation that did the following:

1. Copy a file to a host buffer
2. Copy buffer to the device

Instead you could do it all in one step

1. Memory Map device side buffer
2. Copy file from host to device
3. Unmap memory

On some implementations, the calls of mapping and unmapping can hide the cost of data transfer. As in our example,

1. Memory Map device side buffer [Actually creates a host-side buffer of the same size]
2. Copy file from host to device [Actually writes to the host-side buffer]

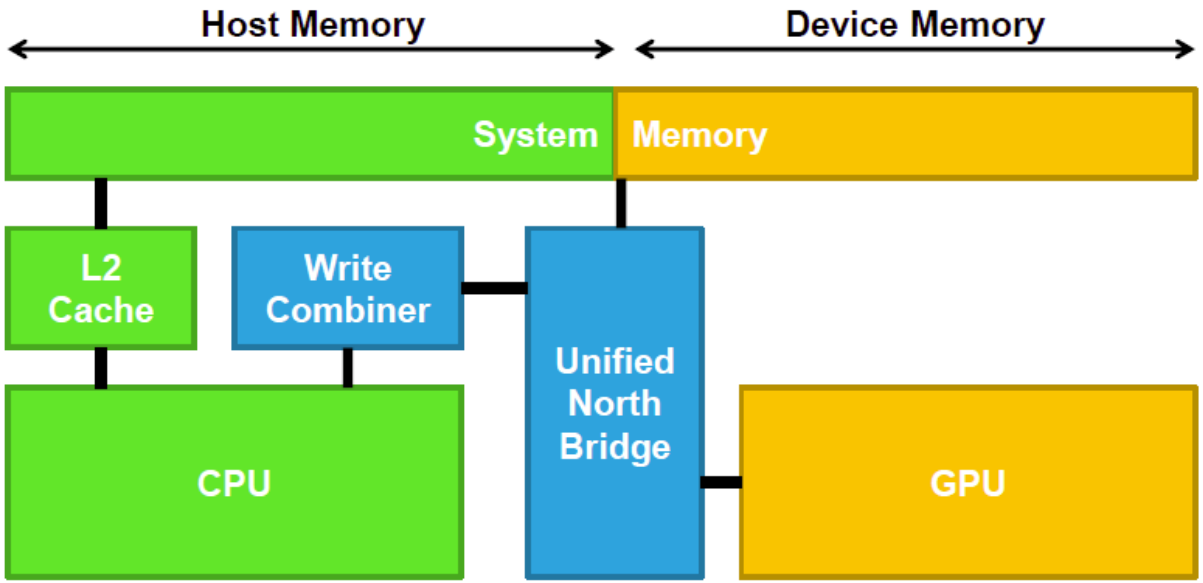
3. Unmap memory [Actually copies data from host-buffer to device-buffer via clEnqueueWriteBuffer]

If the implementation is performing this way, then there will be no benefit to using the mapping approach. However, AMDs newer drivers for OpenCL allow the data to be written directly, making the cost of mapping and unmapping almost 0. For discrete graphics cards, the requests still take place over the PCIe bus, so data transfers can be slow.

In the case of an APU architecture, however, the costs of data transfers using the zero-copy semantics can greatly increase the speed of transfers due to the APUs unique architecture (pictured below). In this architecture, the PCIe bus is replaced with the Unified North Bridge (UNB) that allows for faster transfers.

BE AWARE that when using zero-copy semantics with the memory-mapping, that you will see absolutely horrendous bandwidths when reading a device-side buffer from the host. These bandwidths are on the order of 0.01 Gb/s and can easily become a new bottleneck for your code.

Sorry if this is too much information. This was my thesis topic.



answered Oct 8 '12 at 17:24

KLee1

3,871 3 22 39

+5 for detailed explanation! – gpuguy Oct 10 '12 at 5:32

Can we see your thesis? Sounds awesome! I played a lot with apus for my thesis too – homemade-jam Apr 27 '13 at 12:31

@homemade-jam Sure. Here you go: scholar.lib.vt.edu/theses/available/etd-07272012-152625/… – KLee1 Apr 28 '13 at 17:16

I wonder if MOVNTDQA fixes the 'absolutely horrendous bandwidths when reading a device-side buffer from the host.' Have you tried that? – ArchaeaSoftware Jul 11 '14 at 17:01

Would I be able to use this to copy pixels directly to the framebuffer to be displayed on the next page flip? – enigmaticPhysicist Mar 7 '15 at 7:11