ALCF HANDS-ON HPC WORKSHOP

Argonne NATIONAL LABORATORY

OPENMP EFFICIENT CODE PATTERNS AND PORTING TIPS

YE LUO
Computational Science Division

Sep 23rd 2025, Virtual.

OUTLINE

- Choose between directives and APIs in OpenMP
- Write performant code
- OpenMP and SYCL interoperability

CODING STYLE: DIRECTIVES VS API

PORTABILITY AT THE SOURCE CODE

Choosing OpenMP programing style carefully

OPENMP DIRECTIVE

```
#pragma omp target enter data
map(alloc: a[:100])
```

- Prons:
  - No side effect when turned off
  - Fall back to host for debugging OMP_TARGET_OFFLOAD=disabled
- Cons:
  - Less verbose

OPENMP API

```
int * a_dev =
omp_target_alloc(omp_get_default_d
evice(), 100);
```

- Prons:
  - Explicit device control
- Cons:
  - Need #ifdef _OPENMP
  - Complicated fallback logic

DO NOT USE OPENMP LIKE CUDA

Directives instead of kernels

```
const int N=100;
int array[N];
#pragma omp target \
      map(from, a[:N])
for(int i=0; i<N; i++)
   array[i] = I;
```

Do this

```
const int N=100;
int array[N];
int* array_d = (int*) omp_target_alloc(0, N*sizeof(int));
#pragma omp target is_ptr(array_d)
for(int i=0; i<N; i++)
   array_d[i] = I;
omp_target_memcpy(array, array_d, 0);
```

Don’t do this

WRITE PERFORMANT CODE BY
1. IMPROVING DATA LOCALITY
2. LEVERAGING IMPLICIT ASYNCHRONOUS COMPUTATION

DATA MOVEMENT PLAYS A KEY ROLE

- Data locality is the top priority
  - Interconnect is slower than memory. We write performant MPI code
  - Memory is slower than cache. We implement cache friendly algorithms.
  - CPU-GPU bus is slower than GPU memory. We need to first worry about data transfer.
- GPU 101 teaches kernel programming. It is not the key.
- Managing data movement requires understanding the whole algorithm and implementation. This needs domain expert knowledge.

OPENMP OFFLOAD RUNTIME MOTIONS

Poor performance if every step is synchronous

- Allocate array on device (very slow)
- Transfer H2D (slow)
- Launch the kernel
- Transfer D2H (slow)
- Deallocate array on device (very slow)

```
// simple case
#pragma omp target \
   map(array[:100])
for(int i ...)
{ // operations on array }
```

PRE-ARRANGE MEMORY ALLOCATION

Move beyond textbook example

- Accelerator memory resource allocation/deallocation is orders of magnitude slower than that on the host.
- These operations may also block asynchronous execution.
- Allocate array on host pinned memory to make the transfer asynchronous.

```
// optimized case
// pre-arrange allocation
#pragma omp target enter data \
  map(alloc: array[:100])
…
// use always to enforce transfer
#pragma omp target map(always, array[:100])
for(int i ...) { // operations on array }

// free memory
#pragma omp target exit data \
  map(delete: array[:100])
```

IMPLICIT ASYNCHRONOUS DISPATCH

Using queues/streams

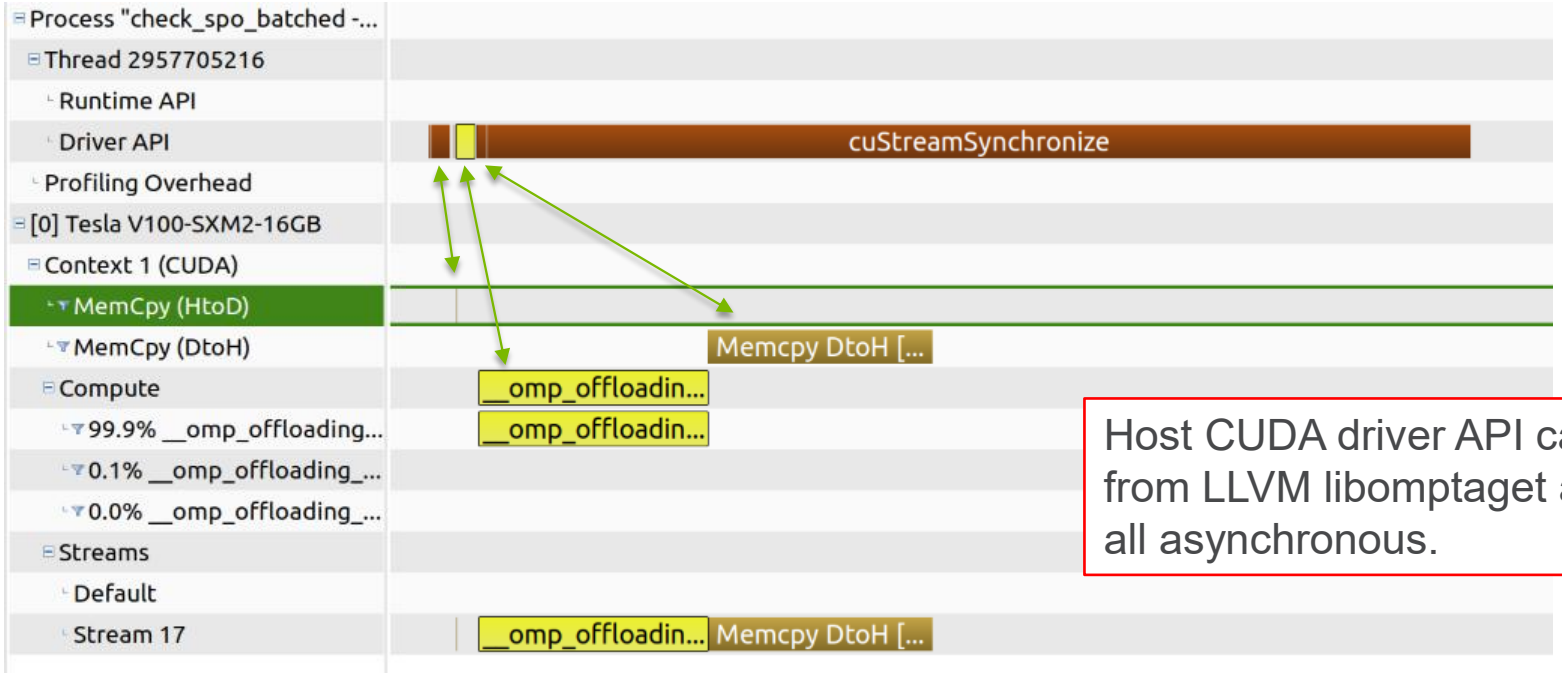
- LLVM Clang OpenMP runtime enqueue non-blocking H2D, kernel, D2H with only one synchronization in the end.
  - Async transfer H2D
  - Async enqueue the kernel
  - Async transfer D2H
  - Synchronization if ‘nowait’ is not used
- Intel implements similar optimization using L0 immediate command list

```
// optimized case
// pre-arrange allocation
#pragma omp target enter data \
  map(alloc: array[:100])
…
// use always to enforce transfer
#pragma omp target map(always, array[:100])
for(int i ...) { // operations on array }

// free memory
#pragma omp target exit data \
  map(delete: array[:100])
```

IMPLICIT ASYNCHRONOUS DISPATCH (CONT)

Maximize asynchronous calls within one target region



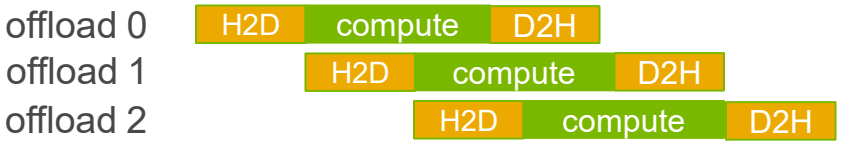
Host CUDA driver API calls from LLVM libomptaget are all asynchronous.

CONCURRENT EXECUTION AND TRANSFER

Overlapping computation and data transfer

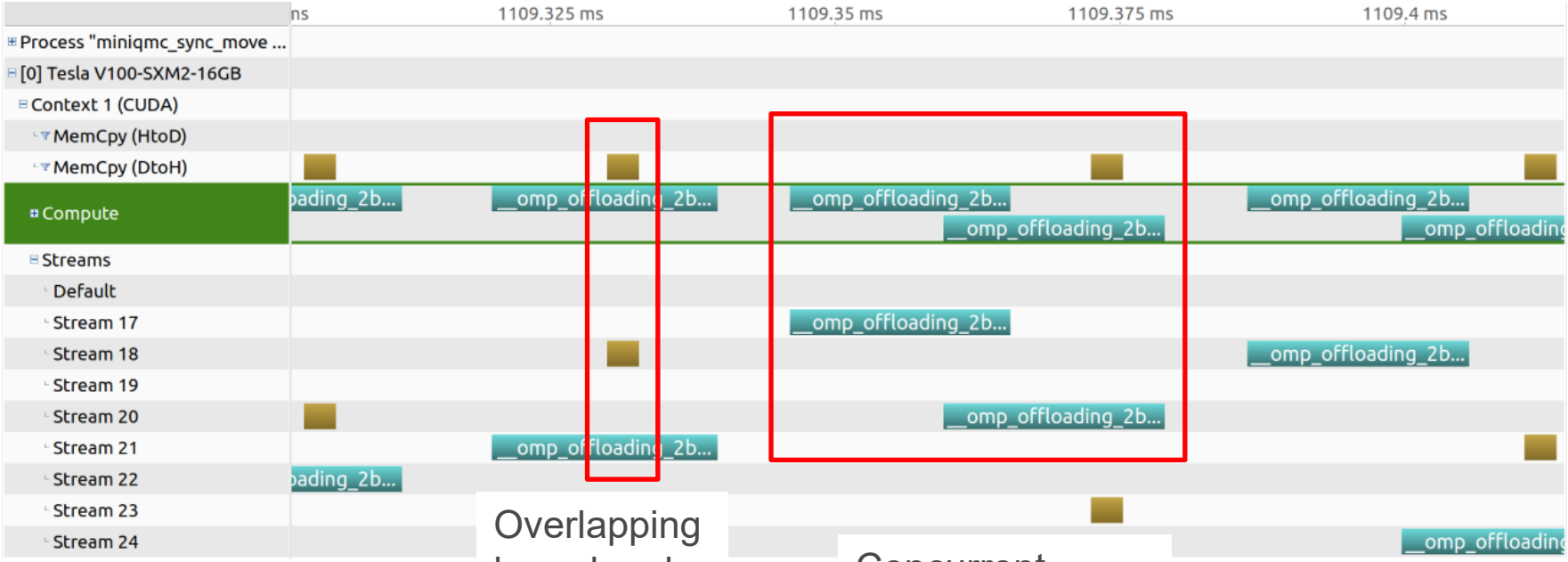
- Need multiple concurrent target regions
- Kernel execution from one target region may overlap with kernel execution or data transfer from another target region

```
#pragma omp parallel for
for (int iw …)
{
    int* array = all_arrays[iw].data();
    #pragma omp target \
      map(always, tofrom: array[:100])
    for(int i ...)
    { // operations on array }
}
```



CONCURRENT EXECUTION (CONT)

From multiple OpenMP threads, in miniQMC



USING PINNED MEMORY

Enable true asynchronous transfer

- Keep CPU cores submitting work to GPUs.
- Method 1. Pin host memory using vendor APIs like **cudaHostRegister**, **sycl::ext::oneapi::experimental::prepare_for_device_copy**
- Method 2. allocated pinned memory using vendor APIs like **cudaMallocHost**, **sycl:: malloc_host/aligned_alloc_host<T>**.
- Method 3. Use OpenMP extension llvm/**omp_target_alloc_host** (supported by icx/icpx)

SYCL AND OPENMP INTEROPERABILITY

SYCL AND OPENMP INTEROPERABILITY

For better serving both world

- Use OpenMP to generate L0 device and context.

```
#pragma omp interop device(id) init(prefer_type("level_zero"), targetsync : interop)
auto hContext = omp_get_interop_ptr(interop, omp_ipr_device_context, &err);
auto hDevice = omp_get_interop_ptr(interop, omp_ipr_device, &err);
```

- Build SYCL objects

```
auto sycl_device = sycl::make_device<sycl::backend::ext_oneapi_level_zero>(
        reinterpret_cast<const sycl::backend_input_t<sycl::backend::ext_oneapi_level_zero,
sycl::device>>(hDevice));
auto sycl_context = sycl::make_context<sycl::backend::ext_oneapi_level_zero>(
        {reinterpret_cast<const sycl::detail::interop<sycl::backend::ext_oneapi_level_zero,
              sycl::context>::type>(hContext),
         {sycl_device}, sycl::ext::oneapi::level_zero::ownership::keep})
auto queue = std::make_unique<sycl::queue>(sycl_context, sycl_device,
                                           sycl::property::queue::in_order());
```

GPU MEMORY QUERY

- get_info<sycl::ext::intel::info::device::free_memory>()
  - SYCL extension
- Not on default.
  - SYCL only code, user initializes sysman.
  - OpenMP code, Need environment variable ZES_ENABLE_SYSMAN=1