AMD

Fusion11

DEVELOPER SUMMIT

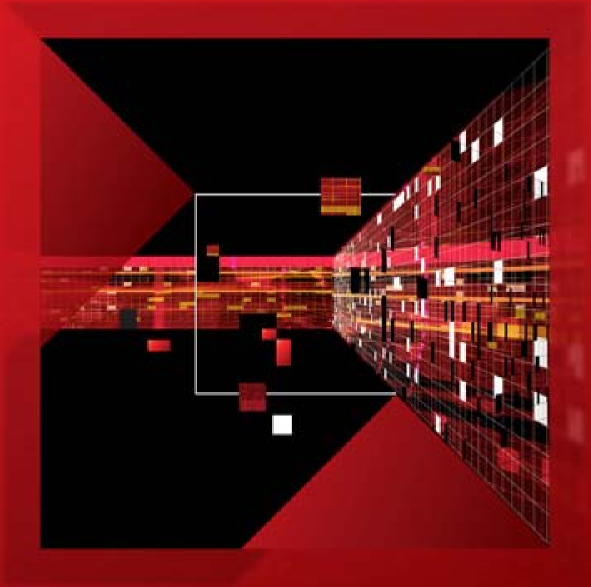
MEMORY SYSTEM ON FUSION APUS

The Benefits of Zero Copy

Pierre Boudier
AMD
Fellow of OpenGL/OpenCL

Graham Sellers
AMD
Manager of OpenGL

INTRODUCTION



INTRODUCTION

- On the APU, one of the key parts of the system is the data path between the GPU and memory
  - Provides low latency access for CPU cores (optimized around caches)
    - Random access, branchy, single threaded, scalar code
  - Provides high throughput access for GPU cores (optimized around latency hiding)
    - Streaming, vectorized, massively multithreaded, data-intensive code

- Llano introduces two new buses for the GPU to access memory:
  - AMD Fusion Compute Link (ONION):
    - This bus is used by the GPU when it needs to snoop the CPU cache, so is a coherent bus
    - This is used for cacheable system memory
  - Radeon Memory Bus (GARLIC):
    - This bus is directly connected to memory and can saturate memory bandwidth, so is a non coherent bus
    - This is used for local memory and USWC (uncached speculative write combine)

TERMINOLOGY

- WC: Write Combine buffers
  - There are 4 WC buffers per core
    - One WC buffer is automatically assigned for a write operation
    - If the writes are contiguous, then it is efficient
    - If there are many non contiguous writes, then partial WC buffer flushes will lower the efficiency
  - The WC buffers are automatically flushed to memory when the GPU is accessed

- Cacheable memory:
  - This is the traditional L1/L2 architecture on AMD CPUs
  - CPU accesses are fast for both read and write
  - Multithreading (from multiple cores) is often necessary to saturate full bandwidth
  - When the GPU accesses this type of memory, the caches are snooped to ensure coherency

TERMINOLOGY CONTINUED

- USWC: Uncached Speculative Write Combined
  - CPU reads are uncached (slow), CPU writes go through the WC buffers
  - GPU access to this type of memory does not need CPU cache probing

- Local video memory:
  - Memory managed by the graphics driver, not available to the OS for generic CPU processes

- Memory pinning and locking:
  - Operation done by the OS for access of system pages by the GPU:
    - Make the page resident (no longer in the swap file)
    - Remove this page from regular CPU paging operation

TERMINOLOGY CONTINUED

- TLB: Translation Lookaside Buffer
  - A dedicated cache used to store the result of page translation
    - Scattered access patterns in virtual address space can cause misses
    - Large pages can be used to limit TLB misses (on both GPU and CPU)

- UNB: Unified North Bridge
  - Arbitrates memory traffic from the GPU client

THE GPU IN THE LLANO SYSTEM

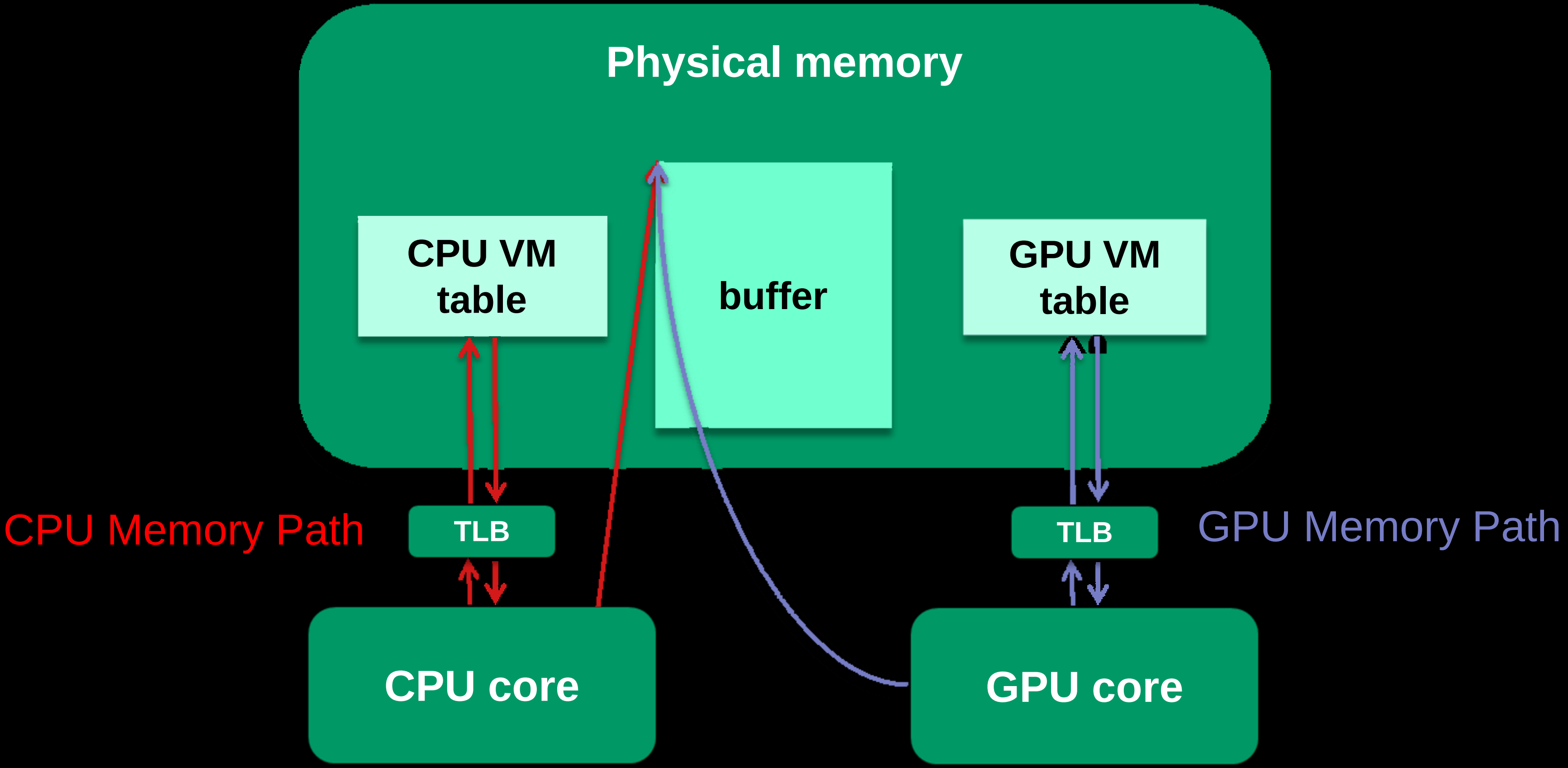
- On Llano, the GPU core is still exposed as a separate graphics engine
  - The GPU is managed by the OS via drivers
    - Leverage existing driver stacks to supports the current ecosystem
  - Memory is split into regular system memory, and carved out “local memory”
    - Same function as the local video memory on discrete
    - Allows the GPU memory controller to optimize throughput and priorities for GFX clients
- Existing and familiar APIs can be used to access the GPU core
  - OpenCL ™, OpenGL ®, DirectX ® and multimedia, …

GPU IN THE SYSTEM

- Both CPU and GPU have their own set of page tables and TLB
  - The memory is generally not coherent
  - The GPU can probe the CPU cache …
  - … but the CPU relies on the driver for synchronization (map/unmap, lock/unlock flush GPU caches)

- The current programming model is a direct consequence:
  - CPU access will page fault on a single access, and the OS will page in/out on demand
  - GPU access is known upfront, and the driver or OS will page in/out on scheduling

SEPARATE PAGE TABLE MANAGEMENT FOR CPU/GPU

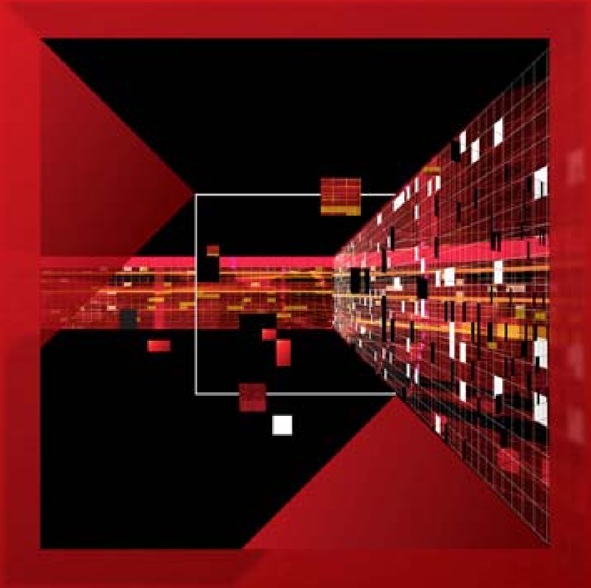


WHAT IS ZERO COPY?

- Many different meanings:
  - A kernel access system memory directly for either read or write
  - A DMA transfer access system memory directly without copying into USWC
  - The CPU directly writes into local memory without doing any DMA

- OpenCL offers several mechanisms to effectively reduce extra copying
  - On Llano, this matters even more than on discrete because bandwidth is shared
- OpenGL has some driver optimizations and some proprietary extensions

DIFFERENT DATA PATHS



EXAMPLES

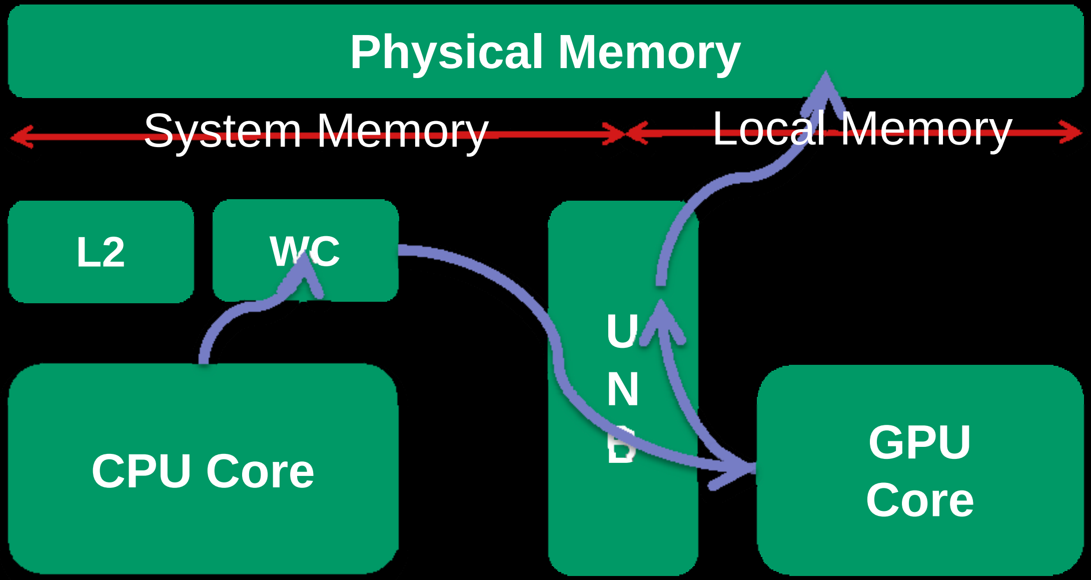
- The next slides will describe the different memory access by the various cores
  - CPU access to local memory
  - CPU access to uncached memory
  - CPU access to cacheable memory
  - GPU access to local memory
  - GPU access to uncached memory
  - GPU access to cacheable memory

CPU ACCESS TO LOCAL MEMORY

- CPU writes into local framebuffer
  - On Llano, this can peak at 8 GB/s
    - On discrete, this was limited by the PCIe bus to around 6 GB/s
  - The data first goes through the WC buffers on the CPU, then goes to the GPU core in order to get the physical address

- CPU reads from local framebuffer
  - Those are still very slow
    - Accesses are uncached
    - Only a single outstanding read is supported

DATAPATH IN THE SYSTEM



SAMPLE CODE (OPENCL)

- Create the buffer with the **GL_MEM_USE_PERSISTENT_MEM_AMD** flag

```
clCreateBuffer(context,
               CL_MEM_READ_ONLY | CL_MEM_USE_PERSISTENT_MEM_AMD,
               bufSize, 0, &error);

clEnqueueMapBuffer(cmd_queue,
                   buffer, CL_TRUE, CL_MAP_WRITE,
                   0, bufSize, 0, NULL, NULL, &error);
```

SAMPLE CODE (OPENGL)

- Under driver control, the PBO allocation can be done in local memory with direct CPU write

```
glGenBuffers(1, &buffer);
glBindBuffer(GL_PIXEL_UNPACK_BUFFER, buffer);
glBufferData(GL_PIXEL_UNPACK_BUFFER, bufferSize, NULL, GL_STATIC_DRAW);
glMapBufferRange(GL_PIXEL_UNPACK_BUFFER, 0, bufferSize,
                 GL_MAP_WRITE_BIT | GL_MAP_INVALIDATE_BUFFER_BIT);
```

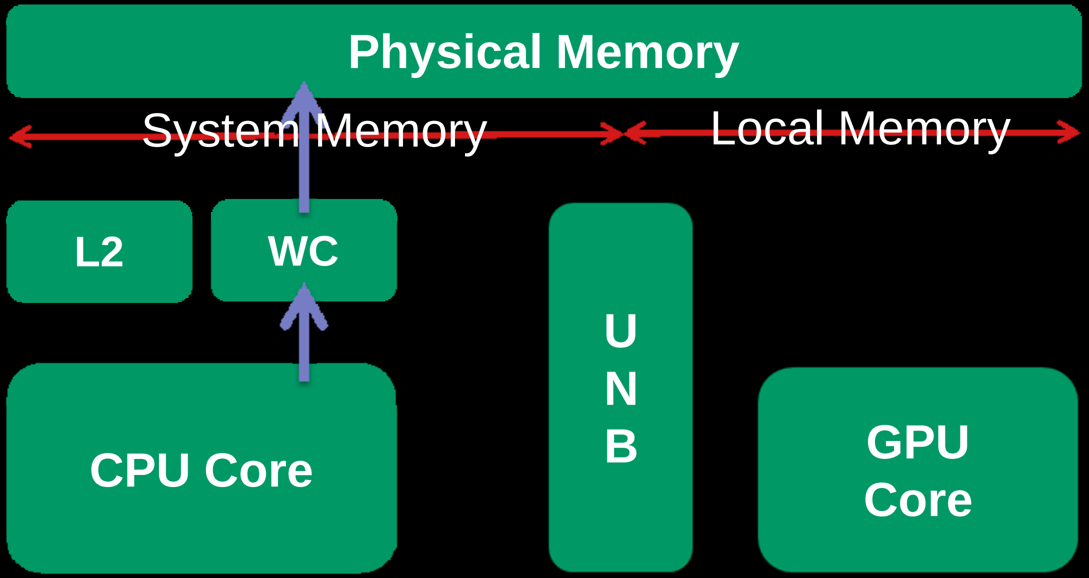
CPU ACCESS TO UNCACHED MEMORY

- CPU writes into uncached memory
  - The writes go through the WC
  - This avoids polluting the CPU cache, when it is known that there will be no cache hit for reads
  - This allows further access by the GPU for this memory without snooping the cache

- CPU reads will first flush the WC, then will be uncached (slow)

- Multithreaded writes can be used to improve bandwidth using multiple CPU cores (~8-13GB/s)
  - Each core has its own set of WC buffers

DATAPATH IN THE SYSTEM



SAMPLE CODE

- OpenCL

```
clCreateBuffer(context, CL_MEM_READ_ONLY | CL_MEM_ALLOC_HOST_PTR,
               bufSize, 0, &error);
clEnqueueMapBuffer(cmd_queue, buffer, CL_TRUE, CL_MAP_WRITE,
                   0,bufSize,0,NULL,NULL, &error);
```

- OpenGL

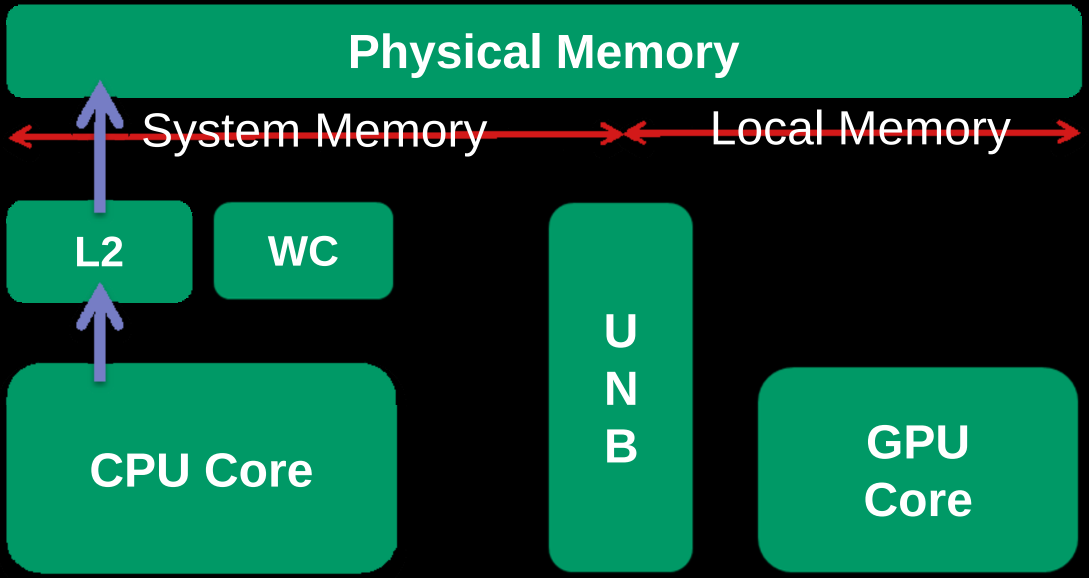
```
glBufferData(GL_PIXEL_UNPACK_BUFFER, size, 0, GL_STREAM_DRAW)
glMapBuffer(GL_PIXEL_UNPACK_BUFFER, GL_WRITE_ONLY)
```

CPU ACCESS TO CACHEABLE MEMORY

- CPU accesses to cacheable memory
  - This is the typical case in C++ code (no difference to discrete)
  - Single threaded performance: ~8GB/s for either read or write
  - Multi-threaded performance: ~13GB/s for either read or write

- The memory can be accessed by the GPU:
  - Pages need to be made resident by the OS, and locked to prevent paging
  - Physical pages need to be programmed into the GPU HW virtual memory page tables
  - Implications:
    - The two operations are done by the device driver (compute/graphics)
    - They take time, so should be done at initialization time if possible
    - There is a limit to how much cacheable memory can be accessed, because it is removed from normal OS usage

DATAPATH IN THE SYSTEM



SAMPLE CODE

- OpenCL

```
clCreateBuffer(context, CL_MEM_READ_WRITE | CL_MEM_ALLOC_HOST_PTR,
               bufSize, 0, &error);
clEnqueueMapBuffer(cmd_queue, buffer, CL_TRUE, CL_MAP_READ,
                   0, bufSize, 0, NULL, NULL, &error);
```

- OpenGL (Typically used for glReadPixels optimization)

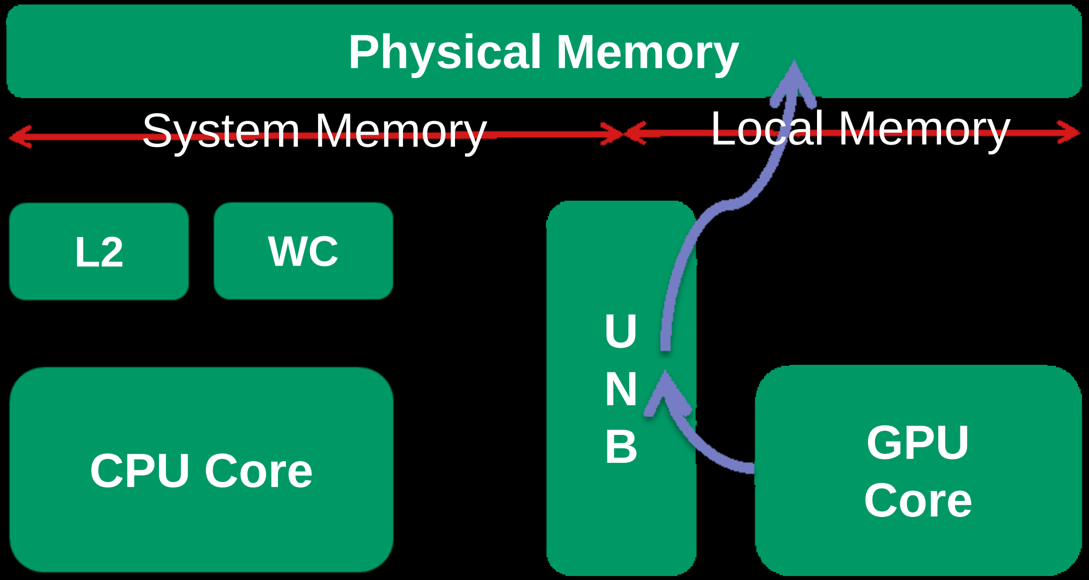
```
glBufferData(GL_PIXEL_PACK_BUFFER, size, 0, GL_STREAM_READ);
glMapBuffer(GL_PIXEL_PACK_BUFFER, GL_READ_ONLY);
```

GPU ACCESS TO LOCAL MEMORY

- GPU reads from local framebuffer
  - This is the optimal path to memory:
    - Radeon Memory Bus (GARLIC) avoids any cache snooping
    - Memory is interleaved to increase throughput efficiency
  - Kernels and shaders can saturate dram bandwidth (measured at ~17GB/s)

- GPU writes to local framebuffer are similar
  - Kernels and shaders can saturate dram bandwidth (measured at ~13 GB/s)

DATAPATH IN THE SYSTEM



SAMPLE CODE

- OpenCL

```
clCreateBuffer(context, CL_MEM_READ_WRITE, bufSize, 0, &error);
```

  Then used directly in the kernel

- OpenGL

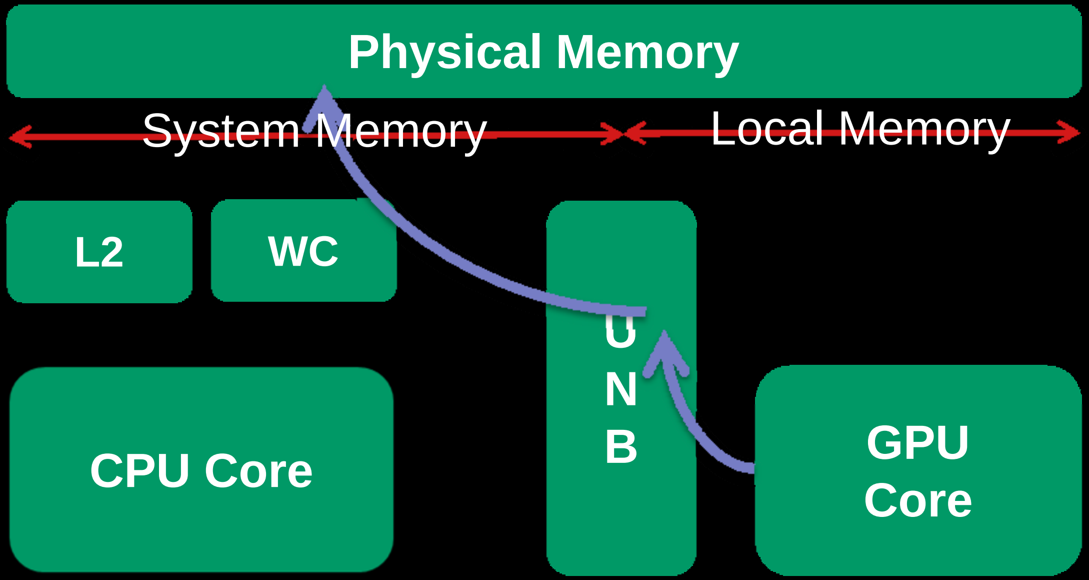
```
glBufferData(GL_ARRAY_BUFFER, size, 0, GL_STATIC_DRAW);
```

  Then used as texture buffer, vertex buffer, transform feedback buffer, …

GPU ACCESS TO UNCACHED MEMORY

- GPU accesses to uncached memory
  - This uses the Radeon Memory Bus (GARLIC)
  - Memory does not have the same interleaving granularity as local memory
  - So slightly lower performance than local memory, but faster than cacheable memory
  - Reads can saturate dram bandwidth (measured at 12 GB/s)
  - Writes are similarly fast but …
    - Usually avoided, however, since CPU reads are really slow from uncached space

DATAPATH IN THE SYSTEM



SAMPLE CODE

- OpenCL

```
clCreateBuffer(context, CL_MEM_READ | CL_MEM_ALLOC_HOST_PTR,
               bufSize, 0, &error);
```

Then used directly in the kernel

- OpenGL

Under driver control (for instance if the buffer is often mapped)

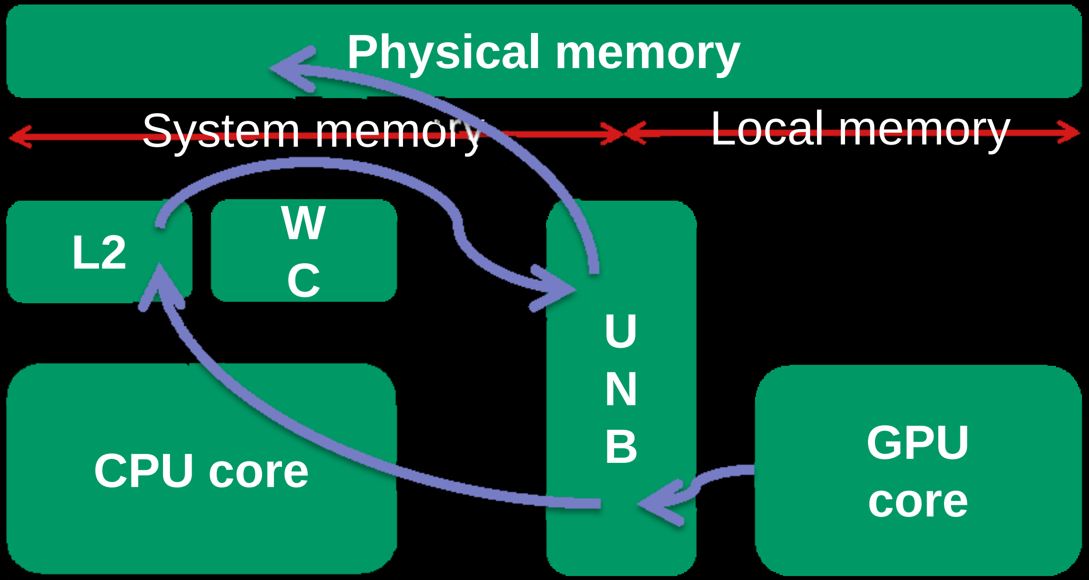
```
glBufferData(GL_PIXEL_UNPACK_BUFFER, size, 0, GL_STREAM_DRAW);
```

GPU ACCESS TO CACHEABLE MEMORY

- GPU accesses to cacheable memory
  - This can be used directly by a kernel or for data upload to the GPU
  - Uses the AMD Fusion Compute Link (ONION), so snoops the cache
  - Reads measured at ~4.5 GB/s and writes at ~5.5 GB/s

  - Often used for sharing data with the CPU

DATAPATH IN THE SYSTEM



SAMPLE CODE

- OpenCL

```
clCreateBuffer(context, CL_MEM_READ_WRITE | CL_MEM_ALLOC_HOST_PTR,
               bufSize, 0, &error);
```

  Then used directly in the kernel

- OpenGL

  Under driver control, typically used for glReadPixels optimization

```
glBufferData(GL_PIXEL_PACK_BUFFER, bufSize, 0, GL_STREAM_READ)
```

OPENGL ZERO COPY EXTENSIONS

- Explicit Zero-Copy is exposed in the GL_AMD_pinned_memory extension

- Introduces a new buffer binding point
  - GL_EXTERNAL_VIRTUAL_MEMORY_BUFFER_AMD

- Pointers passed to glBufferData on this target are considered persistent
  - GPU uses system memory directly and does not make a copy

OPENGL ZERO-COPY – EXAMPLE

- Create and allocate the buffer using the GL_EXTERNAL_VIRTUAL_MEMORY_BUFFER_AMD target

```
void * ptr = malloc(lots);  // This will be our backing store
glGenBuffers(1, &buffer);
glBindBuffer(GL_EXTERNAL_VIRTUAL_MEMORY_BUFFER_AMD, buffer);
glBufferData(GL_EXTERNAL_VIRTUAL_MEMORY_BUFFER_AMD, lots, ptr, GL_STATIC_DRAW);
```

- Stuff data into the buffer however is necessary

```
FILE * f = fopen(“file.bin“, “rb“);  // Be more creative here
fread(ptr, 1, lots, f);
fclose(f);
```

- Bind buffer to desired target and use

```
glBindBuffer(GL_ARRAY_BUFFER, buffer);
glVertexAttribPointer(0, GL_FLOAT, 4, GL_FALSE, NULL);
glEnableVertexAttribArray(0);
// Now the vertex attribute will be read directly from the malloc‘d memory
```

PERFORMANCE (MAY VARY BASED ON SYSTEM/DRIVER)

	LOCAL	UNCACHED	CACHEABLE
GPU READ	17 GB/s	6-12 GB/s	4.5 GB/s
GPU WRITE	12 GB/s	6-12 GB/s	5.5 GB/s
CPU READ	< 1GB/s	< 1GB/s	8-13 GB/s
CPU WRITE	8 GB/s	8-13 GB/s	8-13 GB/s

Disclaimer & Attribution