MANAGING DATA

LEARNING OBJECTIVES

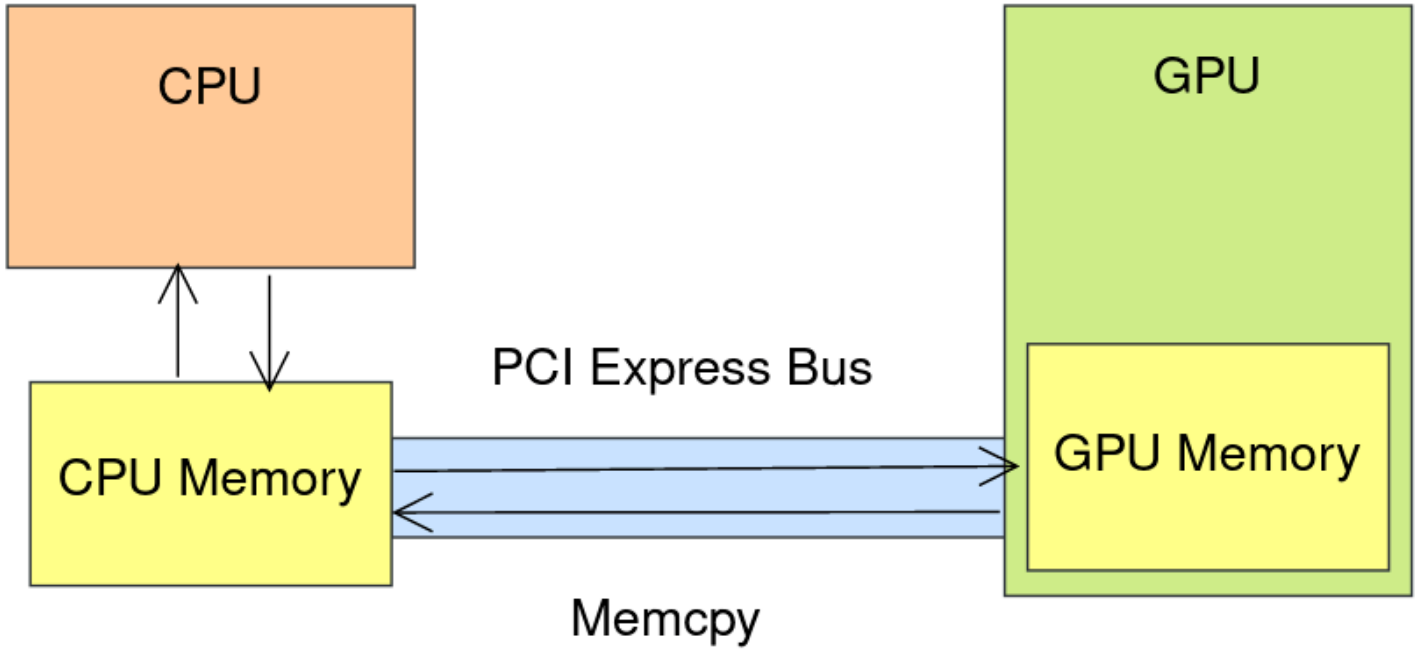
- Learn about the USM data management model
- Learn how to allocate, transfer and free memory using USM.
- Learn how to access data in a kernel function
- Learn about the variations of USM
- Learn about the kinds of USM memory allocations

MEMORY MODELS

- In SYCL there are two models for managing data:
  - The buffer/accessor model.
  - The USM (unified shared memory) model ← *we will focus on this*.
- Which model you choose can have an effect on how you enqueue kernel functions.

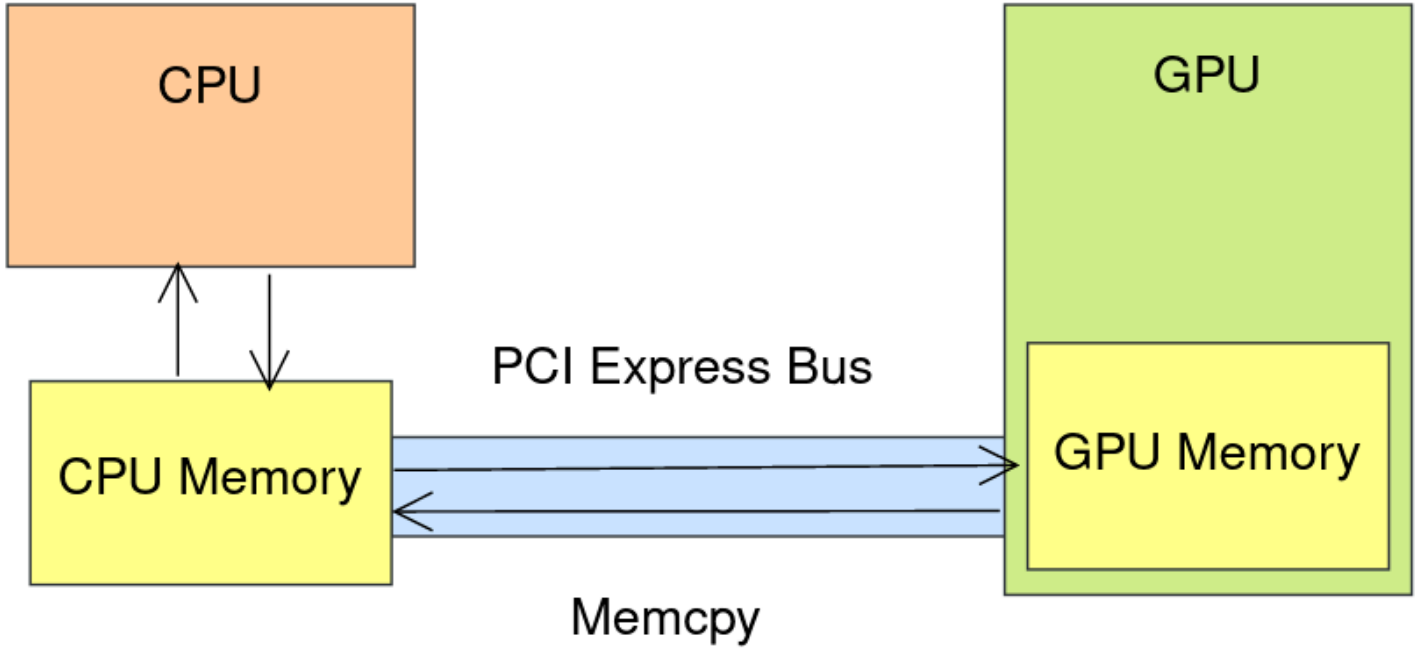
CPU AND GPU MEMORY

- A GPU has its own memory, separate to CPU memory.
- In order for the GPU to use memory from the CPU, the following actions must take place (either explicitly or implicitly):
    - Memory allocation on the GPU.
    - Data migration from the CPU to the allocation on the GPU.
    - Some computation on the GPU.
    - Migration of the result back to the CPU.

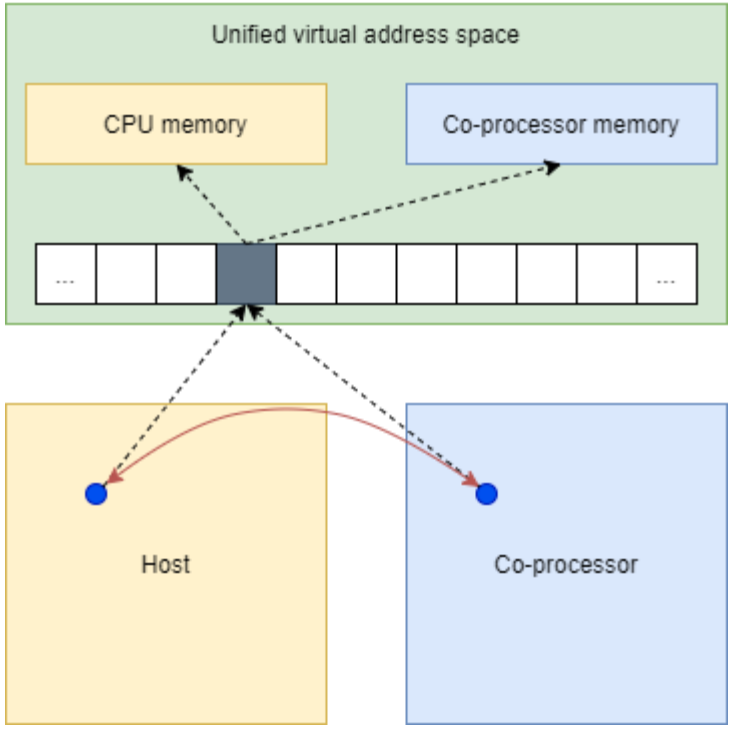


CPU AND GPU MEMORY

- Memory transfers between CPU and GPU are a bottleneck.
- We want to minimize these transfers, when possible.

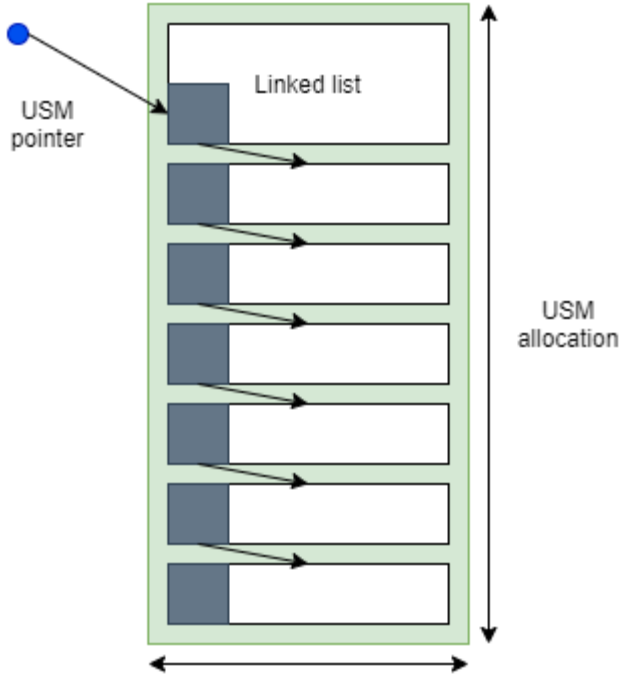


UNIFIED VIRTUAL ADDRESS SPACE



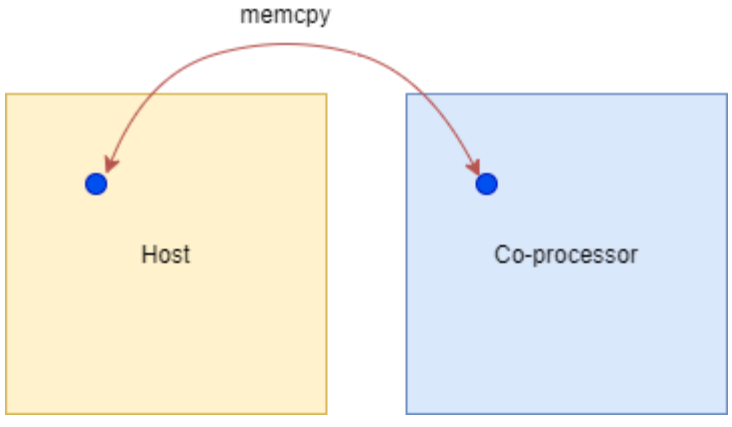
- USM memory allocations return pointers which are consistent between the host application and kernel functions on a device
- Representing data between the host and device(s) does not require creating accessors
- Pointer-based API more familiar to C or C++ programmers

POINTER BASED STRUCTURES



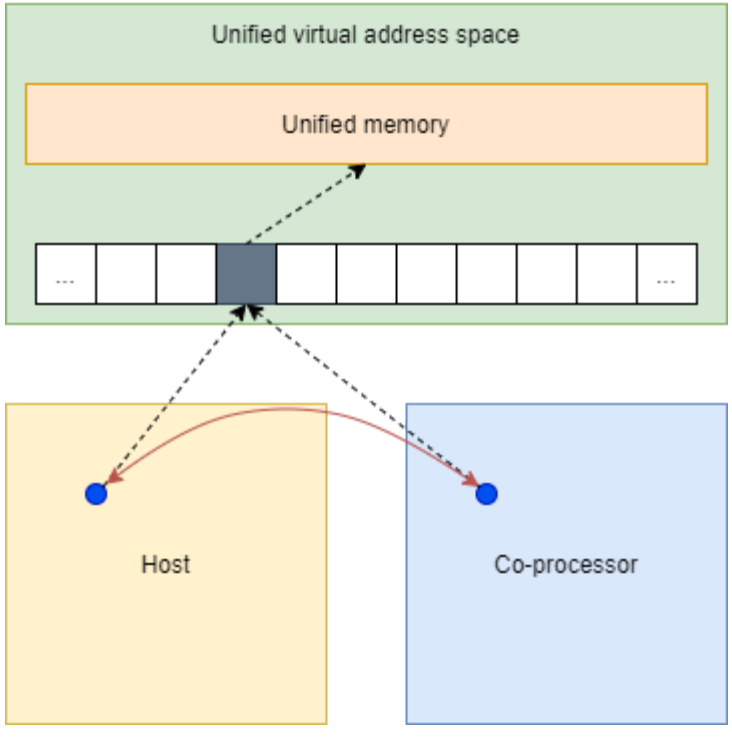
- Data is moved between the host and device(s) in a span of memory in bytes rather than a buffer of a specific type
- Pointers within that region of memory can freely point to any other address in that region
- Easier to port existing C or C++ code to use SYCL

EXPLICIT MEMORY MANAGEMENT



- Memory is allocated and data is moved using explicit routines
- Moving data between the host and device(s) does not require accessors or submitting command groups
- The SYCL runtime will not perform any data dependency analysis, dependencies between commands must be managed manually

SHARED MEMORY ALLOCATIONS



- Some platforms will support variants of USM where memory allocations share the same memory region between the host and device(s)
- No explicit routines are required to move the data between the host and device(s)

USM ALLOCATION TYPES

USM has three different kinds of memory allocation

- A **host** allocation is allocated in host memory
- A **device** allocation is allocation in device memory
- A **shared** allocation is allocated in shared memory and can migrate back and forth

USM VARIANTS

USM has four variants which a platform can support with varying levels of support

	Explicit USM (minimum)	Restricted USM (optional)	Concurrent USM (optional)	System USM (optional)
Consistent pointers	✓	✓	✓	✓
Pointer-based structures	✓	✓	✓	✓
Explicit data movement	✓	✓	✓	✓
Shared memory allocations	✗	✓	✓	✓
Concurrent access	✗	✗	✓	✓
System allocations	✗	✗	✗	✓

QUERYING FOR SUPPORT

Each SYCL platform and its device(s) will support different variants of USM and different kinds of memory allocation

```
if (dev.has(sycl::aspect::usm_device_allocations))
```

USING USM - MALLOC DEVICE

```
// Allocate memory on device
T *device_ptr = sycl::malloc_device<T>(n, myQueue);

// Copy data to device
myQueue.memcpy(device_ptr, cpu_ptr, n * sizeof(T));

// ...
// Do some computation on device
// ...

// Copy data back to CPU
myQueue.memcpy(result_ptr, device_ptr, n * sizeof(T)).wait();

// Free allocated data
sycl::free(device_ptr, myQueue);
```

- It is important to free memory after it has been used to avoid memory leaks.

USING USM - MALLOC SHARED

```
// Allocate shared memory
T *shared_ptr = sycl::malloc_shared<T>(n, myQueue);

// Shared memory can be accessed on host as well as device
for (auto i = 0; i < n; ++i)
  shared_ptr[i] = i;

// ...
// Do some computation on device
// ...

// Free allocated data
sycl::free(shared_ptr, myQueue);
```

- Shared memory is accessible on host and device.
- Performance of shared memory accesses may be poor depending on platform.

QUESTIONS

EXERCISE

Code_Exercises/Managing_Data/source

Implement a SYCL application that adds two variables and returns the result. Use USM to manage memory.