INTRODUCTION TO USM

LEARNING OBJECTIVES

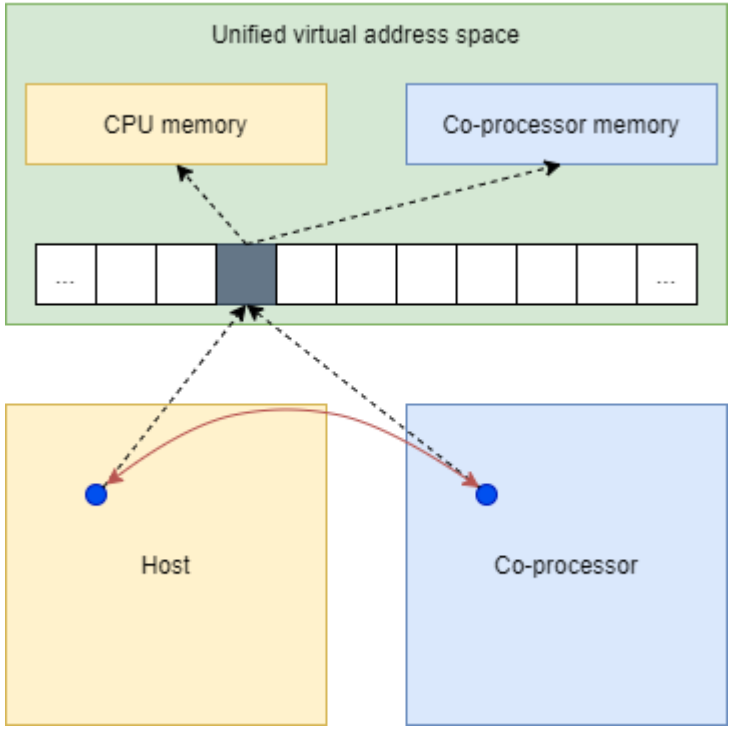
- Learn about the USM data management model
- Learn about the benefits of USM and when to use it
- Learn about the variations of USM
- Learn about the kinds of USM memory allocations

WHAT IS USM?

Unified shared memory (USM) provides an alternative pointer-based data management model to the accessor-buffer model

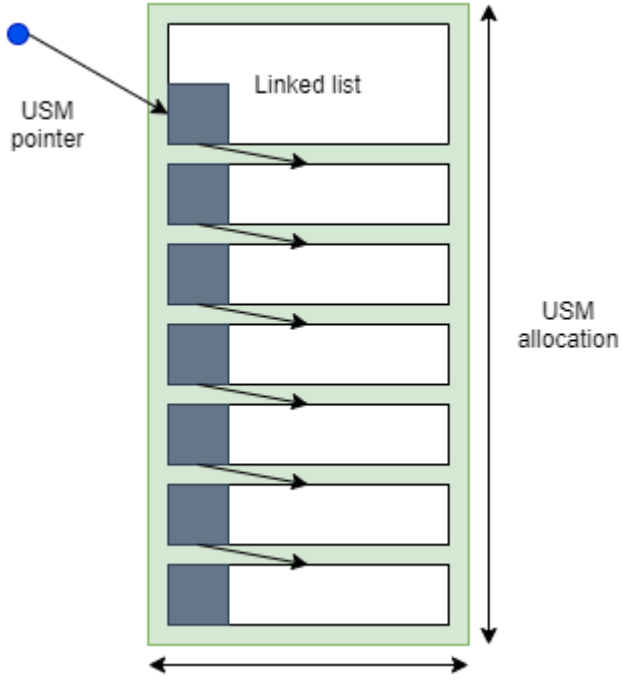
- Unified virtual address space
- Pointer-based structures
- Explicit memory management
- Shared memory allocations

UNIFIED VIRTUAL ADDRESS SPACE



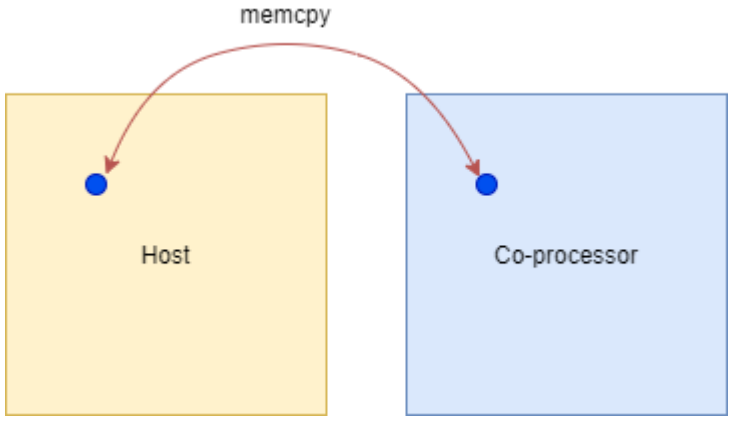
- USM memory allocations return pointers which are consistent between the host application and kernel functions on a device
- Representing data between the host and device(s) does not require creating accessors
- Pointer-based API more familiar to C or C++ programmers

POINTER BASED STRUCTURES



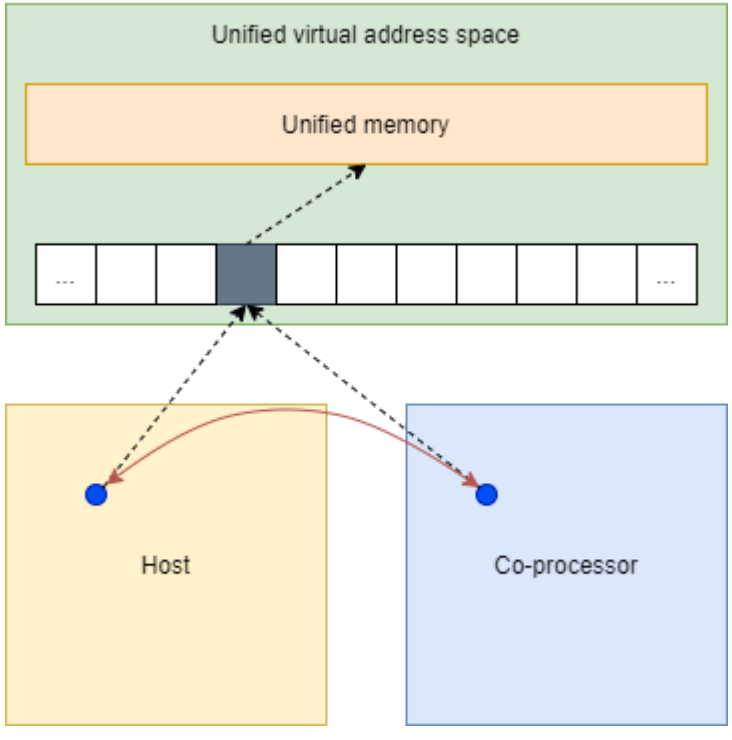
- Data is moved between the host and device(s) in a span of memory in bytes rather than a buffer of a specific type
- Pointers within that region of memory can freely point to any other address in that region
- Easier to port existing C or C++ code to use SYCL

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))
```

BUFFER/ACCESSOR VS USM

So when should you use the accessor/buffer mode and when should you use the USM model?

- The **buffer/accessor** provides guaranteed consistency and automatically manages dependencies
    - Recommended when you need to iterate over or prototype your application
    - Recommended for maximum performance portability
    - Provides a lot of information to the SYCL runtime which can perform many optimizations automatically
- The **USM** model provides a lower-level pointer-based solution with fine grained control
    - Recommended when porting existing C++ applications or using pointer-based structures
    - User has responsibility for certain aspects relevant for performance (e.g. data transfers or data prefetches)

QUESTIONS

EXERCISE

Code_Exercises/Exercise_7_USM_Selector/source

Implement a device selector that chooses a device in your system which supports explicit USM and device allocations