Taking Memory Management to the Next Level: Unified Shared Memory in Action

Michal Mrozek, Ben Ashbaugh, James Brodman

IWOCL 2020

Notices & Disclaimers

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors.

Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. For more complete information visit www.intel.com/benchmarks.

Performance results are based on testing as of dates shown in configurations and may not reflect all publicly available updates. See backup for configuration details. No product or component can be absolutely secure.

Your costs and results may vary.

Intel technologies may require enabled hardware, software or service activation.

© Intel Corporation. Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Other names and brands may be claimed as the property of others.

OpenCL and the OpenCL logo are trademarks of Apple Inc. used by permission by Khronos.

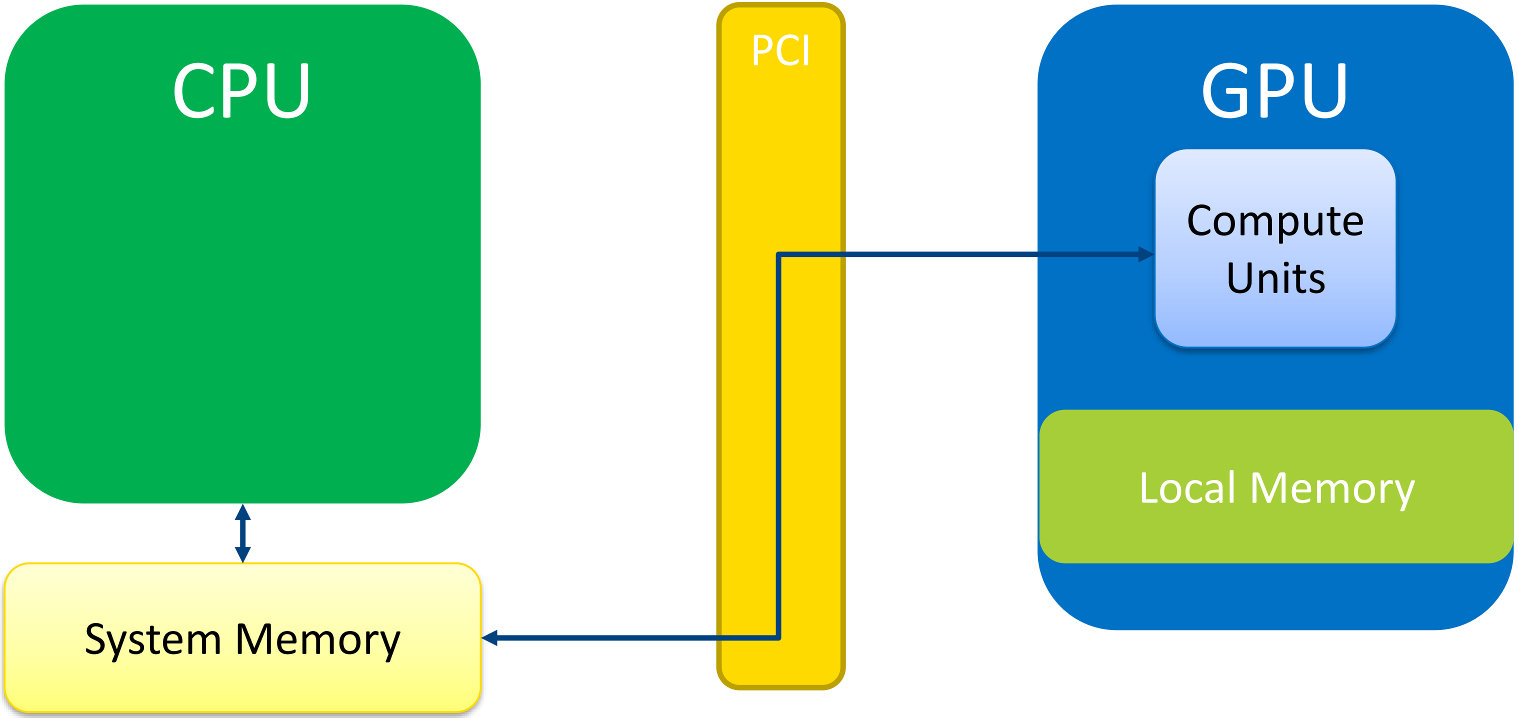
** Other names and brands may be claimed as the property of others.*

Agenda

- Let’s look at Shared Virtual Memory
- Introducing Unified Shared Memory
- Unified Shared Memory in DPC++
- Future Plans and Call To Action

Let’s look at Shared Virtual Memory (SVM) !

SVM allocations and devices with local memory



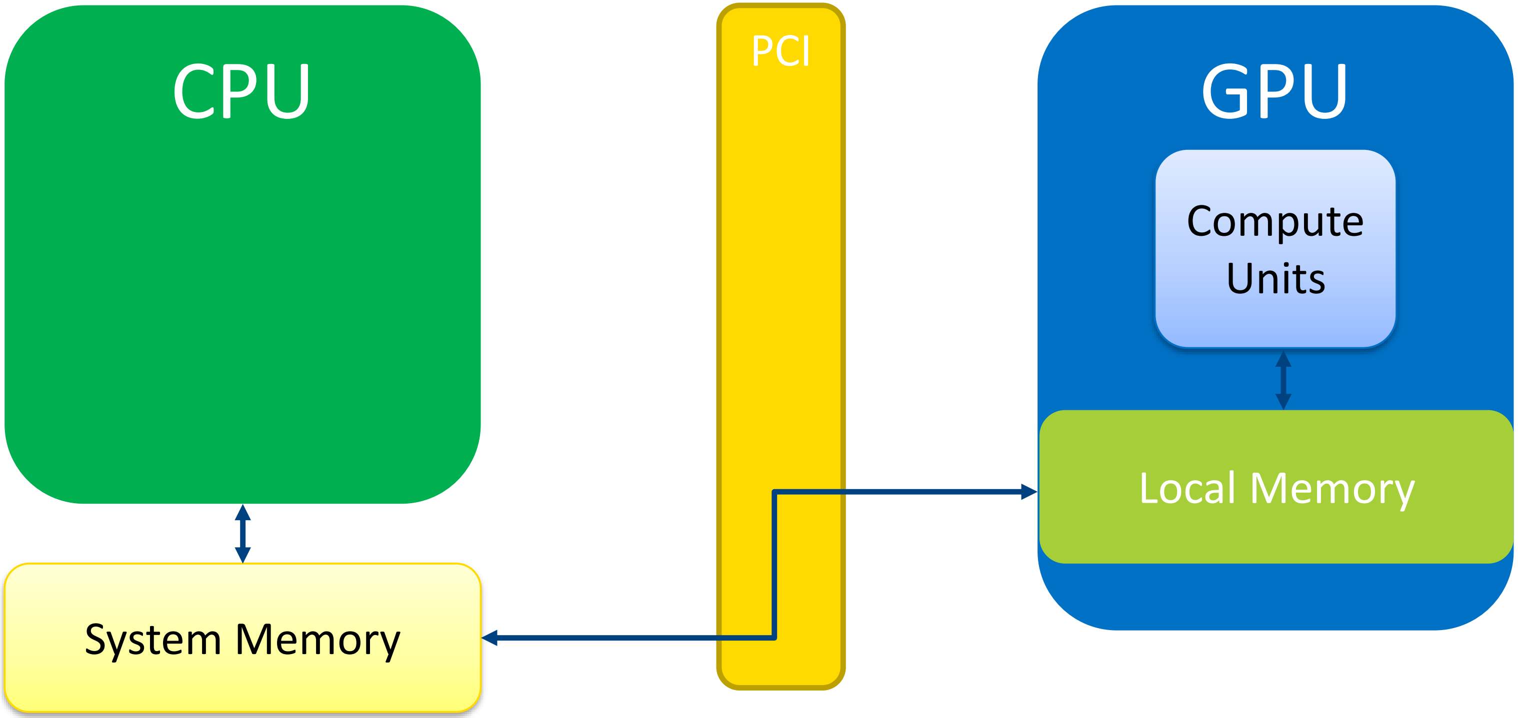
Good:

- Direct access to System Memory

Bad:

- Low Bandwidth due to PCI access

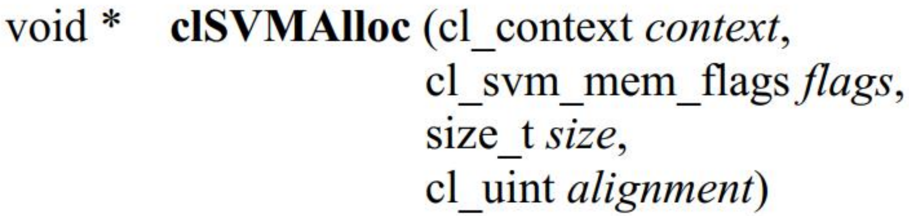
SVM allocations and devices with local memory



Good:
- Fast access to local memory

Bad:
- Requires transfer to system when host wants to use memory

clSvmAlloc and devices with local memory

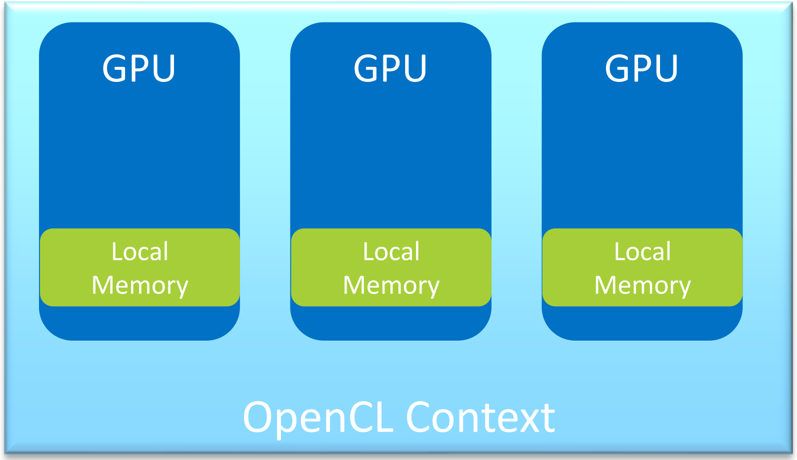


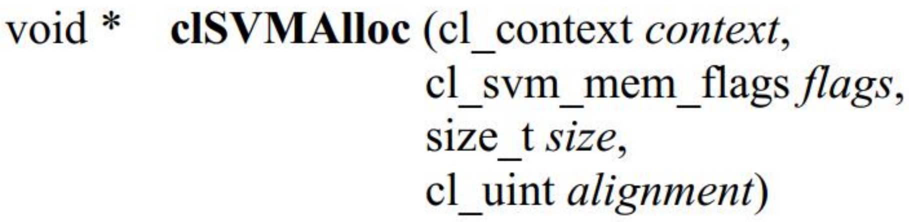
```
void *    clSVMAlloc (cl_context context,
                      cl_svm_mem_flags flags,
                      size_t size,
                      cl_uint alignment)
```

Problem (1) – Memory placement

- Where to place memory ?
  - Host ?
    - What if host never access?
  - Device ?
    - What if caller wants host based allocation?

SVM and multiple devices





```
void *    clSVMAlloc (cl_context context,
                      cl_svm_mem_flags flags,
                      size_t size,
                      cl_uint alignment)
```

Problem (2) – Multi device memory placement

- Where to place memory ?
  - First Device Used ?
  - All devices ?
  - System memory ?
  - Automatic migration ?
  - How to synchronize contents with multi local memory placements ?

Driver heuristics are bad!

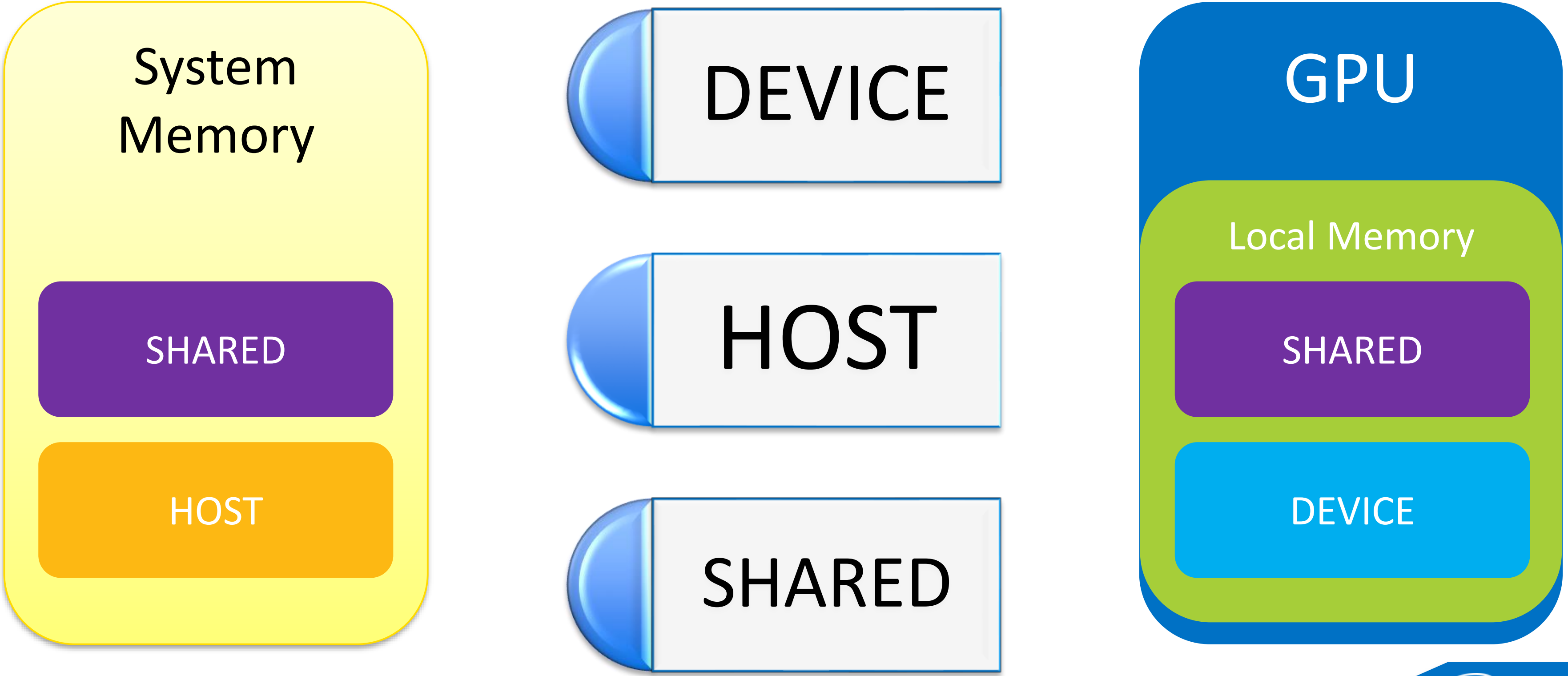
OpenCL 2.0 SVM: Programmer Convenience

(NOT Performance!)

	Coarse Grain	Fine Grain
Buffer	clSVMAlloc/Free, Pointer Representation, Good! Address Equivalence, Specify All Allocations, No Concurrent Access, Map/Unmap	clSVMAlloc/Free, Pointer Representation, Good! Address Equivalence, Specify All Allocations, Concurrent Access (Fine Grain) No Map/Unmap Good!
System	(N/A) Most Implementations Are Here ☹	malloc/free, Good! Pointer Representation, Good! Address Equivalence, Access Any Allocation, Good! Concurrent Access (Fine Grain) No Map/Unmap Good!

Introducing Unified Shared Memory (USM)

Introducing 3 new memory types



System Memory

SHARED

HOST

DEVICE

HOST

SHARED

GPU

Local Memory

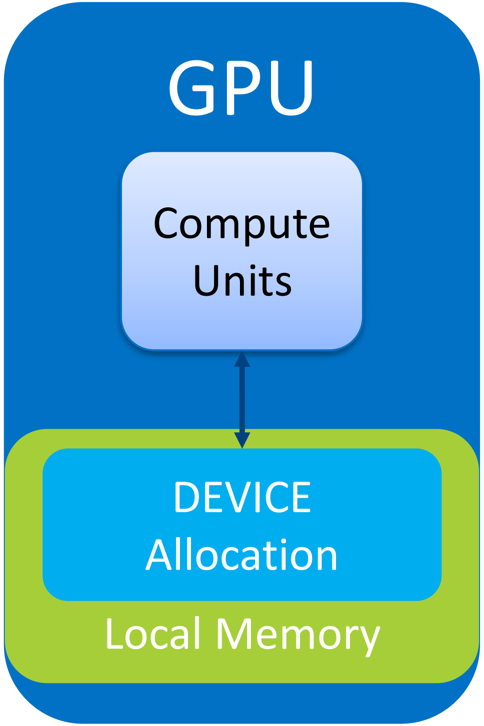
SHARED

DEVICE

Device Allocations: Performance

- No Host access
- No migration
- Available only in one device
- No Map/Unmap
- Best Performance possible
- Pointer representation

```
void* clDeviceMemAllocINTEL(
    cl_context context,
    cl_device_id device,
    const cl_mem_properties_intel* properties,
    size_t size,
    cl_uint alignment,
    cl_int* errcode_ret);
```

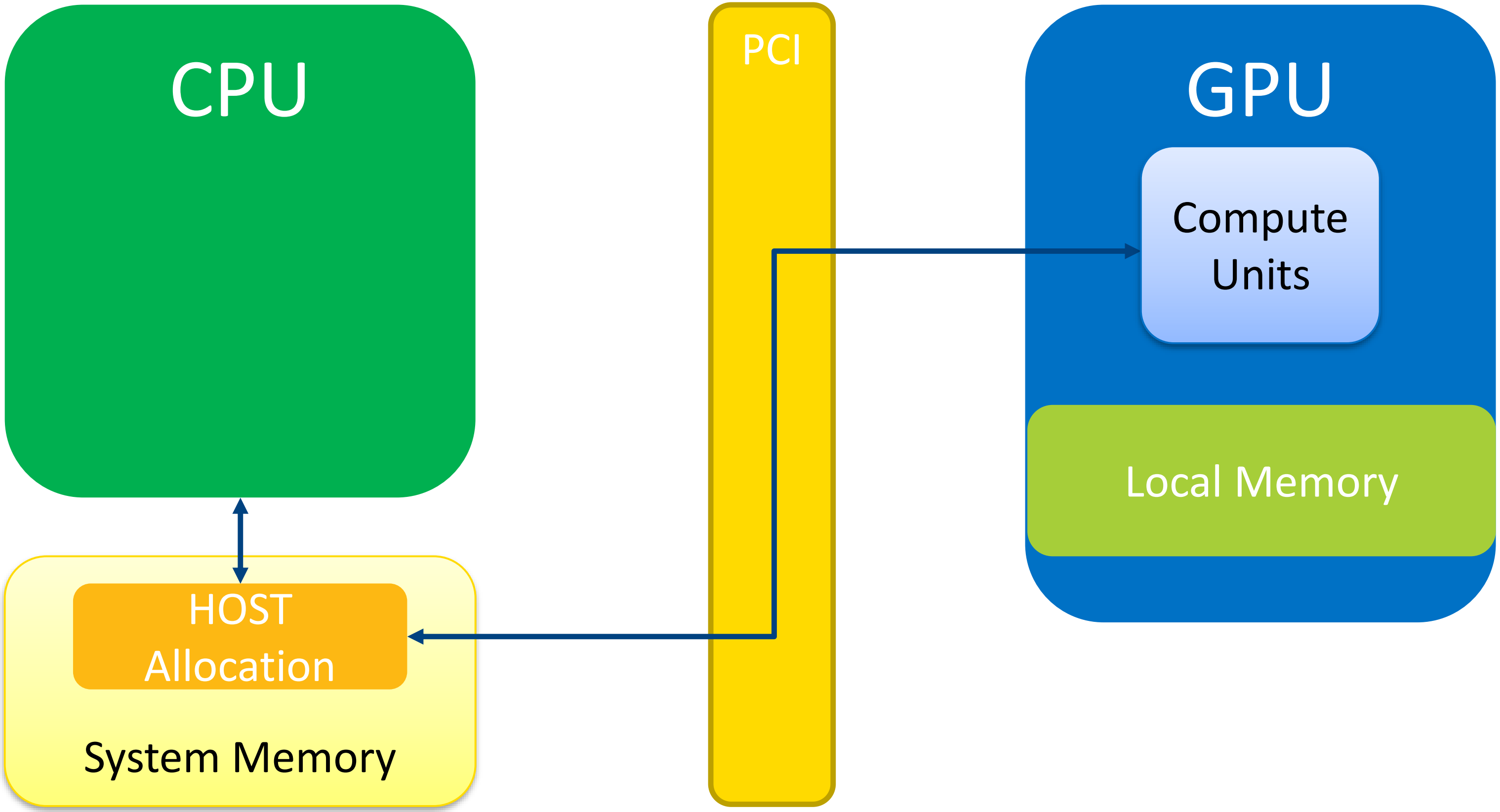


Host Allocations: Zero Copy Sharing (no Migration)

- Accessible by the Host
- Placed in Host memory, doesn’t migrate to local memory
- Accessible by all devices in the context
- No Map/Unmap
- Useful as input / output buffers, Pinned Memory or Staging Allocation
- Possible oversubscription
- Pointer representation
- Address equivalence

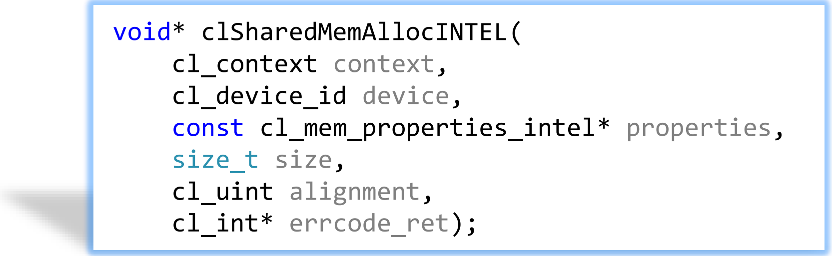
```
void* clHostMemAllocINTEL(
    cl_context context,
    const cl_mem_properties_intel* properties,
    size_t size,
    cl_uint alignment,
    cl_int* errcode_ret);
```

Host Allocation: Direct GPU access to System Memory



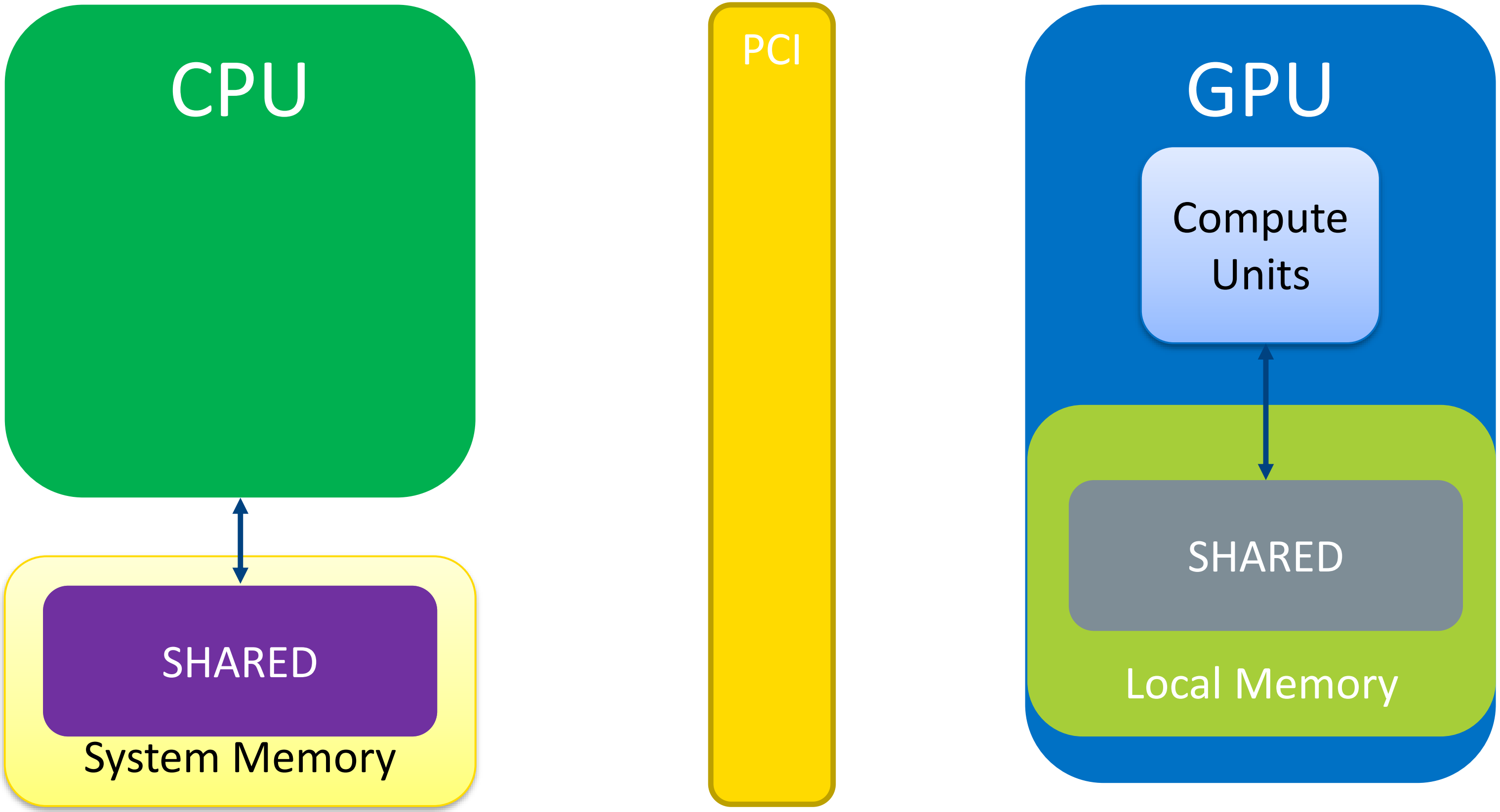
Shared Allocations: Programmer **Convenience**

- Shared Host-Device Ownership
- No Map/Unmap
- Automatic Migration Between Host and Device
- Accessible by all devices in context, passed device show optional initial placement
- Trades control for convenience
- Pointer representation
- Address equivalence

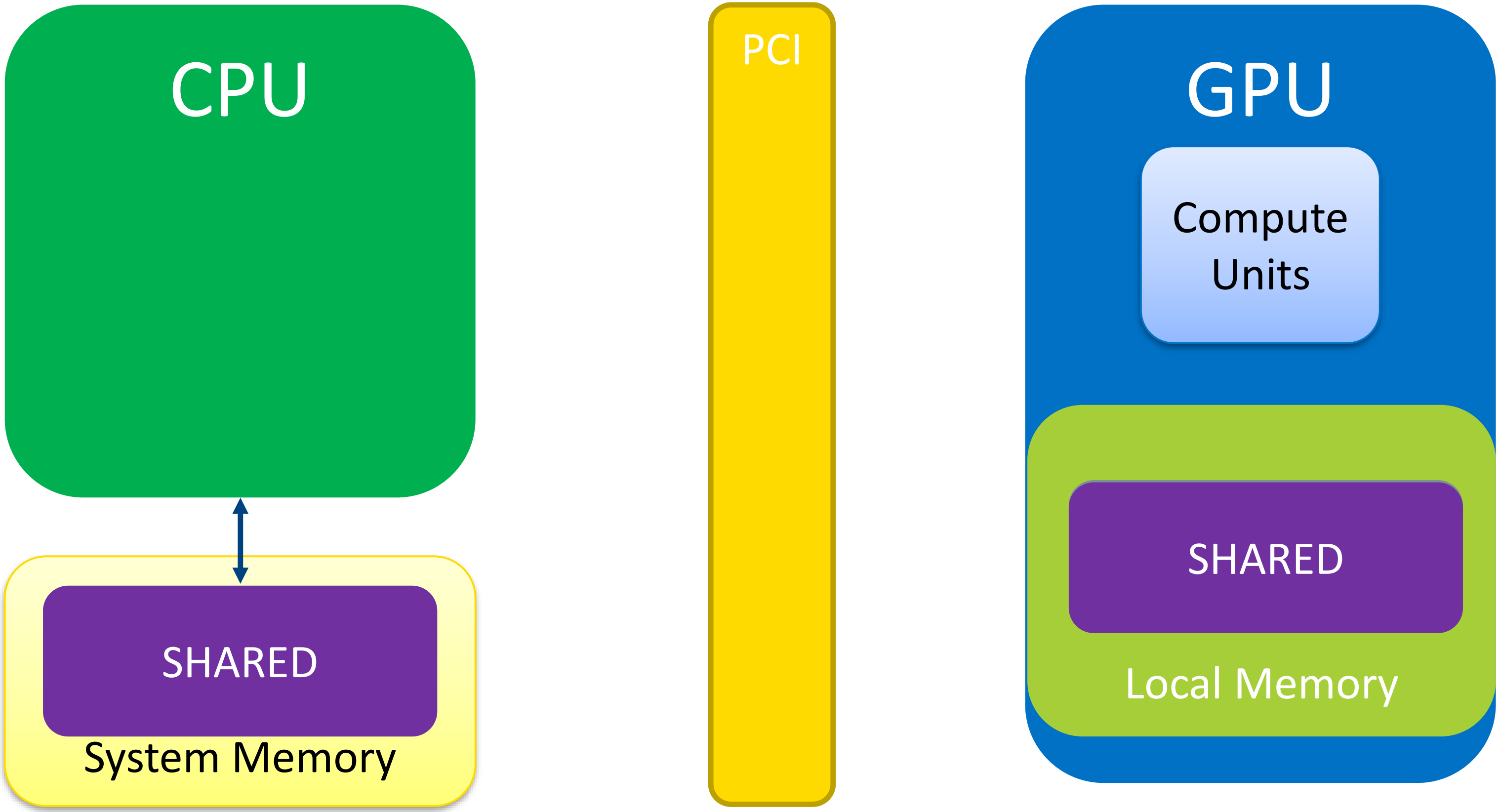


```
void* clSharedMemAllocINTEL(
    cl_context context,
    cl_device_id device,
    const cl_mem_properties_intel* properties,
    size_t size,
    cl_uint alignment,
    cl_int* errcode_ret);
```

Shared allocation – automatic migration to GPU



Shared allocation – automatic migration to CPU



Freeing the memory

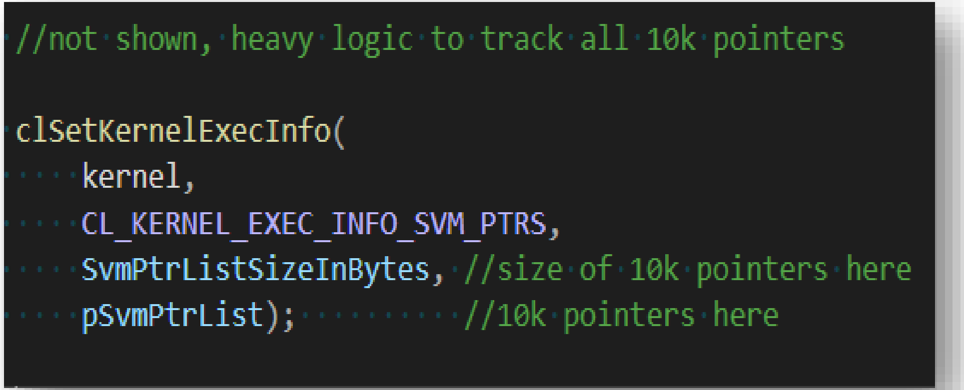
- Blocking version introduced for convenience,
    - Waits for completion of all associated submissions
    - Useful when application do not want to track what is used where
- Non-Blocking version requires synchronization from application

```
cl_int clMemFreeINTEL(
    cl_context context,
    void* ptr);

cl_int clMemBlockingFreeINTEL(
    cl_context context,
    void* ptr);
```

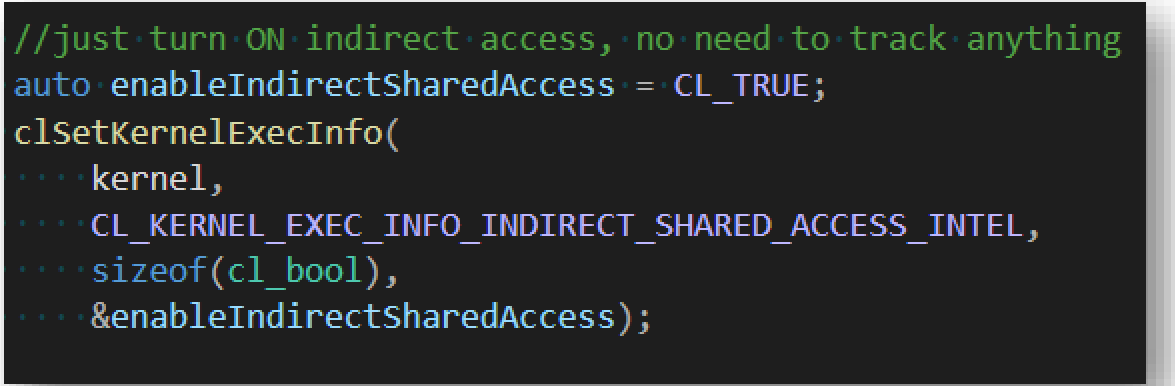
Indirect Access

- Automatic specification of indirect usage per kernel
- No need to track all allocation and pass them
- Saves CPU clocks (no need to validate input)
- Each memory type has its own toggle



```
//not shown, heavy logic to track all 10k pointers

clSetKernelExecInfo(
    kernel,
    CL_KERNEL_EXEC_INFO_SVM_PTRS,
    SvmPtrListSizeInBytes, //size of 10k pointers here
    pSvmPtrList);          //10k pointers here
```



```
//just turn ON indirect access, no need to track anything
auto enableIndirectSharedAccess = CL_TRUE;
clSetKernelExecInfo(
    kernel,
    CL_KERNEL_EXEC_INFO_INDIRECT_SHARED_ACCESS_INTEL,
    sizeof(cl_bool),
    &enableIndirectSharedAccess);
```

Retrieving information from USM pointers

- Currently supported properties:
  - Allocation type
  - Base pointer
  - Allocation size
  - Associated device
  - Allocation flags
- Allows easy pointer integration to existing code bases

```
cl_int clGetMemAllocInfoINTEL(
    cl_context context,
    const void* ptr,
    cl_mem_info_intel param_name,
    size_t param_value_size,
    void* param_value,
    size_t* param_value_size_ret);
```

Unified Shared Memory in DPC++

Unified Shared Memory in DPC++

USM is supported as a SYCL extension in the DPC++ compiler:

DPC++ = C++ and SYCL and **Extensions**

USM provides a pointer-based alternative to SYCL buffers:

- Simpler and more concise code for common patterns
- Easier integration into C++ code bases
- Greater control over memory ownership and accessibility

USM Code Walk-Through

```
// setup
ordered_queue q{ platform::get_platforms()[pi].get_devices()[di] };

auto d = q.get_device();
auto c = q.get_context();

auto s_src = (uint32_t*)malloc_shared(gwx * sizeof(uint32_t), d, c);
auto s_dst = (uint32_t*)malloc_shared(gwx * sizeof(uint32_t), d, c);
```

USM allocations are made against a SYCL ***context***

- Shared and Device USM allocations may also have an associated SYCL ***device***

USM supports three forms of allocation

- **`malloc`**-like (this example), templated **`malloc`**, **`std::allocator`**-like

USM Code Walk-Through

```
// initialize memory
for( size_t i = 0; i < gwx; i++ )
    s_src[i] = (uint32_t)i;
memset(s_dst, 0, gwx * sizeof(uint32_t));
```

For Shared and Host USM allocations: simply access on the host!

- No need for mapping, unmapping, or accessors

For Device USM allocations: must copy to host-accessible allocations

USM Code Walk-Through

```
// execute a kernel to copy buffers
q.parallel_for(range<1>{gwx}, [=](id<1> id) {
    s_dst[id] = s_src[id];
});

q.wait();
```

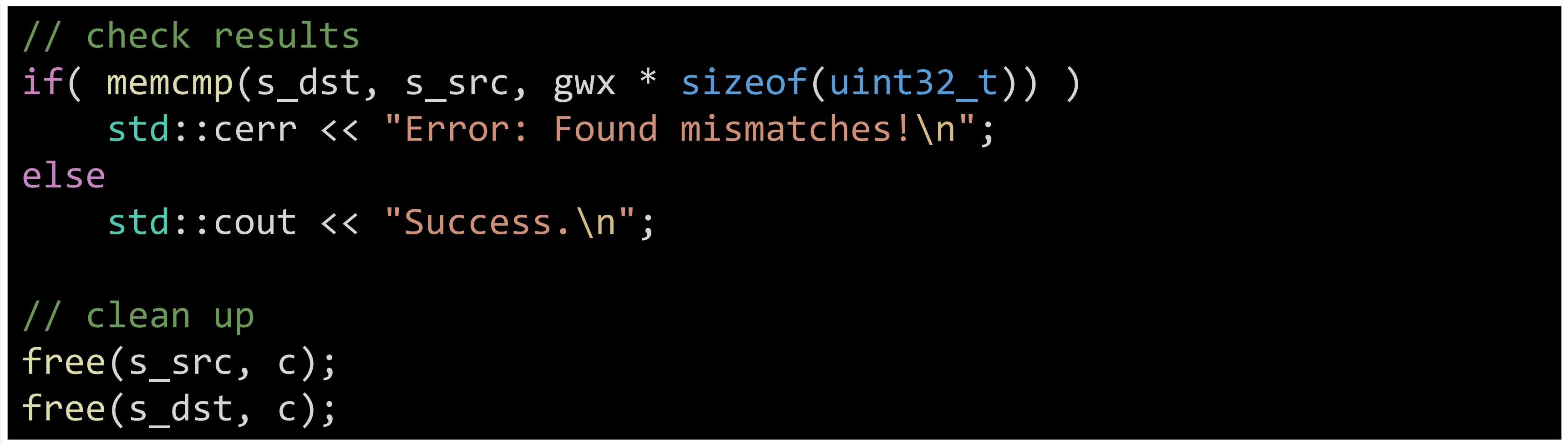
Kernel lambda can capture and use USM pointers directly!

- No need for accessors!

New mechanisms to express dependencies between queue operations:

- **`depends_on`**: define explicit dependencies between queue operations
- **`ordered_queue`** type (this example): implicit in-order execution

USM Code Walk-Through



```
// check results
if( memcmp(s_dst, s_src, gwx * sizeof(uint32_t)) )
    std::cerr << "Error: Found mismatches!\n";
else
    std::cout << "Success.\n";

// clean up
free(s_src, c);
free(s_dst, c);
```

Checking results and freeing allocations is straightforward

- Free function requires the same SYCL context used for allocation

Complete Example

```
// setup
ordered_queue q{ platform::get_platforms()[pi].get_devices()[di] };
auto c = q.get_context();
auto d = q.get_device();
auto s_src = (uint32_t*)malloc_shared(gwx * sizeof(uint32_t), d, c);
auto s_dst = (uint32_t*)malloc_shared(gwx * sizeof(uint32_t), d, c);

// initialize memory
for( size_t i = 0; i < gwx; i++ )
    s_src[i] = (uint32_t)i;
memset(s_dst, 0, gwx * sizeof(uint32_t));

// execute a kernel to copy buffers
q.parallel_for(range<1>{gwx}, [=](id<1> id) {
    s_dst[id] = s_src[id];
});
q.wait();

// check results
if( memcmp(s_dst, s_src, gwx * sizeof(uint32_t)) )
    std::cerr << "Error: Found mismatches!\n";
else
    std::cout << "Success.\n";

// clean up
free(s_src, c);
free(s_dst, c);
```

Future Plans and Call to Action

Future Plans and Call to Action

We recommend including Unified Shared Memory in future standards:

- For both OpenCL and SYCL
- We will continue to develop USM in DPC++

Try USM!

- Your feedback is valuable before standardization!
- If you find USM useful, encourage other implementations to support USM!

Thank you!

Useful Links:

USM Specifications:

- https://github.com/intel/llvm/blob/sycl/sycl/doc/extensions/USM/USM.adoc
- https://github.com/intel/llvm/blob/sycl/sycl/doc/extensions/USM/cl_intel_unified_shared_memory.asciidoc

USM Implementations:

- https://software.intel.com/en-us/oneapi/base-kit
- https://github.com/intel/compute-runtime

USM Samples:

- https://github.com/intel/compute-samples
- https://github.com/bashbaug/SimpleOpenCLSamples/tree/master/samples/usm
- https://github.com/bashbaug/simple-sycl-samples/tree/master/samples/dpcpp/usm