Host-Device Memory Transfer Optimization

Zhongliang Chen

4-6-2015

Outline

- OpenCL memory objects
  - Memory types
  - Placement
  - Mapping and zero copy memory
  - Hardware command queues
- Data transfer options
  - Regular buffers
  - Zero copy buffers
  - Pre-pinned buffers

Memory Types

- Host memory
  - Unpinned
  - Pinned
    - Regular
    - Device-visible
      - Uncached write combining
- Device memory
  - Regular
  - Host-visible

Bandwidth

	CPU Read	CPU Write	GPU Shader Read	GPU Shader Write	GPU DMA Read	GPU DMA Write
CPU Memory	10 - 20	10 - 20	9 - 10	2.5	11 - 12	11 - 12
GPU Memory	0.01	9 - 10	230	120 - 150	N/A	N/A

Unpinned CPU Memory

- Must be pinned before transfer.
  - Medium performance.
- Transfer mechanisms:
  - <= 32KB
    - Data is copied to pinned buffer and transferred by DMA.
  - > 32KB and <= 16MB
    - Pages are pinned, transferred by DMA, and unpinned.
  - > 16MB
    - Blocks of 16MB are pinned and transferred by DMA. Double buffering is used to overlap pinning and transfer.

Pinned CPU Memory

- Locked in CPU memory.
  - Not swapped out.
  - Limited size.
- No staging before transfer.
  - Better performance than unpinned CPU memory.
- GPU accessible
  - Discrete GPU: limited bandwidth through PCIe.
  - APU: slow due to cache coherency traffic.

GPU-Visible CPU Memory

- Pinned. Limited size.
- GPU accessible.
  - No cache coherency traffic.
  - Higher bandwidth than regular pinned memory.
- CPU accessible.
  - A portion is configured as uncached reads and combining writes.
  - Slow reads and scattering writes.
  - Fast streaming writes.
- Used as “GPU memory” on APU.

GPU Memory

- GPU accessible at high bandwidth.
- CPU inaccessible directly.

CPU-Visible GPU Memory

- Limited size.
- GPU accessible.
  - Full bandwidth.
- CPU accessible.
  - Low bandwidth through PCIe.
  - Uncached reads and combining writes.
    - Slow reads and scattering writes.
    - Fast streaming writes.

Memory Placement

```
cl_mem clCreateBuffer (cl_context context,
                       cl_mem_flags flags,
                       size_t size,
                       void *host_ptr,
                       cl_int *errcode_ret)
```

- Deferred allocation
  - Memory space is not allocated until the data is first used.
    - A memory object has a location on every device in the context.
    - The first access is slower than subsequent accesses.

cl_mem_flags
CL_MEM_READ_WRITE
CL_MEM_WRITE_ONLY
CL_MEM_READ_ONLY
CL_MEM_USE_HOST_PTR
CL_MEM_ALLOC_HOST_PTR
CL_MEM_COPY_HOST_PTR
CL_MEM_HOST_WRITE_ONLY
CL_MEM_HOST_READ_ONLY
CL_MEM_HOST_NO_ACCESS

Memory Placement

```
cl_mem clCreateBuffer (cl_context context,
                       cl_mem_flags flags,
                       size_t size,
                       void *host_ptr,
                       cl_int *errcode_ret)
```

Default (none of the following flags)	Discrete GPU	Device memory	Copy	Host memory (different memory area can be used on each map).
	APU	Device-visible host memory		
	CPU	Use *Map Location* directly	Zero copy	
CL_MEM_ALLOC_HOST_PTR, CL_MEM_USE_HOST_PTR (clCreateBuffer when VM is enabled)	Discrete GPU	Pinned host memory shared by all devices in context (unless only device in context is CPU; then, host memory)	Zero copy	Use Location directly (same memory area is used on each map).
	APU			
	CPU			

cl_mem_flags
CL_MEM_READ_WRITE
CL_MEM_WRITE_ONLY
CL_MEM_READ_ONLY
CL_MEM_USE_HOST_PTR
CL_MEM_ALLOC_HOST_PTR
CL_MEM_COPY_HOST_PTR
CL_MEM_HOST_WRITE_ONLY
CL_MEM_HOST_READ_ONLY
CL_MEM_HOST_NO_ACCESS

Memory Placement

```
cl_mem clCreateBuffer (cl_context context,
                       cl_mem_flags flags,
                       size_t size,
                       void *host_ptr,
                       cl_int *errcode_ret)
```

CL_MEM_ALLOC_HOST_PTR, CL_MEM_USE_HOST_PTR (for clCreateImage and clCreateBuffer without VM)	Discrete GPU	Device memory	Copy	Pinned host memory, unless only device in context is CPU; then, host memory (same memory area is used on each map).
	APU	Device-visible memory		
	CPU		Zero copy	

CL_MEM_USE_PERSISTENT_MEM_AMD (when VM is enabled)	Discrete GPU	Host-visible device memory	Zero copy	Use *Location* directly (different memory area can be used on each map).
	APU	Host-visible device memory		
	CPU	Host memory		

cl_mem_flags
CL_MEM_READ_WRITE
CL_MEM_WRITE_ONLY
CL_MEM_READ_ONLY
CL_MEM_USE_HOST_PTR
CL_MEM_ALLOC_HOST_PTR
CL_MEM_COPY_HOST_PTR
CL_MEM_HOST_WRITE_ONLY
CL_MEM_HOST_READ_ONLY
CL_MEM_HOST_NO_ACCESS

Placement Optimization

- Using CPU
  - Create memory objects with `CL_MEM_ALLOC_HOST_PTR` and use map / unmap rather than read / write APIs.
    - Zero copy between host memory and application buffer.
- Using (CPU and GPU) or APU
  - Create memory objects with `CL_MEM_USE_PERSISTENT_MEM_AMD`.
    - Zero copy between host and device.

Buffers vs. Images

- CPU
  - No dedicated hardware for image accesses.
  - Buffers may be preferred if no sampling is needed.
- GPU
  - Dedicated hardware (texture) for image accesses.
  - The best choice is made based on specific memory access patterns.

Memory Mapping

```
void * clEnqueueMapBuffer (cl_command_queue command_queue,
                           cl_mem buffer,
                           cl_bool blocking_map,
                           cl_map_flags map_flags,
                           size_t offset,
                           size_t size,
                           cl_uint num_events_in_wait_list,
                           const cl_event *event_wait_list,
                           cl_event *event,
                           cl_int *errcode_ret)
```

```
cl_int clEnqueueUnmapMemObject (cl_command_queue command_queue ,
                                cl_mem memobj,
                                void *mapped_ptr,
                                cl_uint num_events_in_wait_list ,
                                const cl_event *event_wait_list ,
                                cl_event *event )
```

- Another way to access OpenCL memory objects in addition to Read/Write.
- May provide higher performance.
- Zero copy or copy

Zero Copy vs. Copy

- Zero copy memory objects
  - Located on host / device but accessed directly by host.
  - Not transferred.
- Copy memory objects
  - Located on device.
  - Transferred from / to device when mapped / unmapped.

Zero Copy Benefits

- No transfers performed when mapped / unmapped.
- Better performance when
  - Host memory is accessed by device in a sparse manner, or
  - Copies are too expensive.
- Fast streaming writes.

How to Allocate Zero Copy

- Host resident
  - `CL_MEM_USE_HOST_PTR` or `CL_MEM_ALLOC_HOST_PTR`.
- Device resident
  - `CL_MEM_USE_PERSISTENT_MEM_AMD`.

Copy Transfer Policy

```
void * clEnqueueMapBuffer (cl_command_queue command_queue,
                           cl_mem buffer,
                           cl_bool blocking_map,
                           cl_map_flags map_flags,
                           size_t offset,
                           size_t size,
                           cl_uint num_events_in_wait_list,
                           const cl_event *event_wait_list,
                           cl_event *event,
                           cl_int *errcode_ret)
```

Flags	Transfer on map	Transfer on unmap
CL_MAP_READ	Device to host, if map location is not current.	None.
CL_MAP_WRITE	Device to host, if map location is not current.	Host to device.
CL_MAP_READ | CL_MAP_WRITE	Device to host, if map location is not current.	Host to device.
CL_MAP_WRITE_INVALIDATE_REGION	None.	Host to device.

Map Location

- `map` APIs returns the location where the data is mapped.
- Behaviors depend on the memory types.
  - `CL_MEM_USE_HOST_PTR`
  - `CL_MEM_ALLOC_HOST_PTR`
  - `CL_MEM_USE_HOST_PTR`

```
cl_mem clCreateBuffer (cl_context context,
                       cl_mem_flags flags,
                       size_t size,
                       void *host_ptr,
                       cl_int *errcode_ret)
```

CL_MEM_USE_HOST_PTR

- Pinned.
- `host_ptr` used as the map location.
- Best practice.
  - Align the memory to 4KB to minimize pinning costs.
  - If host memory is updated only once.
    - Use `CL_MEM_ALLOC_HOST_PTR | CL_MEM_COPY_HOST_PTR` instead.
  - If host memory is updated multiple times.
    - Align the memory to the data type size in the kernel.
  - If device memory is updated.
    - Use `CL_MEM_USE_PERSISTENT_MEM_AMD` and `clEnqueueWriteBuffer`.

CL_MEM_ALLOC_HOST_PTR

- The same location is used for all maps.
- Runtime tracks if the location contains an up-to-date copy.
  - If so, no transfer from device on `CL_MAP_READ`.

CL_MEM_COPY_HOST_PTR

- May have different map locations.
- With `CL_MEM_ALLOC_HOST_PTR`.
  - Pinned and initialized.
- Without `CL_MEM_ALLOC_HOST_PTR`.
  - Runtime copies the data to a temporary buffer.
  - Data is copied at the first use.
    - Performance penalty.
  - Best practice: use `clEnqueueWriteBuffer` instead.

Reading / Writing / Copying

- Host <-> Device
- Host <-> Host
  - `memcpy`.
  - May be slow if reading from device-visible host memory.
- Device <-> Device
  - Kernels used for copying buffers.
  - Kernels used for converting from/to linear addresses mode for images.

Command Queue Optimization

- Use non-blocking commands.
- Use events to specify dependencies.
  - The best case is no dependency where the shader and copy (DMA) engines can run in parallel.
- SW and HW queues
  - SW queues are assigned to HW queues on creation time. `hw_queue_id = sw_queue_id % num_hw_queues`.
  - Multiple HW queues is beneficial for small kernels.
    - AMD's latest GPUs have 8 asynchronous copy engines.

Review of Definitions

- Deferred allocation
  - Buffers are not allocated until their first use.
    - First accesses may be slower than subsequent accesses.
- Peak interconnect bandwidth
  - E.g., 16 GB/s with a PCIe 3.0 16x bus
- Pinning
  - Pages in host memory are locked before transferred from / to device.
    - Pinning cost.

Review of Definitions (cont.)

- WC: Write Combine
  - Multiple adjacent writes are combined into cache lines before sent to the external bus.
    - Fast streaming writes.
- Uncached accesses
  - Accesses without using caches.
    - Typically very slow.

Review of Definitions (cont.)

- USWC: Uncached Speculative WC
  - Device-accessible host memory without causing cache coherency traffic
    - Fast streaming writes.
    - Slow reads and scattering writes.

Outline

- OpenCL memory objects
  - Memory types
  - Placement
  - Mapping and zero copy memory
  - Hardware command queues
- Data transfer options
  - Regular buffers
  - Zero copy buffers
  - Pre-pinned buffers

Types of OpenCL Buffers

- Regular device buffers
  - Allocated with `CL_MEM_READ_ONLY`, `CL_MEM_WRITE_ONLY`, or `CL_MEM_READ_WRITE`.
  - Placed on the device memory.
  - Accessed by device at high bandwidth.

Types of OpenCL Buffers (cont.)

- Zero copy buffers
  - Placed on host or device memory.
  - Accessed through hardware interconnect (PCIe).
  - Mapped/Unmapped at low cost.

Types of OpenCL Buffers (cont.)

- Pre-pinned buffers
  - Created with `CL_MEM_ALLOC_HOST_PTR` or `CL_MEM_USE_HOST_PTR`.
  - Used by `clEnqueueCopyBuffer` / `clEnqueueReadBuffer` / `clEnqueueWriteBuffer` at peak interconnect bandwidth.

How to Create Buffers?

- `CL_MEM_ALLOC_HOST_PTR` / `CL_MEM_USE_HOST_PTR`
  - Zero copy buffers on host.
  - Directly accessible by host at host memory bandwidth.
  - Directly accessible by device through interconnect.
  - Pre-pinned sources or destinations for accesses at peak interconnect bandwidth.
  - May be USWC memory (`CL_MEM_ALLOC_HOST_PTR | CL_MEM_READ_ONLY`).

How to Create Buffers? (cont.)

- `CL_MEM_USE_PERSISTENT_MEM_AMD`
  - Zero copy buffers on device.
  - Directly accessible by device at device memory bandwidth.
  - Directly accessible by host through interconnect.
    - Typically with high streaming write bandwidth, but low read and scattering write bandwidth.
  - Copyable from / to device at peak interconnect bandwidth.
  - Very small size

Scenarios

- Application allocates buffers and transfers them with OpenCL.
- Application lets OpenCL allocate and transfer buffers.

Option 1

- `clEnqueueWriteBuffer` / `clEnqueueReadBuffer`.
  - `malloc` and `CL_MEM_USE_HOST_PTR`.

```
pinnedBuffer = clCreateBuffer(CL_MEM_ALLOC_HOST_PTR or CL_MEM_USE_HOST_PTR)
deviceBuffer = clCreateBuffer()
pinnedMemory = clEnqueueMapBuffer(pinnedBuffer)
clEnqueueRead/WriteBuffer(deviceBuffer, pinnedMemory)
clEnqueueUnmapMemObject(pinnedBuffer, pinnedMemory)
```

```
pinnedBuffer = clCreateBuffer(CL_MEM_ALLOC_HOST_PTR or CL_MEM_USE_HOST_PTR)
deviceBuffer = clCreateBuffer()
clEnqueueRead/WriteBuffer(deviceBuffer, pinnedBuffer)
```

Pinning Cost!

Option 2

- clEnqueueCopyBuffer.

```
pinnedBuffer = clCreateBuffer(CL_MEM_ALLOC_HOST_PTR or CL_MEM_USE_HOST_PTR)
deviceBuffer = clCreateBuffer()
pinnedMemory = clEnqueueMapBuffer(pinnedBuffer)
Application modifies pinnedMemory
clEnqueueUnmapMemObject(pinnedBuffer, pinnedMemory)
clEnqueueCopyBuffer(pinnedBuffer, deviceBuffer)
```

```
pinnedBuffer = clCreateBuffer(CL_MEM_ALLOC_HOST_PTR or CL_MEM_USE_HOST_PTR)
deviceBuffer = clCreateBuffer()
clEnqueueCopyBuffer(pinnedBuffer, deviceBuffer)
pinnedMemory = clEnqueueMapBuffer(pinnedBuffer)
Application reads pinnedMemory
clEnqueueUnmapMemObject(pinnedBuffer, pinnedMemory)
```

Option 3

- clEnqueueMapBuffer / clEnqueueUnmapMemObject.

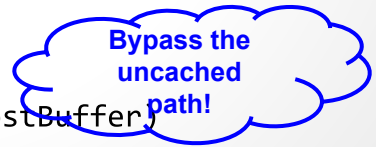
```
pinnedBuffer = clEnqueueMapBuffer(deviceBuffer, CL_MAP_WRITE or CL_MAP_READ)
Application writes or reads pinnedBuffer
clEnqueueUnmapMemObject(deviceBuffer, pinnedBuffer)
```

Option 4

- Zero copy device buffer directly accessed by host.

```
zeroCopyDeviceBuffer = clCreateBuffer(CL_MEM_USE_PERSISTENT_MEM_AMD)
hostBuffer = clEnqueueMapBuffer(zeroCopyDeviceBuffer)
Application writes hostBuffer
clEnqueueUnmapMemObject(zeroCopyDeviceBuffer, hostBuffer)

// optional
clEnqueueReadBuffer(zeroCopyDeviceBuffer)
      // or
clEnqueueCopyBuffer(zeroCopyDeviceBuffer, hostBuffer)
```



Option 5

- Zero copy host buffer directly accessed by device.

```
zeroCopyHostBuffer = clCreateBuffer(CL_MEM_ALLOC_HOST_PTR)
hostBuffer = clEnqueueMapBuffer(zeroCopyHostBuffer, CL_MAP_READ | CL_MAP_WRITE)
Application reads or writes hostBuffer
clEnqueueUnmapMemObject(hostBuffer, zeroCopyHostBuffer)
clEnqueueNDRangeKernel()
```

Be aware that GPU kernel bandwidth is an order of magnitude lower compared to a regular device buffer.

Demo

`BufferBandwidth`

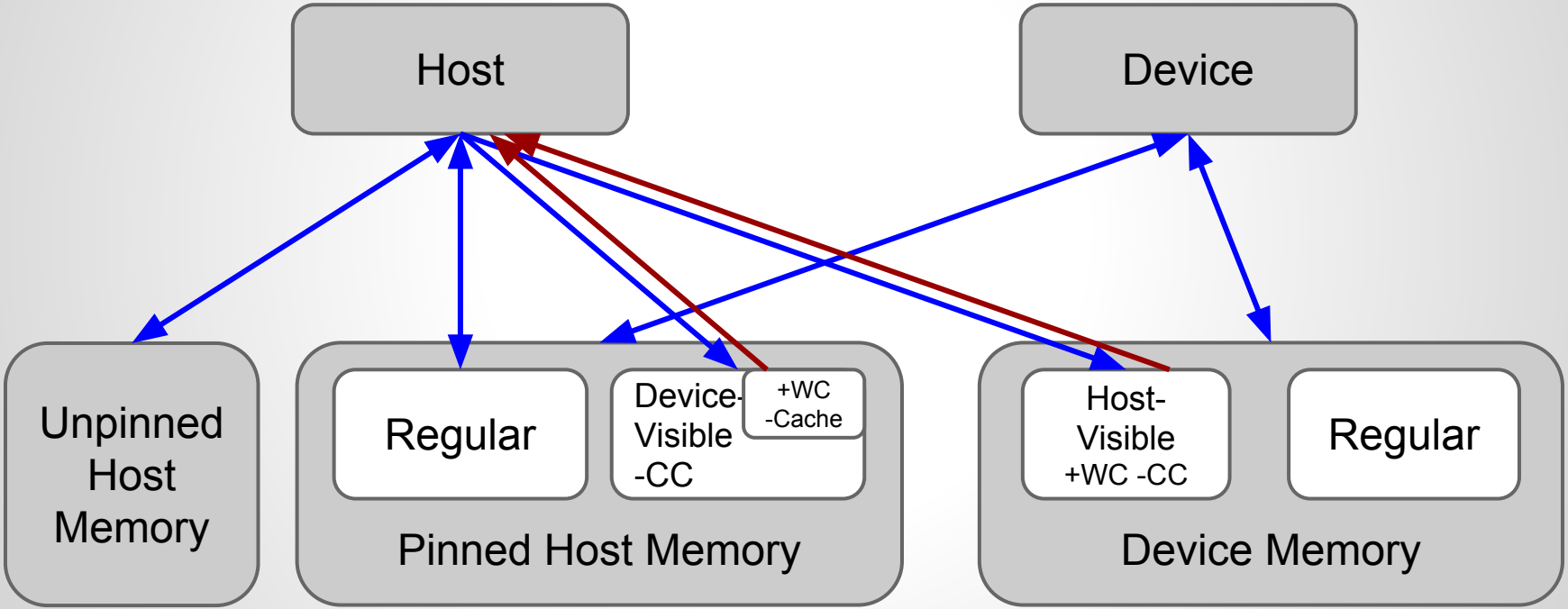
Summary

- OpenCL memory types, placement, and mapping.
- Data transfer options: regular, zero copy, or pre-pinned buffers.

- General principle

Decrease host-device transfer as much as possible.
If you still have to use it, choose the best practice.

Takeaway


Host
Device
Unpinned Host Memory
Regular
Device-Visible -CC
+WC -Cache
Pinned Host Memory
Host-Visible +WC -CC
Regular
Device Memory

Thank you!

Questions?

Reference

- OpenCL university kit
- AMD OpenCL Programming User Guide
- AMD OpenCL Programming Optimization Guide
- Heterogeneous Computing with OpenCL
  - By Benedict Gaster, Lee Howes, David R. Kaeli, Perhaad Mistry and Dana Schaa