Offloading in OpenMP

Christian Terboven <terboven@itc.rwth-aachen.de>

with slides developed for ISC/SC tutorials with members of the OpenMP Language Committee

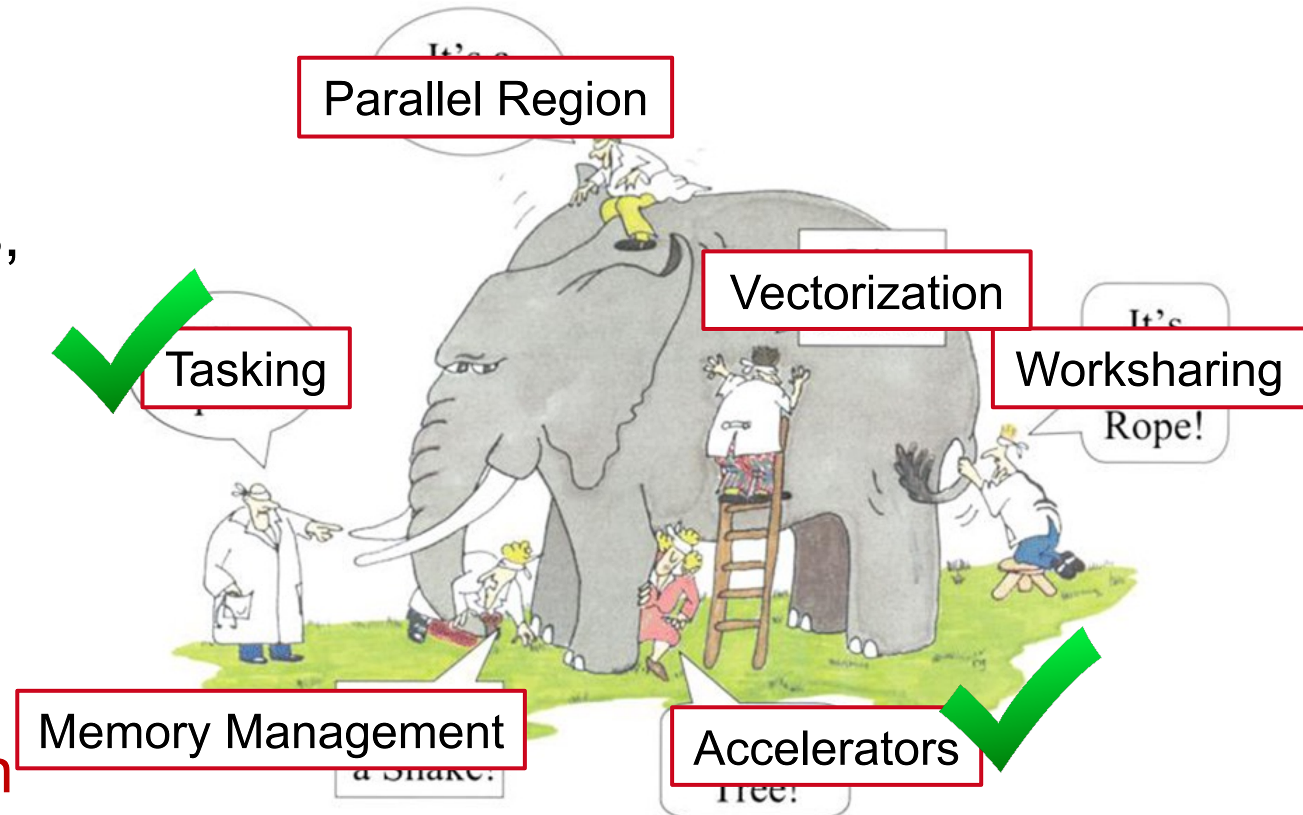
April 5th, 2022

NHR PerfLab Seminar Series

What is OpenMP?

- De-facto standard Application Programming Interface (API) to write <u>shared memory parallel</u> applications in C, C++, and Fortran
- Consists of Compiler Directives, Runtime routines and Environment variables

- Version 5.0 has been released at SC 2018
- Versions 5.1 and 5.2 have been released at SC 2020 and 2021



NHR4CES IT Center RWTH AACHEN UNIVERSITY

Overview

Running example for this presentation: saxpy (in one form or the other)

```
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)

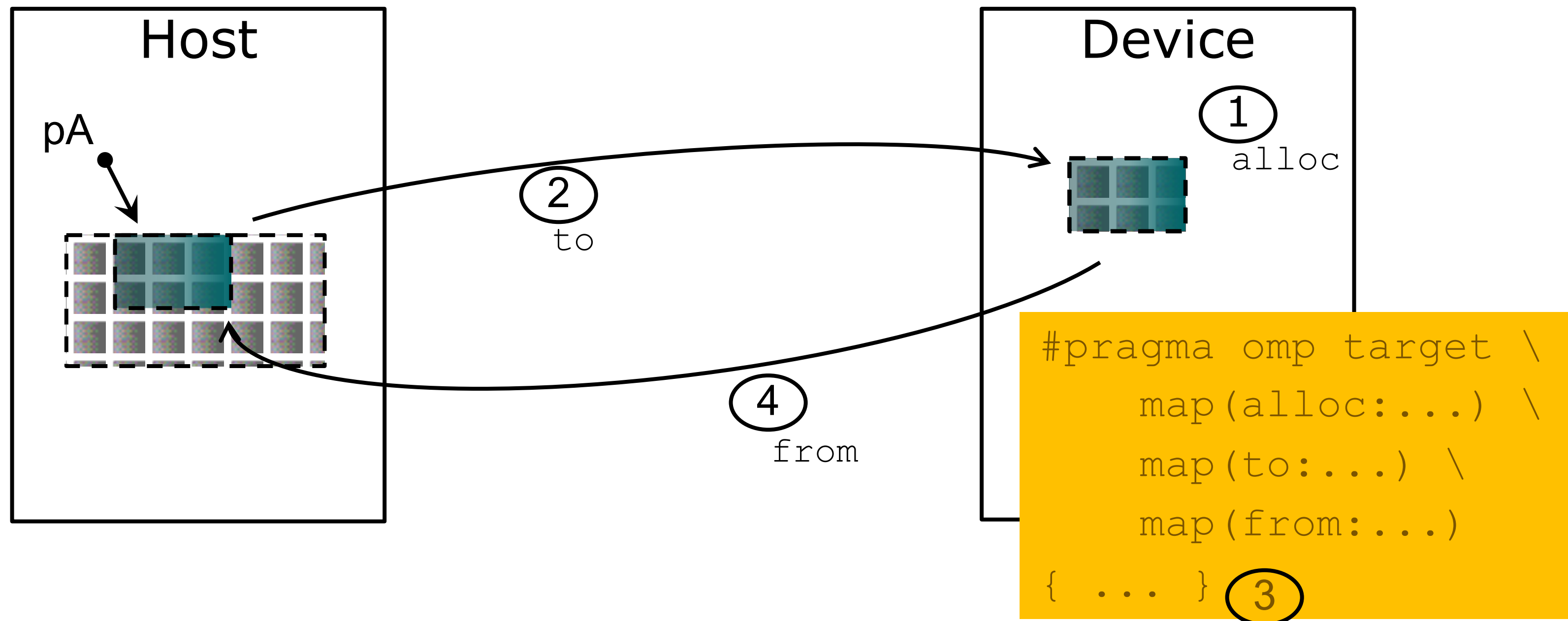
Timing code (not needed, just to have a bit more code to show ☺)

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

NHR4CES | IT Center | RWTH AACHEN UNIVERSITY

Offloading illustrated

- Offload region and data environment is lexically scoped
  - Data environment is destroyed at closing curly brace
  - Allocated buffers/data are automatically released



- The `target` construct transfers the control flow to the target device
- OpenMP separates offload and parallelism
  - Programmers need to explicitly create parallel regions on the target device
  - In theory, this can be combined with any OpenMP construct
  - In practice, there is only a useful subset of OpenMP features for a target device such as a GPU, e.g., no I/O, limited use of base language features.

Running example for this presentation: saxpy / 1

```
void saxpy() {
    float a, x[SZ], y[SZ];
    double t = 0.0;
    double tb, te;
    tb = omp_get_wtime();
#pragma omp target "map(tofrom:y[0:SZ])"
    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);
}
```

host

a
x[0:SZ]
y[0:SZ]

target

x[0:SZ]
y[0:SZ]

host

The compiler identifies variables that are used in the `target` region.

All accessed arrays are copied from host to device and back

Presence check: only transfer if not yet allocated on the device.

Copying x back is not necessary: it was not changed.

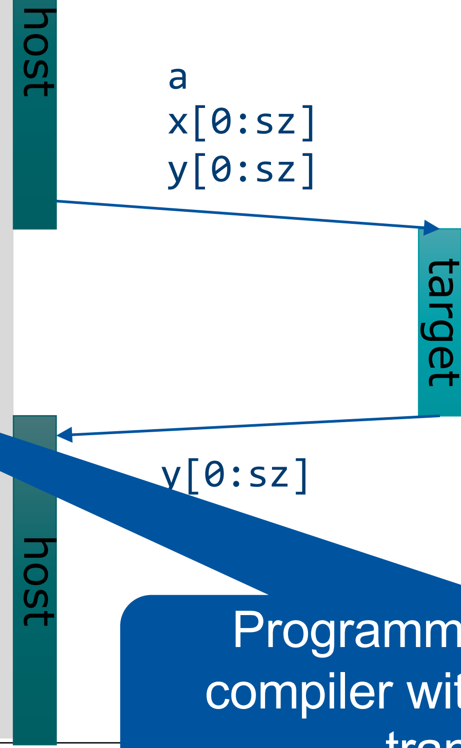
```
clang -fopenmp -fopenmp-targets=amdgcn-amd-amdhsa -Xopenmp-target=amdgcn-amd-amdhsa -march=gfx908
```


NHR4CES IT Center RWTH AACHEN UNIVERSITY

Running example for this presentation: saxpy / 2

```
void saxpy(float a, float* x, float* y,
           int sz) {
    double t = 0.0;
    double tb, te;
    tb = omp_get_wtime();
#pragma omp target map(to:x[0:sz]) \
                   map(tofrom:y[0:sz])
    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);
}
```

The compiler cannot determine the size of memory behind the pointer.



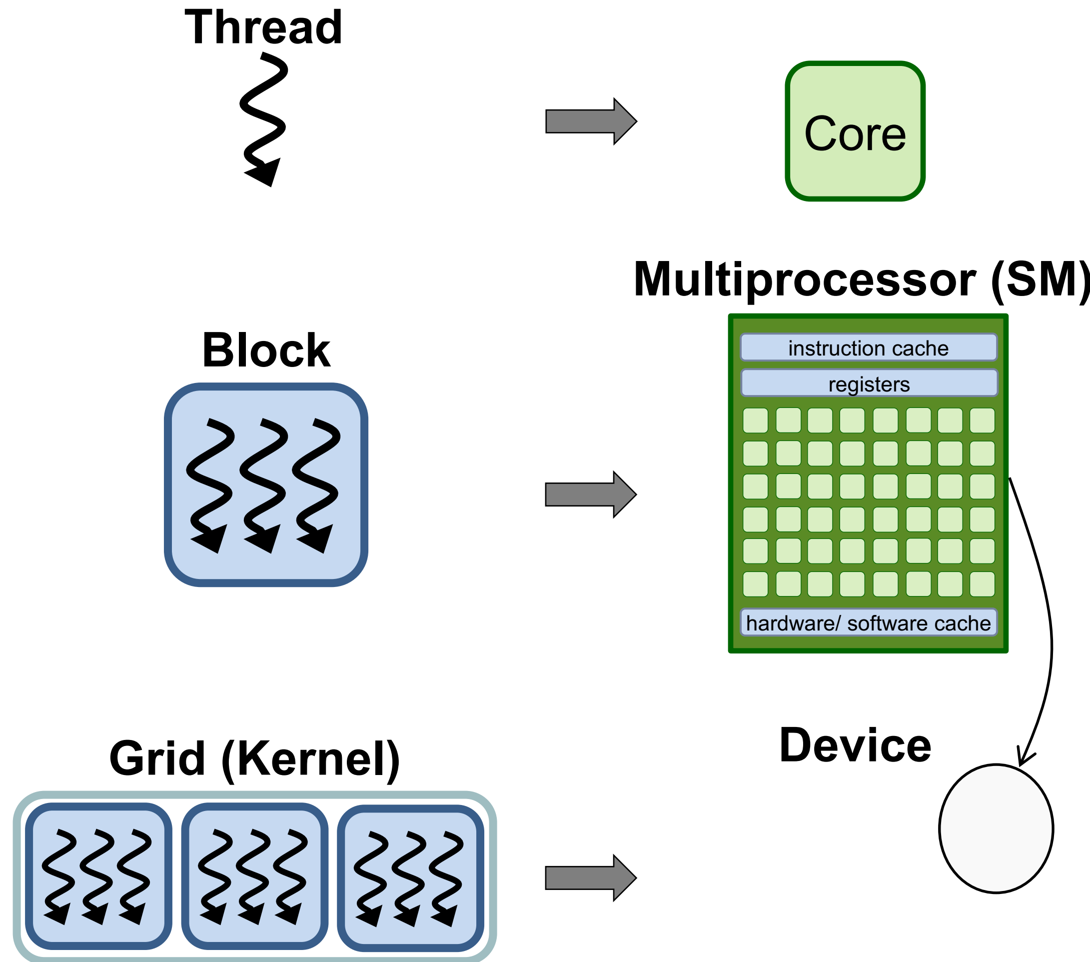
Programmers have to help the compiler with the size of the data transfer needed.

```
clang -fopenmp -fopenmp-targets=amdgcn-amd-amdhsa -Xopenmp-target=amdgcn-amd-amdhsa -march=gfx908
```


NHR4CES IT Center RWTH AACHEN UNIVERSITY

Fundamental concepts

Mapping to Hardware

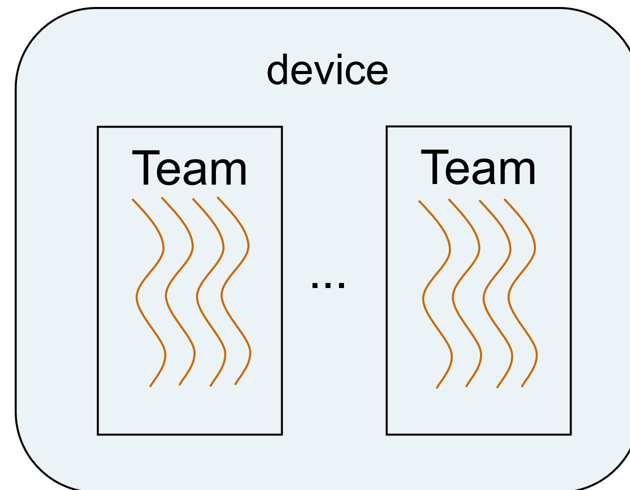


- Each thread is executed by a core
- Each block is executed on a SM
- Several concurrent blocks can reside on a SM depending on shared resources
- Each kernel is executed on a device

NHR4CES | IT Center | RWTH AACHEN UNIVERSITY

OpenMP Terminology

- League:
  the set of threads teams created by a **`teams`** construct
- Contention group:
  threads of a team in a league and their descendant threads



NHR4CES | IT Center | RWTH AACHEN UNIVERSITY

Running example for this presentation: saxpy / 3

- Manual code transformation
  - Tile the loops into an outer loop and an inner loop
  - Assign the outer loop to “teams” (OpenCL: work groups)
  - Assign the inner loop to the “threads” (OpenCL: work items)

NHR4CES | IT Center | RWTH AACHEN UNIVERSITY

Running example for this presentation: saxpy / 4

- For convenience, OpenMP defines composite constructs to implement the required code transformations

```
void saxpy(float a, float* x, float* y, int sz) {
    #pragma omp target teams distribute parallel for simd \
            num_teams(num_blocks) map(to:x[0:sz]) map(tofrom:y[0:sz])
    for (int i = 0; i < sz; i++) {
        y[i] = a * x[i] + y[i];
    }
}
```


NHR4CES | IT Center | RWTH AACHEN UNIVERSITY

Optimizing data transfers and asynchronous Offloads

Optimizing 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.
    - `target enter data`, `target exit data`: unstructured data movement

```
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);
}   }
```

```
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];
}
```

Example

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

host
target
host
target
host

NHR4CES | IT Center | RWTH AACHEN UNIVERSITY

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 speak: they become an OpenMP task)

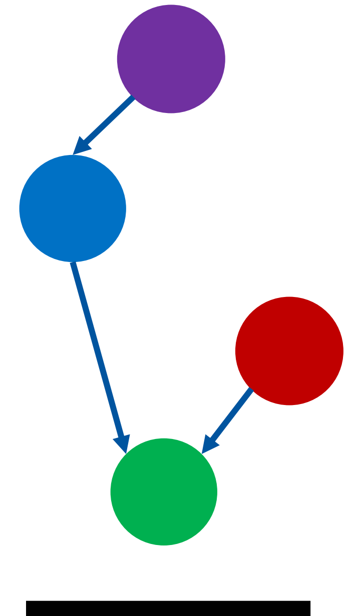
```
#pragma omp task                                                depend(out:a)
    init_data(a);

#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:z[:N])  nowait      depend(out:z)
    compute_3(b, z, N);

#pragma omp target map(to:y[:N]) map(to:z[:N])    nowait      depend(in:x) depend(in:z)
    compute_4(z, x, y, N);

#pragma omp taskwait
```

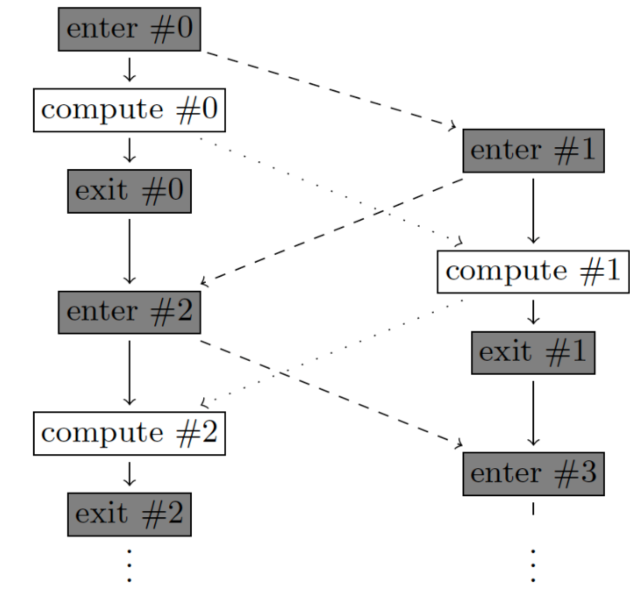


NHR4CES | IT Center | RWTH AACHEN UNIVERSITY

Pipelining: asynchronous unstructured data movement and kernel execution

```
double A[ BLOCKS * LEN ];
int enter , compute ;

#pragma omp target enter data nowait map(to: A [0: LEN ]) \
        depend ( out: enter ) depend ( out: A [0: LEN ])
for (int block = 0; block < BLOCKS ; block ++) {
   #pragma omp target enter data nowait depend ( inout : enter ) \
           map (to: A[( block + 1) * LEN: LEN ]) \
           depend (out : A[( block + 1) * LEN: LEN ]) \
           depend (in: A[( block - 1) * LEN: LEN ])
   #pragma omp target nowait depend ( inout : compute ) \
           map (to: A[ block * LEN: LEN ]) \
           depend ( inout : A[ block * LEN: LEN ])
   {
      // do computation here
   }
   #pragma omp target exit data nowait \
           map ( release : A[ block * LEN: LEN ]) \
           depend ( inout : A[ block * LEN: LEN ])
}
```



Pipelining concept to compute multiple blocks of size len

Strength of OpenMP: integration!

 J. Hahnfeld, C. Terboven, J. Price, H.-J. Pflug, M. S. Müller: Evaluation of Asynchronous Offloading Capabilities of Accelerator Programming Models for Multiple Devices. WACCPD @ SC2017

CES | IT Center | RWTH AACHEN UNIVERSITY

Advanced Task and Target Synchronization

Asynchronous API Interaction.

Asynchronous API Interaction

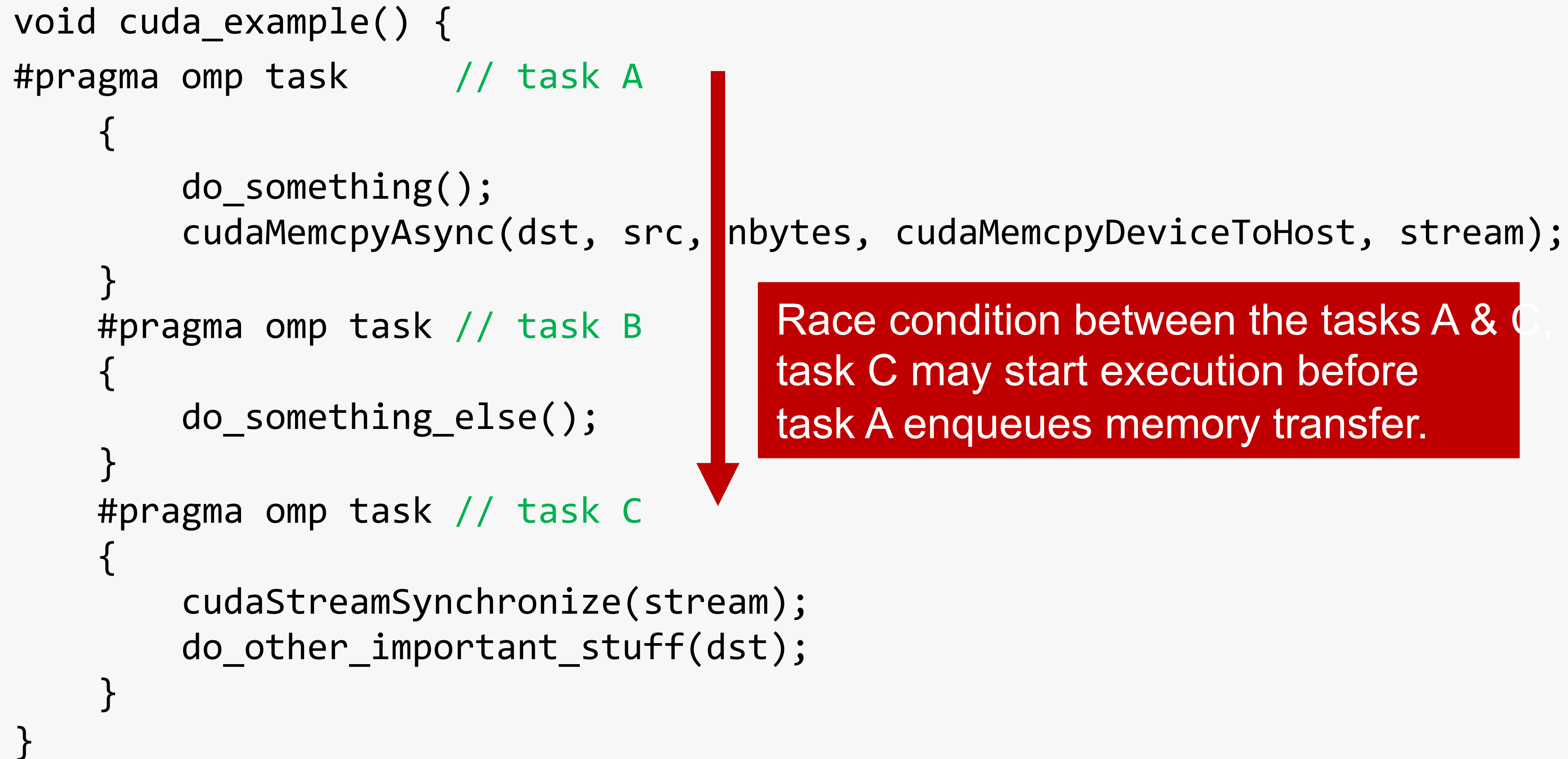
- Some APIs are based on asynchronous operations
  - MPI asynchronous send and receive
  - Asynchronous I/O
  - HIP, CUDA and OpenCL stream-based offloading
  - In general: any other API/model that executes asynchronously with OpenMP (tasks)
- Example: CUDA memory transfers

```
do_something();
cudaMemcpyAsync(dst, src, nbytes, cudaMemcpyDeviceToHost, stream);
do_something_else();
cudaStreamSynchronize(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?

NHR4CES IT Center RWTH AACHEN UNIVERSITY

Try 1: Use just OpenMP Tasks



- This solution does not work!

NHR4CES
IT Center
RWTH AACHEN UNIVERSITY

Try 2: Use just OpenMP Tasks Dependences

```
void cuda_example() {
#pragma omp task depend(out:stream)    // task A
    {
        do_something();
        cudaMemcpyAsync(dst, src, nbytes, cudaMemcpyDeviceToHost, stream);
    }
    #pragma omp task                    // task B
    {
        do_something_else();
    }
    #pragma omp task depend(in:stream) // task C
    {
        cudaStreamSynchronize(stream);
        do_other_important_stuff(dst);
    }
}
```

Synchronize execution of tasks through dependen
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.

NHR4CES | IT Center | RWTH AACHEN UNIVERSITY

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_t` datatype
- Detached task clause: `detach(event)`
- Runtime API: `void omp_fulfill_event(omp_event_t *event)`

NHR4CES | IT Center | RWTH AACHEN UNIVERSITY

Detaching Tasks

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

    #pragma omp taskwait ② ④
}
```

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

NHR4CES | IT Center | RWTH AACHEN UNIVERSITY

Putting It All Together

```
void CUDART_CB callback(cudaStream_t stream, cudaError_t status, void *cb_dat) {
 ③ omp_fulfill_event((omp_event_t *) cb_data);
}
void cuda_example() {
    omp_event_t *cuda_event;
#pragma omp task detach(cuda_event) // task A
    {
        do_something();
        cudaMemcpyAsync(dst, src, nbytes, cudaMemcpyDeviceToHost, stream);
        cudaStreamAddCallback(stream, callback, cuda_event, 0);
 ① }
#pragma omp task                    // task B
        do_something_else();

#pragma omp taskwait ② ④
#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

NHR4CES IT Center RWTH AACHEN UNIVERSITY

Removing the taskwait Construct

```
void CUDART_CB callback(cudaStream_t stream, cudaError_t status, void *cb_dat) {
  ② omp_fulfill_event((omp_event_t *) cb_data);
}
void cuda_example() {
    omp_event_t *cuda_event;
#pragma omp task depend(out:dst) detach(cuda_event) // task A
    {
        do_something();
        cudaMemcpyAsync(dst, src, nbytes, cudaMemcpyDeviceToHost, stream);
  ①     cudaStreamAddCallback(stream, callback, cuda_event, 0);
    }
#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

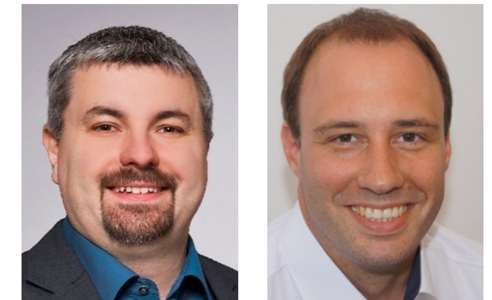
NHR4CES | IT Center | RWTH AACHEN UNIVERSITY

Summary

Summary

- OpenMP is ready to use 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 GPU-native data management
  - Interoperability with GPU-native streaming interfaces
  - Unified shared memory support



- This was not intended as a tutorial! If you want one, look out for these guys at conferences!
  - Some slides were developed in our joint tutorial efforts