Accelerators

Dirk Schmidl
schmidl@itc.rwth-aachen.de

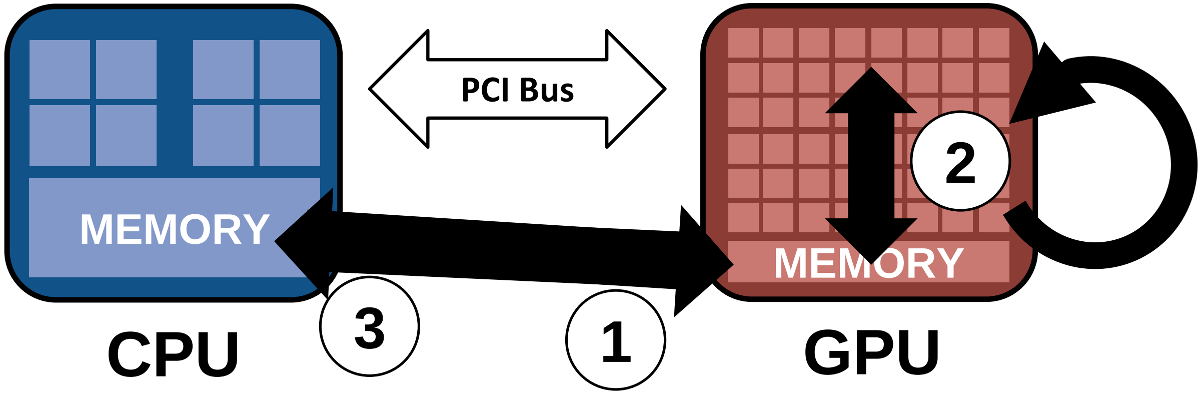
Thanks to the following people for providing parts of the slides:
- Christian Terboven (RWTH Aachen)
- Sandra Wienke (RWTH Aachen)
- Michael Klemm (Intel)

Devices

- In how differs an accelerator from just another core?
  - different functionality, i.e. optimized for something special
  - different (possibly limited) instruction set
    → heterogeneous device

- Assumptions used as design goals for OpenMP 4.0:
  - every accelerator device is attached to one host device
  - it is probably heterogeneous
  - it may or may not share memory with the host device

Execution Model

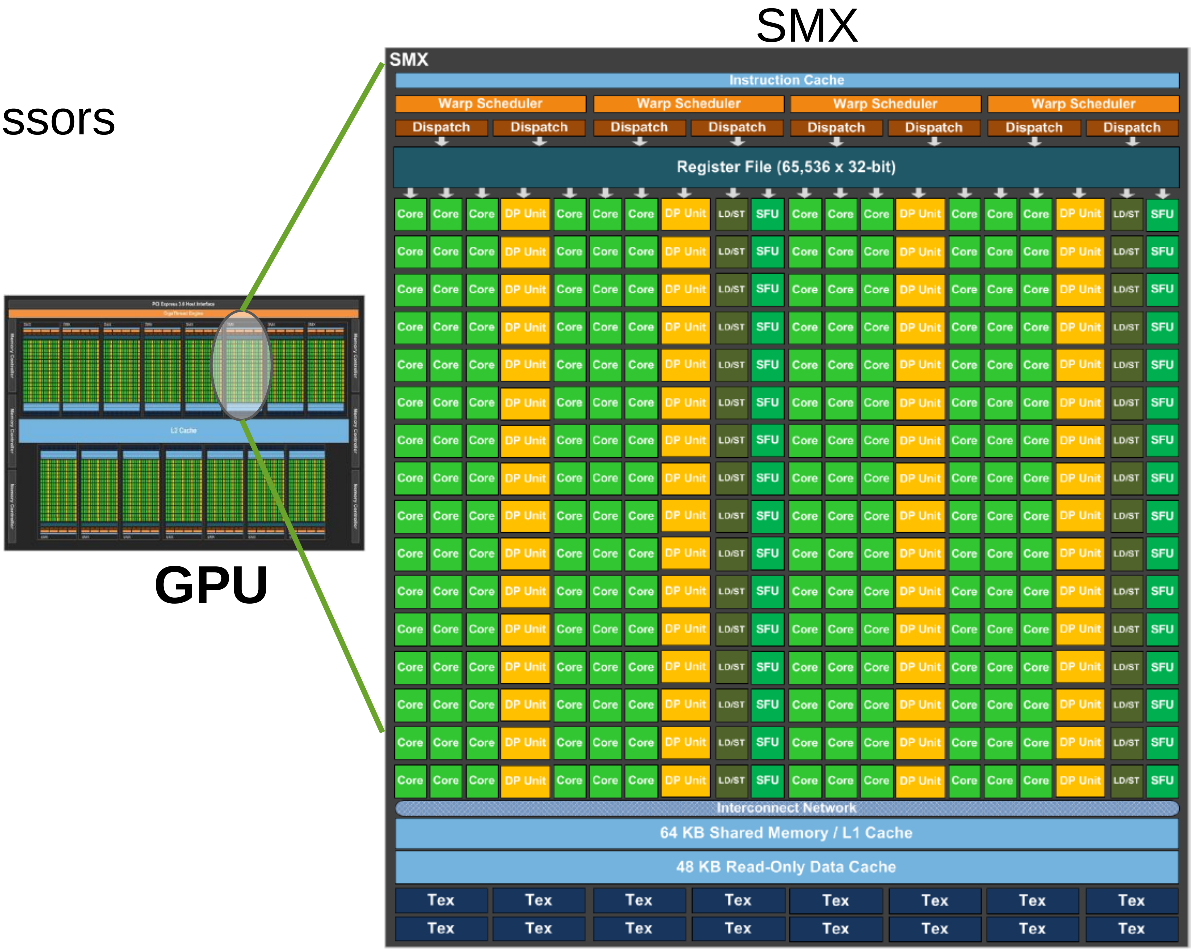


- Host-directed execution model
  - Copy input data from CPU mem. to device mem.
  - Execute the device program
  - Copy results from device mem. to CPU mem.

NVIDIA Kepler

- 7.1 billion transistors
- 13-15 streaming multiprocessors extreme (SMX)
  - Each comprises 192 cores
- 2496-2880 cores
- Memory hierarchy

- Peak performance (K20)
  - SP: 3.52 TFlops
  - DP: 1.17 TFlops
- ECC support



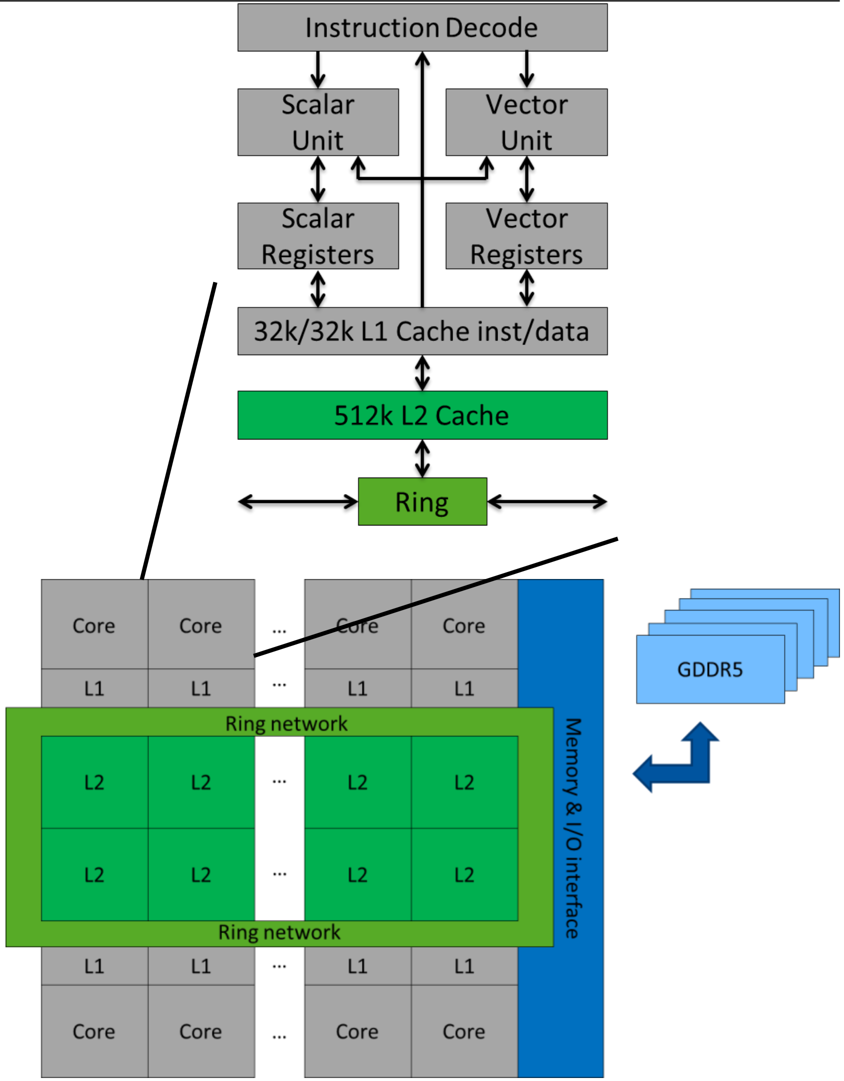
http://www.nvidia.com/content/PDF/kepler/NVIDIA-Kepler-GK110-Architecture-Whitepaper.pdf

Intel Knights Corner

Source: Intel

Intel Xeon Phi Coprocessor

- 1 x Intel Xeon Phi @ 1090 MHz
- 60 Cores (in-order)
- ~ 1 TFLOPS DP Peak
- 4 hardware threads per core (SMT)
- 8 GB GDDR5 memory
- 512-bit SIMD vectors (32 registers)
- Fully-coherent L1 and L2 caches
- Plugged into PCI Express bus



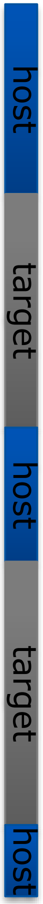
Execution and data model

The target construct

```
#pragma omp target device(0) map(to:input[:N]) map(from:tmp[:N])
#pragma omp parallel for
    for (i=0; i<N; i++)
      tmp[i] = some_computation(input[i], i);

    do_some_other_stuff_on_host();

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

offload

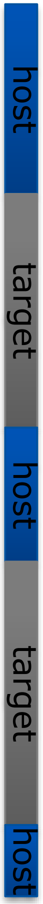
shaping and slicing



The target data construct

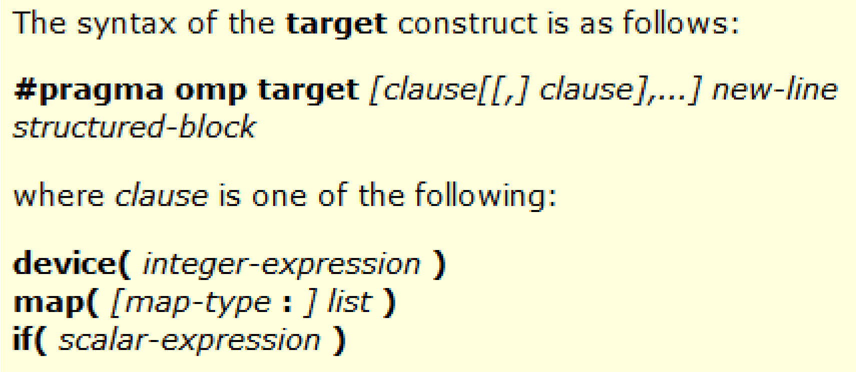
data region

```
#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);

    do_some_other_stuff_on_host();

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



The target Construct

- Transfers execution to a device
  - the region is executed on a device
  - the host thread waits for the region to be completed
  - data transfer is performed at entry and exit if needed



The syntax of the **target** construct is as follows:

#pragma omp target *[clause[[,] clause],...] new-line*
structured-block

where *clause* is one of the following:

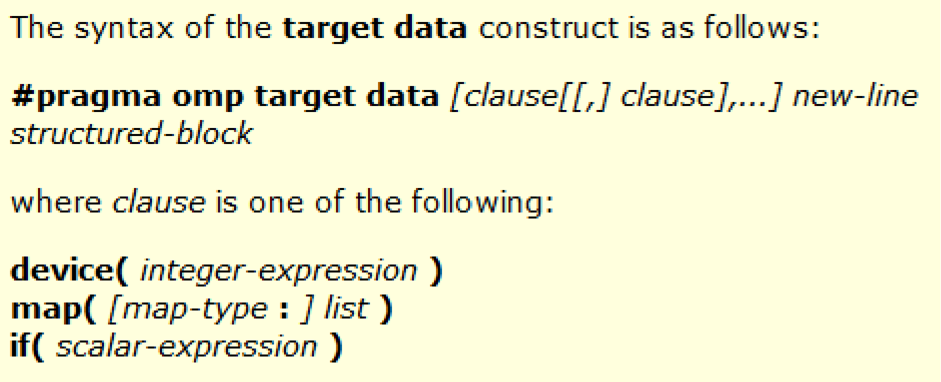
device(*integer-expression* **)**
map(*[map-type* **:** *] list* **)**
if(*scalar-expression* **)**

- Map a variable from the current task's data environment to the device data environment associated with the construct
  - the list items that appear in a map clause may include array sections
  - ***alloc***-type: each new corresponding list item has an undefined initial value
  - ***to***-type: each new corresponding list item is initialized with the original lit item's value
  - ***from***-type: declares that on exit from the region the corresponding list item's value is assigned to the original list item
  - ***tofrom***-type: the default, combination of to and from

The target data construct

- Creates a device data environment for the extent of the region
  - when a target data construct is encountered, a new device data environment is created, and the encountering task executes the target data region
  - when an if clause is present and the if-expression evaluates to false, the device is the host

- C/C++



The syntax of the **target data** construct is as follows:

#pragma omp target data *[clause[[,] clause],...] new-line*
structured-block

where *clause* is one of the following:

device(*integer-expression* **)**
map(*[map-type* **:** *] list* **)**
if(*scalar-expression* **)**

Synchronization of mapped variables

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

ERROR: Mapping of present data does not do an Update.

Synchronization of mapped variables

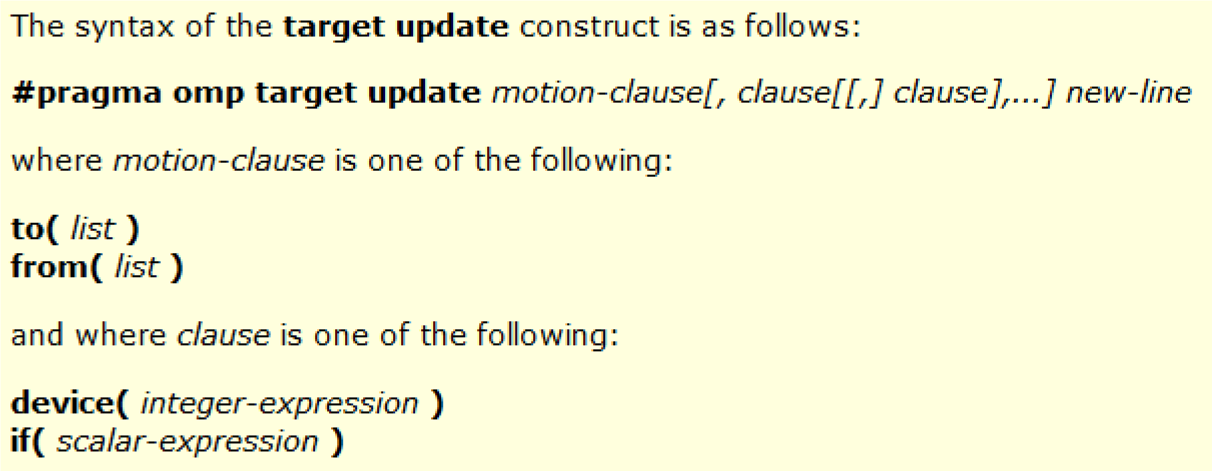
```
#pragma omp target data map(alloc:tmp[:N]) map(to:input[:N))
  map(from:res)
  {
#pragma omp target
#pragma omp parallel for
    for (i=0; i<N; i++)
      tmp[i] = some_computation(input[i], i);

    update_input_array_on_the_host(input);

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

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

update explicitly

target update

- Makes the corresponding list items in the device data environment consistent with their original list items, according to the specified motion clauses

- C/C++



The syntax of the **target update** construct is as follows:

#pragma omp target update *motion-clause[, clause[[,] clause],...] new-line*

where *motion-clause* is one of the following:

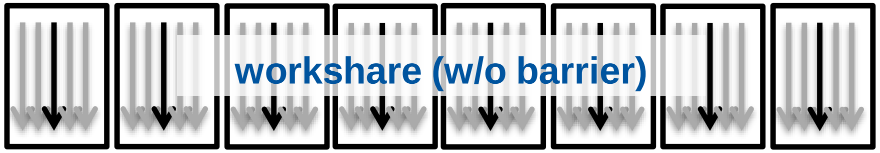
to(*list* **)**
from(*list* **)**

and where *clause* is one of the following:

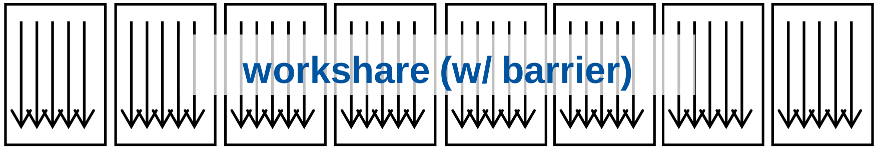
device(*integer-expression* **)**
if(*scalar-expression* **)**

Accelerated worksharing

```
void saxpy(float * restrict y, float * restrict x, float a, int n)
{
#pragma omp target teams map(to:n,a,x[:n]) map(y[:n])
{
 int block_size = n/omp_get_num_teams();

#pragma omp distribute dist_sched(static, 1)
  for (int i = 0; i < n; i += block_size){
```



```
#pragma omp parallel for
    for (int j = i; j < i + block_size; j++) {
```



```
        y[j] = a*x[j] + y[j];
}}
}
```

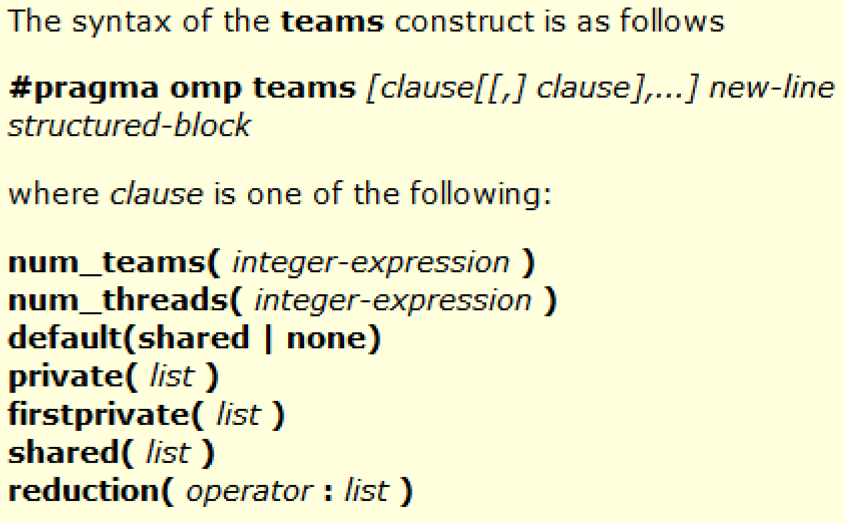
The teams construct

- Creates a league of thread teams where the master thread of each team executes the region
  - the number of teams is determined by the num_teams clause, the number of threads in each team is determined by the num_threads clause, within a team region team numbers uniquely identify each team (omp_get_team_num())
  - once created, the number of teams remeinas constant for the duration of the teams region
- The teams region is executed by the master thread of each team
- The threads other than the master thread to not begin execution until the master thread encounteres a parallel region
- Only the following constructs can be closely nested in the team region: distribute, parallel, parallel loop/for, parallel sections and parallel workshare

teams construct (2/2)

- A teams construct must be contained within a target construct, which must not contain any statements or directives outside of the teams construct
- After the teams have completed execution of the teams region, the encountering thread resumes execution of the enclosing target region

- C/C++



The syntax of the **teams** construct is as follows

#pragma omp teams *[clause[[,] clause],...] new-line*
structured-block

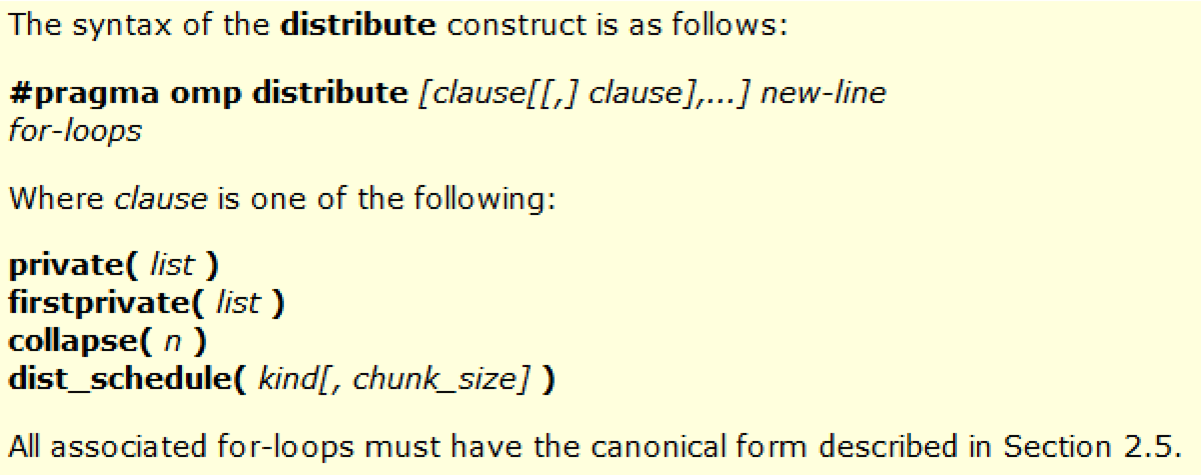
where *clause* is one of the following:

num_teams(*integer-expression* **)**
num_threads(*integer-expression* **)**
default(shared | none)
private(*list* **)**
firstprivate(*list* **)**
shared(*list* **)**
reduction(*operator* **:** *list* **)**

distribute construct

- Specifies that the iteration of one or more loops will be executed by the thread teams, the iterations are distributed across the master threads of all teams
  - there is no implicit barrier at the end of a distribute construct
  - a distribute construct must be closely nested in a teams region

- C/C++:



The syntax of the **distribute** construct is as follows:

#pragma omp distribute *[clause[[,] clause],...] new-line*
for-loops

Where *clause* is one of the following:

private(*list* **)**
firstprivate(*list* **)**
collapse(*n* **)**
dist_schedule(*kind[, chunk_size]* **)**

All associated for-loops must have the canonical form described in Section 2.5.

Run saxpy twice (Intel KNC)

```
  // Run SAXPY TWICE
#pragma omp target data map(to:x[0:n])
{
#pragma omp target map(tofrom:y[0:n])
#pragma omp parallel for
for (int i = 0; i < n; ++i){
        y[i] = a*x[i] + y[i];
  }

  // y is needed and modified on the host here
#pragma omp target map(tofrom:y[0:n])
#pragma omp parallel for
  for (int i = 0; i < n; ++i){
        y[i] = b*x[i] + y[i];
  }
}
```

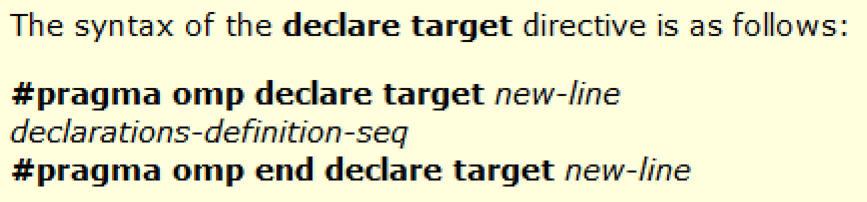
Run saxpy twice (GPGPU)

```
  // Run SAXPY TWICE
#pragma omp target data map(to:x[0:n])
{
#pragma omp target map(tofrom:y[0:n])
#pragma omp teams
#pragma omp distribute
#pragma omp parallel for
for (int i = 0; i < n; ++i){
       y[i] = a*x[i] + y[i];
  }

  // y is needed and modified on the host here
#pragma omp target map(tofrom:y[0:n])
#pragma omp teams
#pragma omp distribute
#pragma omp parallel for
  for (int i = 0; i < n; ++i){
       y[i] = b*x[i] + y[i];
  }
}
```

declare target directive

- Specifies that [static] variables, functions (C, C++ and Fortran) and subroutines (Fortran) are mapped to a device
  - if a list item is a function or subroutine then a device-specific version of the routines is created that can be called from a target region
  - if a list item is a variable then the original variable is mapped to a corresponding variable in the initial device data environment for all devices (if the variable is initialized it is mapped with the same value)
  - all declarations and definitions for a function must have a declare target directive

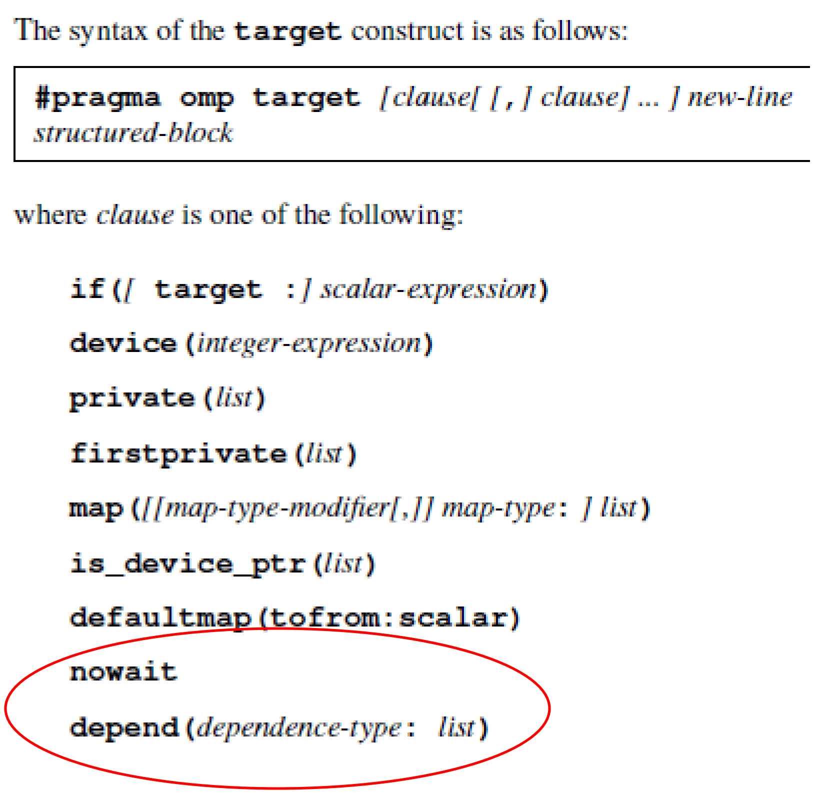
- C/C++:



The syntax of the **declare target** directive is as follows:

#pragma omp declare target *new-line*
declarations-definition-seq
#pragma omp end declare target *new-line*

OpenMP 4.5 – asynchonous execution



The syntax of the **target** construct is as follows:

```
#pragma omp target [clause[ [,] clause] ... ] new-line
structured-block
```

where *clause* is one of the following:

```
if([ target :] scalar-expression)
device(integer-expression)
private(list)
firstprivate(list)
map([[map-type-modifier[,]] map-type: ] list)
is_device_ptr(list)
defaultmap(tofrom:scalar)
nowait
depend(dependence-type: list)
```

- The **nowait** clause indicates that the encountering thread does not wait for the target region to complete.
- A host task is generated that encloses the target region.
- The **depend** clause can be used for synchronization with other tasks

OpenMP 4.5 - Unstructured data movement

- Structured **target data** construct is too restrictive and does not fit for C++ (de)constructors.
- **target enter data**
  - Map variable to a device
- **target exit data**
  - Map variable from a device

C/C++

```
#pragma omp target enter data [clause]


#pragma omp target exit data [clause]
```

- Clauses are if, device, map, depende and nowait with their usual meaning.

Device Pointers

- New clauses
  - **#pragma omp target data … use_device_ptr(list) ..**
  - **#pragma omp target … is_device_ptr(list) ..**

- New API
  - **void* omp_target_alloc(size_t** *size*, **int** *device_num***);**
  - **void omp_target_free(void *** *device_ptr*, **int** *device_num***);**
  - **int omp_target_is_present(void *** *ptr*, **size_t** *offset*, **int** *device_num***);**
  - **int omp_target_memcpy(void *** *dst*, **void *** *src*, **size_t** *length*, **size_t** *dst_offset*, **size_t** *src_offset*, **int** *dst_device_num*, **int** *src_device_num***);**
  - **int omp_target_memcpy_rect(void *** *dst*, **void *** *src*, **size_t** *element_size*, **int** *num_dims*, **const size_t*** *volume*, **const size_t*** *dst_offsets*, **const size_t*** *src_offsets*, **const size_t*** *dst_dimensions*, **const size_t*** *src_dimensions*,**int** *dst_device_num*, **int** *src_device_num***);**
  - **int omp_target_associate_ptr(void *** *host_ptr*, **void *** *device_ptr*, **size_t** *size*, **size_t** *device_offset*, **int** *device_num***);**
  - **int omp_target_disassociate_ptr(void *** *ptr*, **int** *device_num***);**
  - **int omp_get_initial_device (void)**

Thank you for your attention!

Questions?