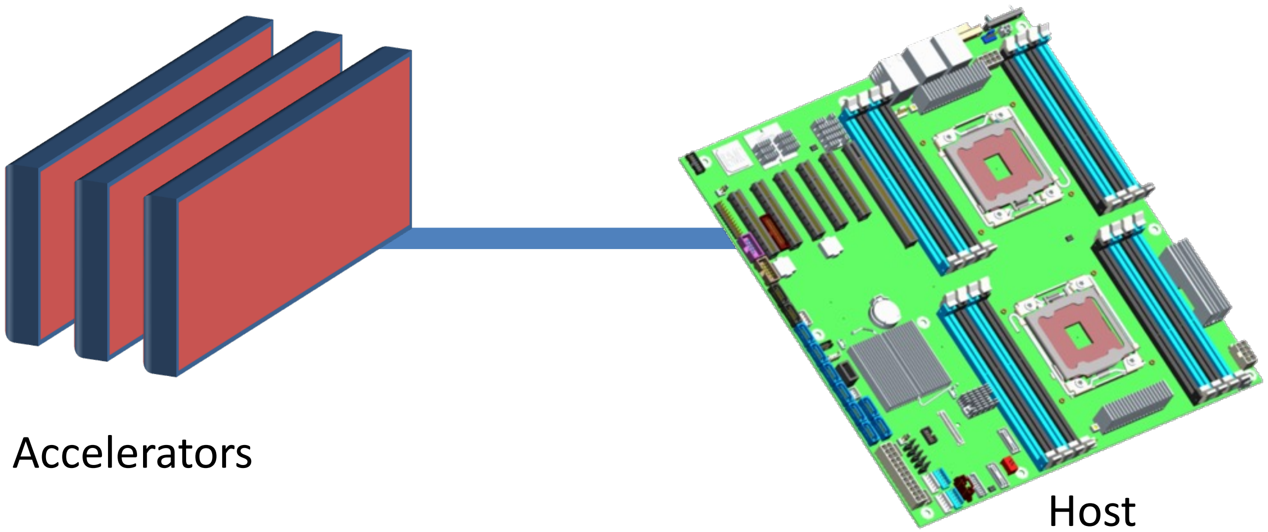
Intro to OpenMP offloading

Lecture 8

Sunita Chandrasekaran
Associate Professor, University of Delaware
PDC Summer School
Aug 2023

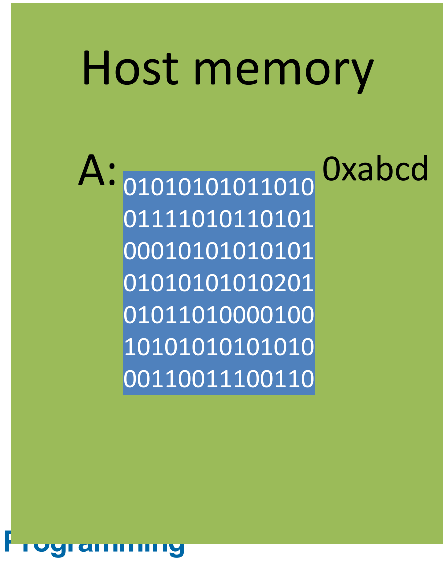
Device Model

- As of version 4.0 the OpenMP API supports accelerators/coprocessors
- Device model:
  - →One host for “traditional” multi-threading
  - →Multiple accelerators/coprocessors of the same kind for offloading



OpenMP Execution Model for Devices

- Offload region and its data environment are bound to the lexical scope of the construct
  - →Data environment is created at the opening curly brace
  - →Data environment is automatically destroyed at the closing curly brace
  - →Data transfers (if needed) are done at the curly braces, too:
    - →Upload data from the host to the target device at the opening curly brace.
    - →Download data from the target device at the closing curly brace.




```
!$omp target            &
!$omp   map(alloc:A) &
!$omp   map(to:A)       &
!$omp   map(from:A)   &
   call compute(A)
!$omp end target
```

OpenMP for Devices - Constructs

- Transfer control and data from the host to the device
- Syntax (C/C++)

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

- Syntax (Fortran)

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

- Clauses

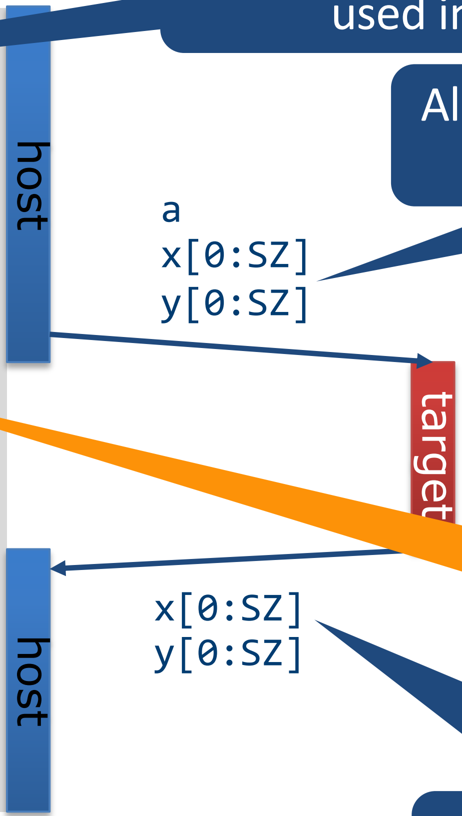
```
device(scalar-integer-expression)
map([{alloc | to | from | tofrom}:] list)
if(scalar-expr)
```

Example: saxpy

```
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);
}
```

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

All accessed arrays are copied from host to device and back



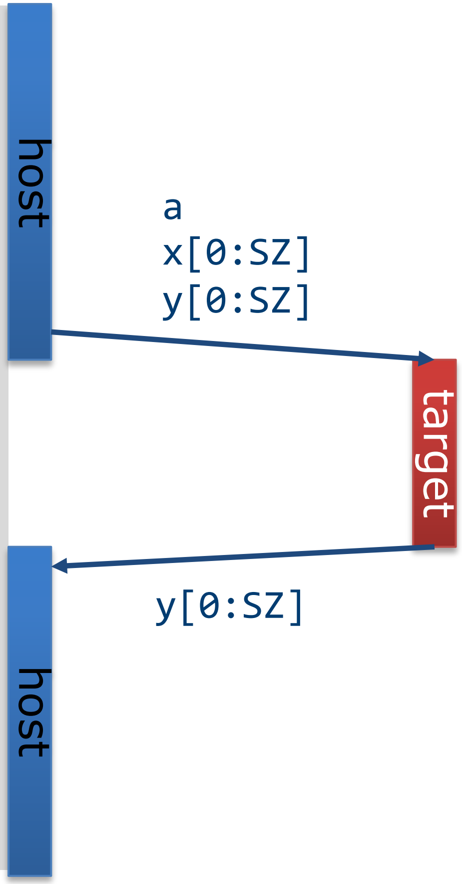
Presence check: only transfer if not yet allocated on the device.

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

```
clang -fopenmp --offload-arch=gfx90a ...
```

Example: saxpy

```
void saxpy() {
    double a, x[SZ], y[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);
}
```

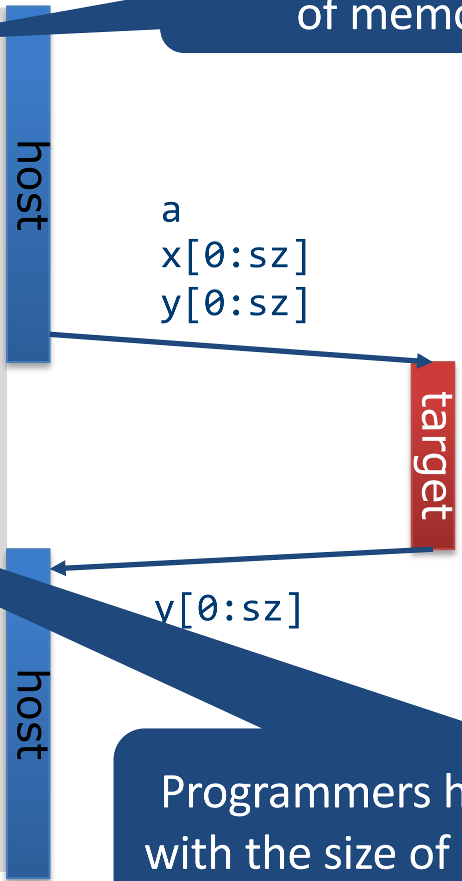



```
clang -fopenmp --offload-arch=gfx90a ...
```

Example: saxpy

```
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 --offload-arch=gfx90a
```

PARALLEL

The parallel construct creates a team of OpenMP threads that execute the region.

```
#pragma omp parallel [clauses]
      structured-block

clause:
num_threads(integer-expression)
default(shared | none)
private(list)
firstprivate(list)
shared(list)
reduction(reduction-identifier : list)
```

OpenMP offloading

The **TARGET** construct consists of a target directive and an execution region. It is used to transfer both the control flow from the host to the device and the data between the host and device.

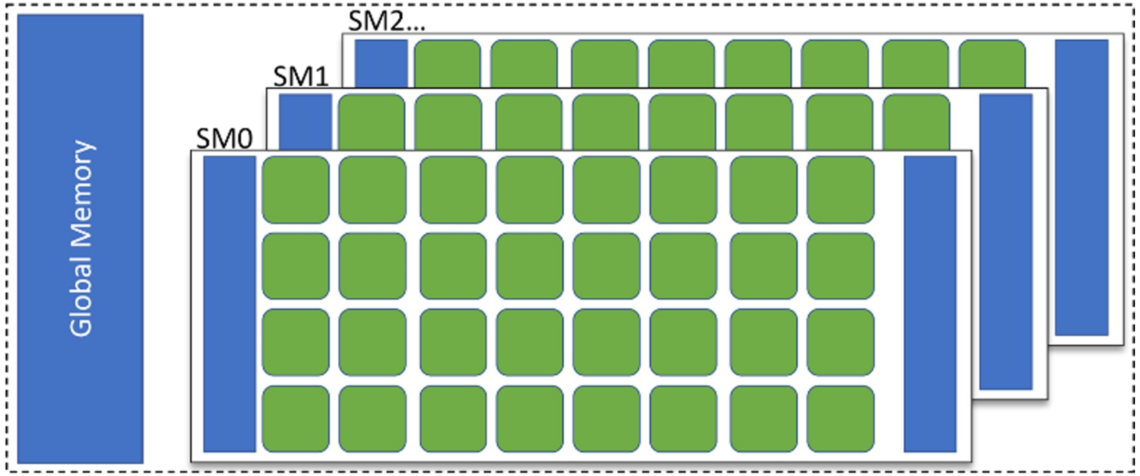
```
#pragma omp target [clauses]
    structured-block

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

OpenMP offloading

OpenMP uses directives in C/C++/Fortran to allow the programmer to define parallelism

Using the target directive (introduced in OpenMP 4.0) specifies that a given code region should be compiled for an offloaded device (such as a GPU)

```
#pragma omp target
{
   < GPU Code>
}
```



Hello World

```c
/* Copyright (c) 2019 CSC Training */
/* Copyright (c) 2021 ENCCS */
#include <stdio.h>

#ifdef _OPENMP
#include <omp.h>
#endif

int main()
{
  int num_devices = omp_get_num_devices();
  printf("Number of available devices %d\n", num_devices);

#pragma omp target
  {
      if (omp_is_initial_device()) {
        printf("Running on host\n");
      } else {
        int nteams= omp_get_num_teams();
        int nthreads= omp_get_num_threads();
        printf("Running on device with %d teams in total and %d threads in each team\n",nteams,nthreads);
      }
  }

}
```

Teams

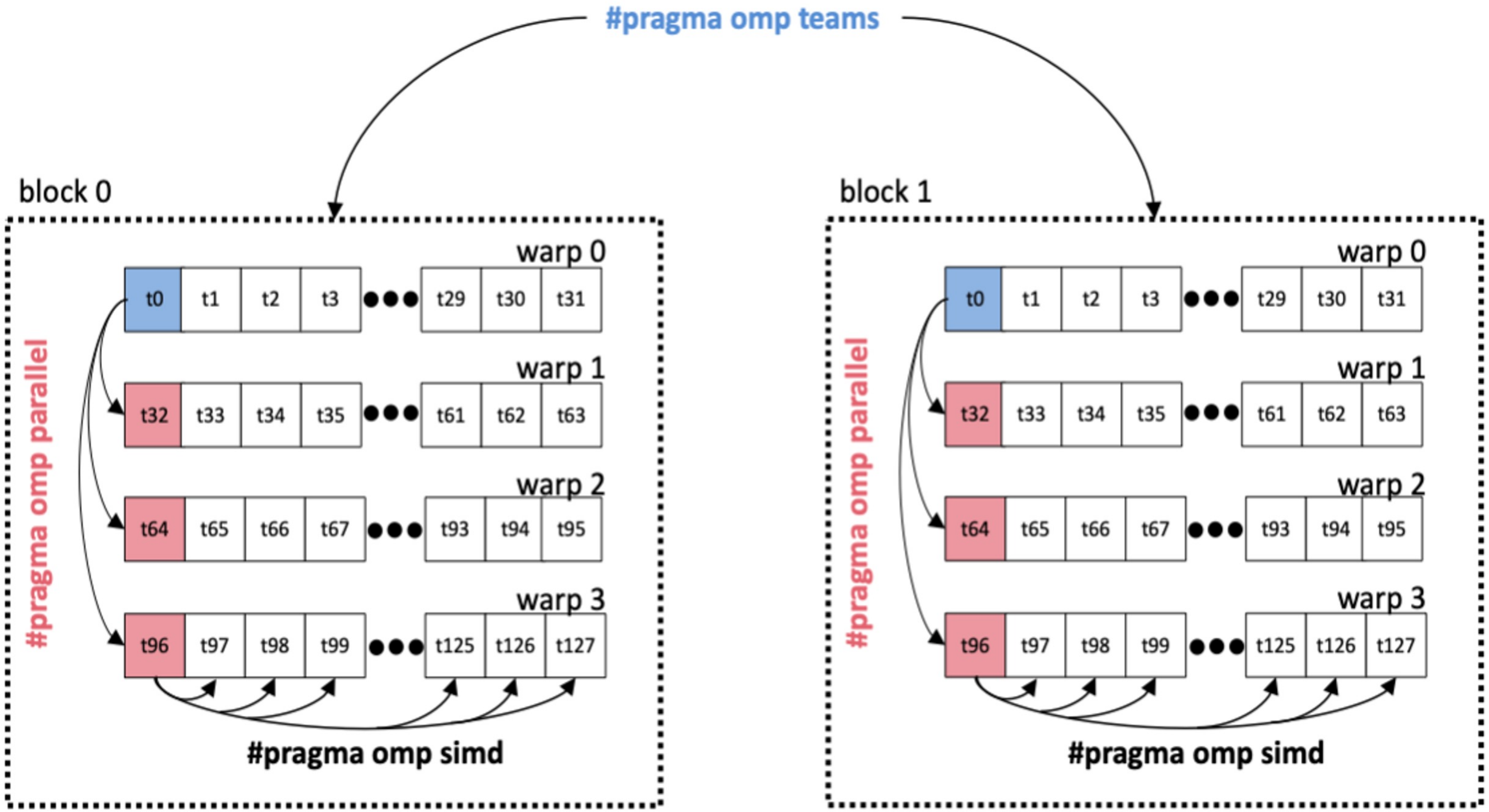
- The TEAMS construct creates a league of one-thread teams where the thread of each team executes concurrently and is in its own contention group.
- The number of teams created is implementation defined, but is no more than num_teams if specified in the clause.
- The maximum number of threads participating in the contention group that each team initiates is implementation defined as well, unless thread_limit is specified in the clause.
- Threads in a team can synchronize but no synchronization among teams.
- The TEAMS construct must be contained in a TARGET construct, without any other directives, statements or declarations in between.

```
#pragma omp teams [clauses]
      Structured-block
```

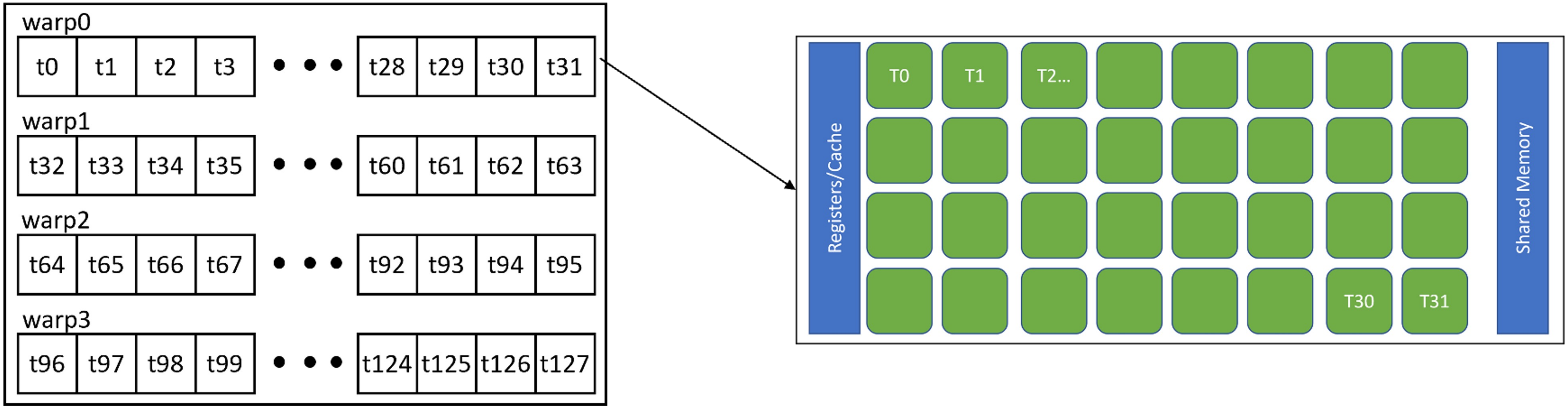
clause:

```
num_teams(integer-expression)
thread_limit(integer-expression)
default(shared | none)
private(list)
firstprivate(list)
shared(list)
reduction(reduction-identifier : list)
```

OpenMP parallelism to GPU hardware



Threads to GPU hardware



DISTRIBUTE

- The DISTRIBUTE construct is a coarsely worksharing construct which distributes the loop iterations across the master threads in the teams, but no worksharing within the threads in one team.
- No implicit barrier at the end of the construct and no guarantee about the order the teams will execute.

```
#pragma omp distribute [clauses]
      for-loops
```

clause:

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

TEAMS DISTRIBUTE construct

- Coarse-grained parallelism
- Spawns multiple single-thread teams
- No synchronization of threads in different teams

TEAMS and DISTRIBUTE and PARALLEL constructs

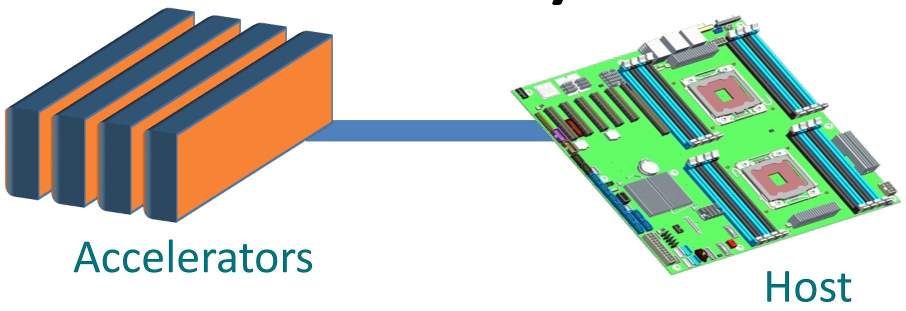
```c
/* Copyright (c) 2019 CSC Training */
/* Copyright (c) 2021 ENCCS */
#include <stdio.h>
#include <math.h>
#define NX 102400

int main(void)
{
  double vecA[NX],vecB[NX],vecC[NX];
  double r=0.2;

/* Initialization of vectors */
  for (int i = 0; i < NX; i++) {
      vecA[i] = pow(r, i);
      vecB[i] = 1.0;
  }

/* dot product of two vectors */

  #pragma omp target teams distribute parallel for

  for (int i = 0; i < NX; i++) {
     vecC[i] = vecA[i] * vecB[i];
  }

  double sum = 0.0;
  /* calculate the sum */
  for (int i = 0; i < NX; i++) {
    sum += vecC[i];
  }
  printf("The sum is: %8.6f \n", sum);
  return 0;
}
```

TEAMS and PARALLEL construct

```
/* Copyright (c) 2019 CSC Training */
/* Copyright (c) 2021 ENCCS */
#include <stdio.h>

#ifdef _OPENMP
#include <omp.h>
#endif

int main()
{
  int num_devices = omp_get_num_devices();
  printf("Number of available devices %d\n", num_devices);
```

```
#pragma omp target

  #pragma omp teams num_teams(2) thread_limit(3)

  #pragma omp parallel

  {
      if (omp_is_initial_device()) {
        printf("Running on host\n");
      } else {
        int nteams= omp_get_num_teams();
        int nthreads= omp_get_num_threads();
        printf("Running on device with %d teams in total and %d threads in each team\n",nteams,nthreads);
      }
  }

}
```

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.

Data Mapping

- The MAP clause on a device construct explicitly specifies how items are mapped from the host to the device data environment.
- The common mapped items consist of arrays(array sections), scalars, pointers, and structure elements.
- The various forms of the map clause are summarised in the following table

MAP clauses

`map([map-type]:list)`	map clause
`map(to:list)`	On entering the region, variables in the list are initialized on the device using the orig
`map(from:list)`	At the end of the target region, the values from variables in the list are copied into th
`map(tofrom:list)`	the effect of both a map-to and a map-from
`map(alloc:list)`	On entering the region, data is allocated and uninitialized on the device
`map(list)`	equivalent to ``map(tofrom:list)``

Role of the Presence Check

- If `map` clauses are not added to `target` constructs, presence checks determine if data is already available in the device data environment:

```
subroutine saxpy(a, x, y, n)
    use iso_fortran_env
    integer :: n, i
    real(kind=real32) :: a
    real(kind=real32), dimension(n) :: x
    real(kind=real32), dimension(n) :: y

!$omp target
    do i=1,n
        y(i) = a * x(i) + y(i)
    end do   "present?(y)" "present?(x)"
!$omp end target
end subroutine
```

  – OpenMP maintains a mapping table that records what memory pointers have been mapped.
  – That table also maintains the translation between host memory and device memory.
  – Constructs with no `map` clause for a data item then determine if data has been mapped and if not, a `map(tofrom:…)` is added for that data item.

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.

```
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];
}
```

TEAMS and DISTRIBUTE and PARALLEL constructs

```c
/* Copyright (c) 2019 CSC Training */
/* Copyright (c) 2021 ENCCS */
#include <stdio.h>
#include <math.h>
#define NX 102400

int main(void)
{
  double vecA[NX],vecB[NX],vecC[NX];
  double r=0.2;

/* Initialization of vectors */
  for (int i = 0; i < NX; i++) {
      vecA[i] = pow(r, i);
      vecB[i] = 1.0;
  }

/* dot product of two vectors */

  #pragma omp target teams distribute
map(from:vecC[0:NX])
map(to:vecA[0:NX],vecB[0:NX])

  for (int i = 0; i < NX; i++) {
     vecC[i] = vecA[i] * vecB[i];
  }

  double sum = 0.0;
  /* calculate the sum */

  #pragma omp target map(tofrom:sum)

  for (int i = 0; i < NX; i++) {
    sum += vecC[i];
  }
  printf("The sum is: %8.6f \n", sum);
  return 0;
}
```

`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)
```

Create a data region using TARGET DATA and add map clauses for data transfer.

```c
/* Copyright (c) 2019 CSC Training */
/* Copyright (c) 2021 ENCCS */
#include <stdio.h>
#include <math.h>
#define NX 102400

int main(void)
{
  double vecA[NX],vecB[NX],vecC[NX];
  double r=0.2;

/* Initialization of vectors */
  for (int i = 0; i < NX; i++) {
     vecA[i] = pow(r, i);
     vecB[i] = 1.0;
  }

/* dot product of two vectors */

  #pragma omp target data map(from:vecC[0:NX])
  {
     #pragma omp target map(to:vecA[0:NX],vecB[0:NX])

     for (int i = 0; i < NX; i++) {
        vecC[i] = vecA[i] * vecB[i];
     }

/* Initialization of vectors again */
     for (int i = 0; i < NX; i++) {
        vecA[i] = 0.5;
        vecB[i] = 2.0;
     }


     #pragma omp target map(to:vecA[0:NX],vecB[0:NX])

     for (int i = 0; i < NX; i++) {
        vecC[i] = vecC[i] + vecA[i] * vecB[i];
     }

  }

  double sum = 0.0;
  /* calculate the sum */
  for (int i = 0; i < NX; i++) {
    sum += vecC[i];
  }
  printf("The sum is: %8.6f \n", sum);
  return 0;
}
```

`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 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

TARGET enter and exit data

#pragma omp target enter data

The omp target enter data directive maps variables to a device data environment.
The omp target enter data directive can reduce data copies to and from the offloading device when multiple target regions are using the same data.

#pragma omp target exit data

The omp target exit data directive unmaps variables from a device data environment. The omp target exit data directive can limit the amount of device memory when you use the omp target enter data construct to map items to the device data environment.

Take home code to play with

What does EP do?

What was our experimental setup?

EP on KTH system

- AMD + offloading + MI250x

EP on an NVIDIA system

- NVHPC SDK + A100

Discuss performance differences between both the runs

Profilers

EP + NVIDIA profilers

EP + AMD profilers

What to look for?