OpenMP Offload Programming

Dr.-Ing. Michael Klemm
Principal Member of Technical Staff
HPC Center of Excellence

Agenda

- Avoiding Data Transfers
- Asynchronous Offloading
- Hybrid Programming with OpenMP and HIP
- Integrating Asynchronous Programming Models

 |

Running Example for this Presentation: saxpy

```
void saxpy() {
    float a, x[SZ], y[SZ];
    // left out initialization
    double t = 0.0;
    double tb, te;
    tb = omp_get_wtime();
    #pragma omp parallel for firstprivate(a)
    for (int i = 0; i < SZ; i++) {
        y[i] = a * x[i] + y[i];
    }
    te = omp_get_wtime();
    t = te - tb;
    printf("Time of kernel: %lf\n", t);
}
```

Timing code (not needed, just to have a bit more code to show ☺)

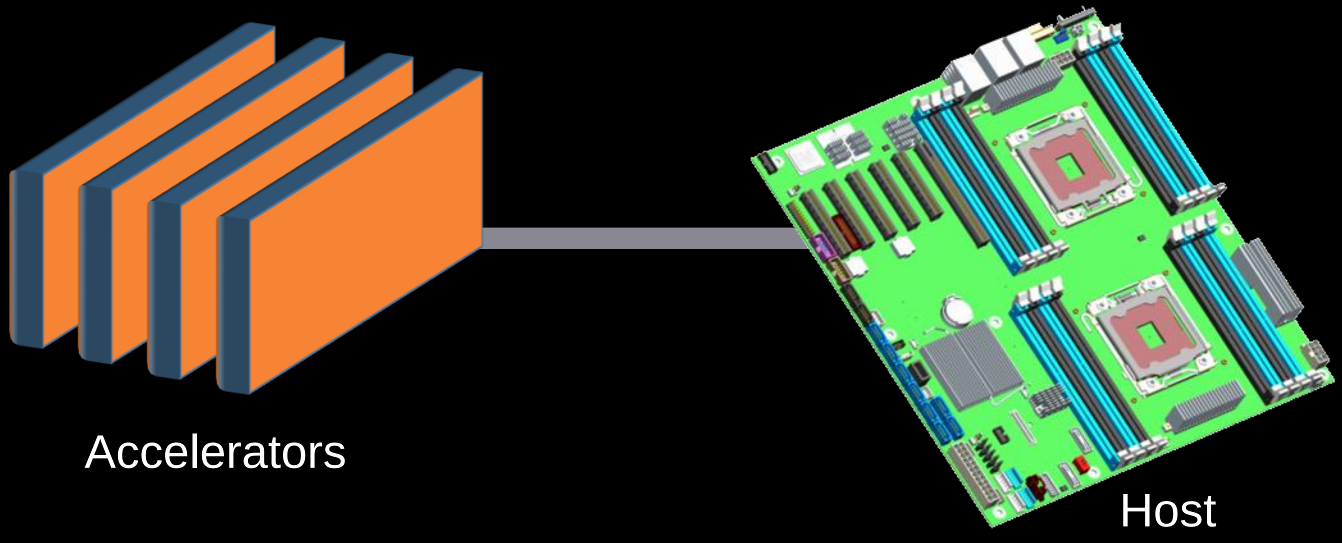
This is the code we want to execute on a target device (i.e., GPU)

Timing code (not needed, just to have a bit more code to show ☺)

Don't do this at home!
Use a BLAS library for this!

Avoiding Data Transfers

Optimizing Data Transfers is Key to Performance



- Connections between host and accelerator are typically lower-bandwidth, higher-latency interconnects
  - Bandwidth host memory: hundreds of GB/sec
  - Bandwidth accelerator memory: TB/sec
  - PCIe Gen 4 bandwidth (16x): tens of GB/sec

- Unnecessary data transfers must be avoided, by
  - only transferring what is actually needed for the computation, and
  - making the lifetime of the data on the target device as long as possible.

Optimize Data Transfers

- Reduce the amount of time spent transferring data
  - Use `map` clauses to enforce direction of data transfer.
  - Use `target data`, `target enter data`, `target exit data` constructs to keep data environment on the target device.

No `map` clauses! Presence checks will find data via the pointer.

```
void example() {
    float tmp[N], data_in[N], float data_out[N];
    #pragma omp target data map(alloc:tmp[:N])  \
                            map(to:a[:N],b[:N]) \
                            map(tofrom:c[:N])
    {
        zeros(tmp, N);
        compute_kernel_1(tmp, a, N); // uses target
        saxpy(2.0f, tmp, b, N);
        compute_kernel_2(tmp, b, N); // uses target
        saxpy(2.0f, c, tmp, N);
}   }
```

Create data environment.

```
void zeros(float* a, int n) {
    #pragma omp target teams distribute parallel for
    for (int i = 0; i < n; i++)
        a[i] = 0.0f;
}
```

```
void saxpy(float a, float* y, float* x, int n) {
    #pragma omp target teams distribute parallel for
    for (int i = 0; i < n; i++)
        y[i] = a * x[i] + y[i];
}
```


AMD

`target data` Construct Syntax

- Create scoped data environment and transfer data from the host to the device and back
- Syntax (C/C++)

```
#pragma omp target data [clause[[,] clause],…]
structured-block
```

- Syntax (Fortran)

```
!$omp target data [clause[[,] clause],…]
structured-block
!$omp end target data
```

- Clauses

```
device(scalar-integer-expression)
map([{alloc | to | from | tofrom | release | delete}:] list)
if(scalar-expr)
```


`target update` Construct Syntax

- Issue data transfers to or from existing data device environment
- Syntax (C/C++)

```
#pragma omp target update [clause[[,] clause],…]
```

- Syntax (Fortran)

```
!$omp target update [clause[[,] clause],…]
```

- Clauses

```
device(scalar-integer-expression)
to(list)
from(list)
if(scalar-expr)
```


Example: target data and target update

```
#pragma omp target data device(0) map(alloc:tmp[:N]) map(to:input[:N)) map(from:res)
  {
#pragma omp target device(0)
#pragma omp teams distribute parallel for simd
    for (i=0; i<N; i++)
      tmp[i] = some_computation(input[i], i);

    update_input_array_on_the_host(input);

#pragma omp target update device(0) to(input[:N])

#pragma omp target device(0)
#pragma omp teams parallel for simd reduction(+:res)
    for (i=0; i<N; i++)
      res += final_computation(input[i], tmp[i], i)
  }
```

host

target

host

target

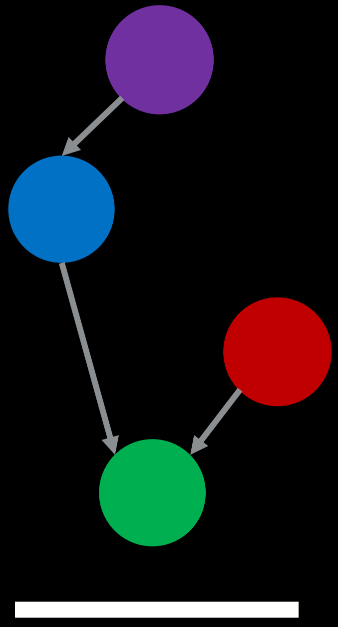
host

Avoiding Data Transfers

Asynchronous Offloads

- OpenMP `target` constructs are synchronous by default.
  - The encountering host thread awaits the end of the `target` region before continuing.
  - The `nowait` clause makes the target constructs asynchronous
    (in OpenMP lingo: they become an OpenMP task).

```
#pragma omp task                                                     depend(out:a)
    init_data(a);

#pragma omp target map(to:a[:N]) map(from:x[:N])        nowait   depend(in:a) depend(out:x)
    compute_1(a, x, N);

#pragma omp target map(to:b[:N]) map(from:y[:N])        nowait   depend(in:b) depend(out:y)
    compute_2(b, y, N);

#pragma omp target map(to:x[:N],y[:N]) map(from:z[:N]) nowait   depend(in:x) depend(in:y)
    compute_3(x, y, z, N);

#pragma omp taskwait
```



AMD

Hybrid Programming with OpenMP + HIP

Hybrid Programming

- Hybrid programming here stands for the interaction of OpenMP with a lower-level programming model, e.g.,
  - OpenCL
  - HIP
- OpenMP supports these interactions
  - Calling low-level HIP kernels from OpenMP application code
  - Calling OpenMP kernels from low-level application code
  - Interaction with the underlying asynchronous stream mechanism

HIP Buffer Management

```
void example() {
    HIPCALL(hipSetDevice(0));

    compute_1(n, x);
    compute_2(n, y);

    HIPCALL(hipMalloc(&x_dev, sizeof(*x_dev) * count));
    HIPCALL(hipMalloc(&y_dev, sizeof(*y_dev) * count));
    HIPCALL(hipMemcpy(x_dev, x, sizeof(*x) * count, hipMemcpyHostToDevice));
    HIPCALL(hipMemcpy(y_dev, y, sizeof(*y) * count, hipMemcpyHostToDevice));

    saxpy_omp(count, a, x_dev, y_dev);

    HIPCALL(hipMemcpy(y, y_dev, sizeof(*y) * count, hipMemcpyDeviceToHost));
    HIPCALL(hipFree(x_dev));
    HIPCALL(hipFree(y_dev));

    compute_3(n, y);
}
```

Allocate buffers to hold data on the target GPU.

Copy the data from the host memory to the GPU buffer space.

Copy result data back from GPU.

Deallocate the buffers on the target GPU.

AMD

HIP Buffer Management

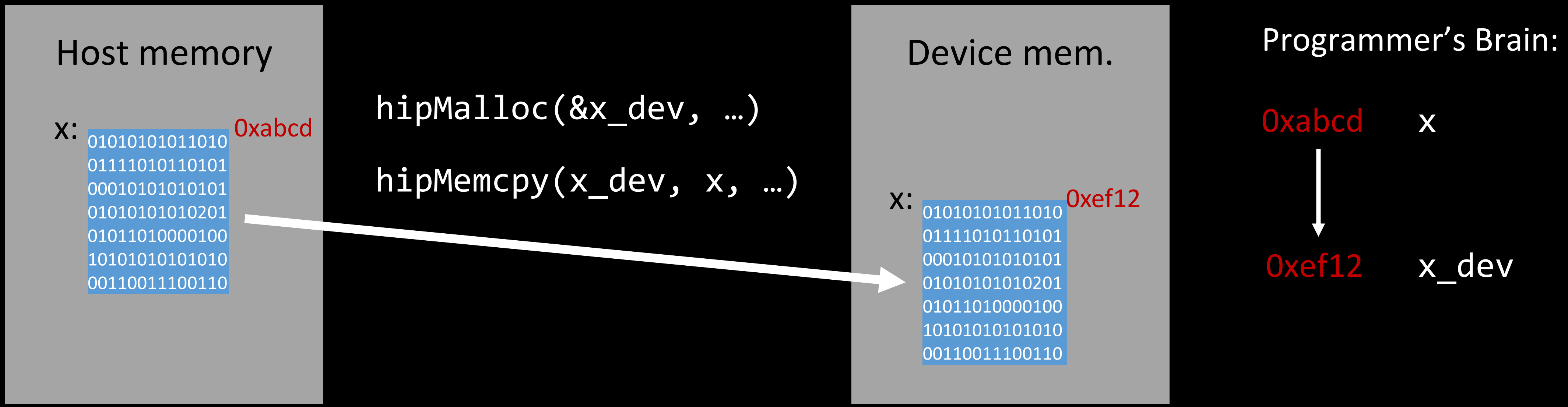
```
void example() {
    HIPCALL(hipSetDevice(0));

    compute_1(n, x);
    compute_2(n, y);

    HIPCALL(hipMalloc(&x_dev, sizeof(*x_dev) * count));
    HIPCALL(hipMalloc(&y_dev, sizeof(*y_dev) * count));
    HIPCALL(hipMemcpy(x_dev, x, sizeof(*x) * c
    HIPCALL(hipMemcpy(y_dev, y, sizeof(*y) * c

    saxpy_omp(count, a, x_dev, y_dev);

    HIPCALL(hipMemcpy(y, y_dev, sizeof(*y) * c
    HIPCALL(hipFree(x_dev));
    HIPCALL(hipFree(y_dev));

    compute_3(n, y);
}
```

```
void saxpy_omp(size_t n, float a,
               float * x, float * y) {
#pragma omp target teams distribute \
                   parallel for simd
    for (size_t i = 0; i < n; ++i) {
        y[i] = a * x[i] + y[i];
    }
}
```

OpenMP region needs to access the existing device pointers, no pointer translation please!

HIP "Pointer Translation"

- In the HIP model, "pointer translation" is handled by the programmer!
  - Explicitly associate host pointer ("x") with device pointer ("`x_dev`").
  - Association is done via the `hipMemcpy()` API that requires both as arguments.



 |

AMD

Disabling OpenMP Presence Check (and Pointer Translation)

- The OpenMP `target` construct has the `is_device_ptr()` clause that
  - instructs the OpenMP implementation to not do a presence check for the listed entities, and
  - avoids pointer translation and passes the given pointer value into the kernel w/o further interpretation.

```
void saxpy_omp(size_t n, float a, float * x, float * y) {
    #pragma omp target teams distribute parallel for \
                schedule(nonmonotonic:static,1)      \
                is_device_ptr(x, y)
    for (size_t i = 0; i < n; ++i) {
        y[i] = a * x[i] + y[i];
    }
}
```

 |

Calling HIP from OpenMP Offload Regions

Example: Calling saxpy

```
void example() {
    float a = 2.0;
    float * x;
    float * y;

    #pragma omp target data map(to:x[0:count]) map(tofrom:y[0   nt])
    {
        compute_1(n, x);
        compute_2(n, y);
        #pragma omp target update to(x[0:count
        saxpy(n, a, x, y)
        compute_3(n, y);
    }
}
```

Allocate device memory for x and y, and specify directions of data transfers

Let's assume that we want to implement the `saxpy()` function in a low-level language.

```
void saxpy(size_t n, float a,
           float * x, float * y) {
#pragma omp target teams distribute \
                    parallel for …
    for (size_t i = 0; i < n; ++i) {
        y[i] = a * x[i] + y[i];
    }
}
```

 |

HIP Kernel for saxpy()

- Assume a HIP version of the SAXPY kernel:

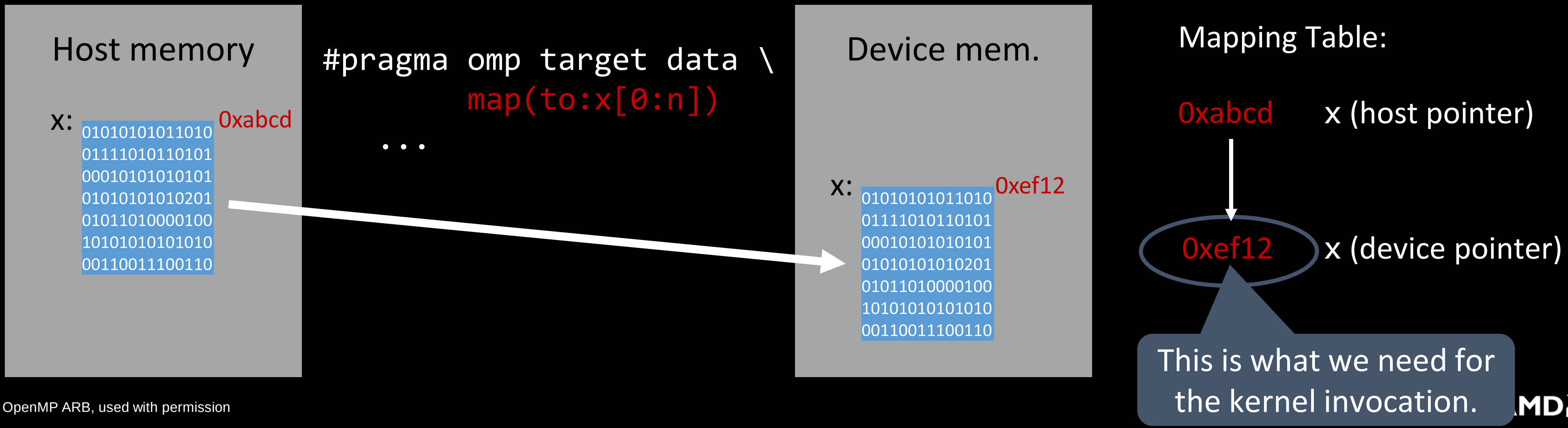
```
__global__ void saxpy_kernel(size_t n, float a, float * x, float * y) {
    size_t i = threadIdx.x + blockIdx.x * blockDim.x;
    y[i] = a * x[i] + y[i];
}

void saxpy_hip(size_t n, float a, float * x, float * y) {
    assert(n % 256 == 0);
    saxpy_kernel<<<n/256,256,0,NULL>>>(n, a, x, y);
}
```

These are device pointers!

- We need a way to translate the host pointer that was mapped by OpenMP directives and retrieve the associated device pointer.

Pointer Translation /1

- When creating the device data environment, OpenMP creates a mapping between
  - the (virtual) memory pointer on the host and
  - the (virtual) memory pointer on the target device.
- This mapping is established through the data-mapping directives and their clauses.



 |

Pointer Translation /2

- The target data construct defines the `use_device_ptr` clause to perform pointer translation.
  - The OpenMP implementation searches for the host pointer in its internal mapping tables.
  - The associated device pointer is then returned.

```
type * x = 0xabcd;
#pragma omp target data use_device_ptr(x)
{
    example_func(x);   // x == 0xef12
}
```

- Note: the pointer variable is “shadowed” within the `target data` construct for the translation.

Putting it Together…

```
void example() {
    float a = 2.0;
    float * x = ...;   // assume: x = 0xabcd
    float * y = ...;

    // allocate the device memory
    #pragma omp target data map(to:x[0:count]) map(tofrom:y[0:count])
    {
        compute_1(n, x);  // mapping table: x:[0xabcd,0xef12], x = 0xabcd
        compute_2(n, y);
        #pragma omp target update to(x[0:count]) to(y[0:count])  // update x and y on the target
        #pragma omp target data use_device_ptr(x,y)
        {
             saxpy_hip(n, a, x, y) // mapping table: x:[0xabcd,0xef12], x = 0xef12
        }
    }
    compute_3(n, y);
}
```


AOMP Implementation Status

- Call HIP kernel with OpenMP-managed buffers (`use_device_ptr`) ✓
- Call OpenMP kernels with HIP-managed buffers (`is_device_ptr`) ✓
- HIP and OpenMP kernels co-existence in same translation unit ✗

Integrating Asynchronous Programming Models

Asynchronous API Interaction

- Some APIs are based on asynchronous operations
  - MPI asynchronous send and receive
  - Asynchronous I/O
  - HIP stream-based offloading
  - In general: any other API/model that executes asynchronously with OpenMP (tasks)
- Example: HIP asynchronous memory transfers

```
do_something();
hipMemcpyAsync(dst, src, nbytes, hipMemcpyDeviceToHost, stream);
do_something_else();
hipStreamSynchronize(stream);
do_other_important_stuff(dst);
```

- Programmers need a mechanism to marry asynchronous APIs with the parallel task model of OpenMP
  - How to synchronize completions events with task execution?

Try 1: Use just OpenMP Tasks

```
void hip_example() {
#pragma omp task     // task A
    {
        do_something();
        hipMemcpyAsync(dst, src, nbytes, hipMemcpyDeviceToHost, stream);
    }
    #pragma omp task // task B
    {
        do_something_else();
    }
    #pragma omp task // task C
    {
        hipStreamSynchronize(stream);
        do_other_important_stuff(dst);
    }
}
```

Race condition between the tasks A & C, task C may start execution before task A enqueues memory transfer.

- This solution does not work!

[Public]

Try 2: Use just OpenMP Tasks Dependences

```
void hip_example() {
#pragma omp task depend(out:stream)    // task A
    {
        do_something();
        hipMemcpyAsync(dst, src, nbytes, hipMemcpyDeviceToHost, stream);
    }
    #pragma omp task                   // task B
    {
        do_something_else();
    }
    #pragma omp task depend(in:stream) // task C
    {
        hipStreamSynchronize(stream);
        do_other_important_stuff(dst);
    }
}
```

Synchronize execution of tasks through dependence. May work, but task C will be blocked waiting for the data transfer to finish

- This solution may work, but
  - Takes a thread away from execution while the system is handling the data transfer and may be problematic if the called interface is not thread-safe!

AMD

OpenMP Detachable Tasks

- OpenMP 5.0 introduces the concept of a detachable task
  - Task can detach from executing thread without being “completed”
  - Regular task synchronization mechanisms can be applied to await completion of a detached task
  - Runtime API to complete a task

- Detached task events: `omp_event_handle_t` datatype
- Detached task clause: `detach(event)`
- Runtime API: `void omp_fulfill_event(omp_event_handle_t event)`

Detaching Tasks

```
omp_event_handle_t event;
void detach_example() {
#pragma omp task detach(event)
    {
        important_code();
    }①

    #pragma omp taskwait ②④
}
```

Some other thread/task:

```
omp_fulfill_event(event);③
```

1. Task detaches
2. `taskwait` construct cannot complete
3. Signal event for completion
4. Task completes and `taskwait` can continue

Putting It All Together

```
void callback(hipStream_t stream, hipError_t status, void *cb_dat) {
 ③ omp_fulfill_event(* (omp_event_handle_t *) cb_data);
}
void hip_example() {
    omp_event_handle_t hip_event;
#pragma omp task detach(hip_event) // task A
    {
        do_something();
        hipMemcpyAsync(dst, src, nbytes, hipMemcpyDeviceToHost, stream);
        hipStreamAddCallback(stream, callback, &hip_event, 0);
 ①  }
#pragma omp task                    // task B
        do_something_else();

#pragma omp taskwait②④
#pragma omp task                    // task C
    {
        do_other_important_stuff(dst);
}   }
```

1. Task A detaches
2. taskwait does not continue
3. When memory transfer completes, callback is invoked to signal the event for task completion
4. taskwait continues, task C executes

[Public]

Removing the taskwait Construct

```
void callback(hipStream_t stream, hipError_t status, void *cb_dat) {
  ② omp_fulfill_event(* (omp_event_handle_t *) cb_data);
}
void hip_example() {
    omp_event_handle_t hip_event;
#pragma omp task depend(out:dst) detach(hip_event) // task A
    {
        do_something();
        hipMemcpyAsync(dst, src, nbytes, hipMemcpyDeviceToHost, stream);
        hipStreamAddCallback(stream, callback, &hip_event, 0);
  ① }
#pragma omp task                        // task B
        do_something_else();

#pragma omp task depend(in:dst) ③       // task C
    {
        do_other_important_stuff(dst);
}   }
```

1. Task A detaches and task C will not execute because of its unfulfilled dependency on A
2. When memory transfer completes, callback is invoked to signal the event for task completion
3. Task A completes and C's dependency is fulfilled

AMD

Summary

- OpenMP API is ready to use AMD discrete GPUs for offloading compute
  - Mature offload model w/ support for asynchronous offload/transfer
  - Tightly integrates with OpenMP multi-threading on the host

- More, advanced features (not covered here)
  - Memory management API
  - Interoperability with native streaming interfaces
  - Unified shared memory support

Disclaimer

The information presented in this document is for informational purposes only and may contain technical inaccuracies, omissions, and typographical errors. The information contained herein is subject to change and may be rendered inaccurate for many reasons, including but not limited to product and roadmap changes, component and motherboard version changes, new model and/or product releases, product differences between differing manufacturers, software changes, BIOS flashes, firmware upgrades, or the like. Any computer system has risks of security vulnerabilities that cannot be completely prevented or mitigated. AMD assumes no obligation to update or otherwise correct or revise this information. However, AMD reserves the right to revise this information and to make changes from time to time to the content hereof without obligation of AMD to notify any person of such revisions or changes.

THIS INFORMATION IS PROVIDED 'AS IS." AMD MAKES NO REPRESENTATIONS OR WARRANTIES WITH RESPECT TO THE CONTENTS HEREOF AND ASSUMES NO RESPONSIBILITY FOR ANY INACCURACIES, ERRORS, OR OMISSIONS THAT MAY APPEAR IN THIS INFORMATION. AMD SPECIFICALLY DISCLAIMS ANY IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, OR FITNESS FOR ANY PARTICULAR PURPOSE. IN NO EVENT WILL AMD BE LIABLE TO ANY PERSON FOR ANY RELIANCE, DIRECT, INDIRECT, SPECIAL, OR OTHER CONSEQUENTIAL DAMAGES ARISING FROM THE USE OF ANY INFORMATION CONTAINED HEREIN, EVEN IF AMD IS EXPRESSLY ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

AMD