Introduction to OpenMP Device Offload: Data Movement

Swaroop Pophale
Computer Scientist, CSMD, ORNL

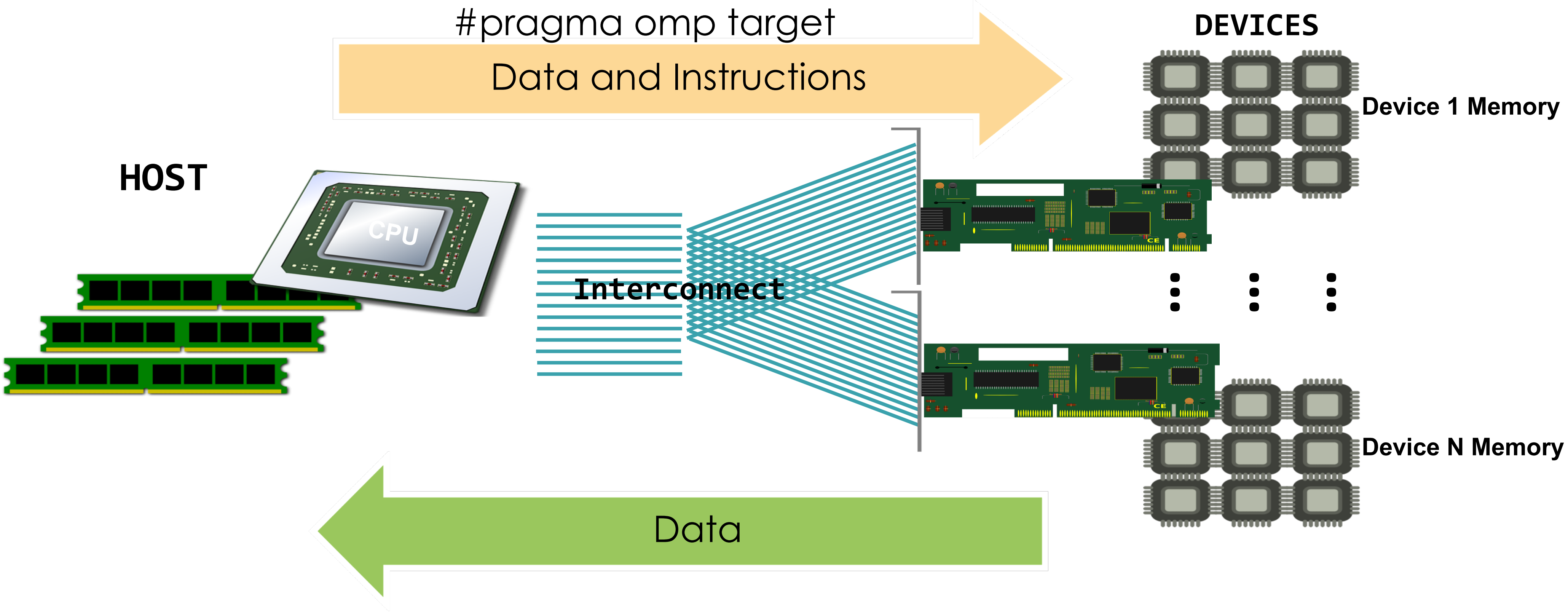
ORNL is managed by UT-Battelle, LLC for the US Department of Energy

Outline

- Recap OpenMP Offload – the **target** construct
- Target construct
- Device Data Persistence
- OpenMP Device Data Directives
- Allocating Memory on the Device
- Target Update construct
- Declare Target
- Unified Shared Memory
- Hands On

Recap: OpenMP Offload

- OpenMP offload constructs were introduced in OpenMP 4.0 and further enhanced in later versions.



Recap: Device Execution Directives

- #pragma omp target
- #pragma omp target teams
- #pragma omp target teams distribute
- #pragma omp target teams distribute parallel for
- #pragma omp target parallel for
- #pragma omp target parallel loop*
- #pragma omp target teams loop*
- #pragma omp target simd
- #pragma omp target parallel for simd
- #pragma omp target teams distribute simd
- #pragma omp target teams distribute parallel for simd

*Currently loop variations are NOT optimal on Frontier

Target Directive Clauses

- Data/memory related clauses allowed on the target directive:
  - **map**
  - **is_device_ptr**
  - has_device_addr
  - defaultmap
  - allocate
  - uses_allocators*

*Currently not completely supported on Frontier

Target construct: the Map Clause

Syntax: map([[map-type-modifier[,] [map-type-modifier[,] ...] map-type :] locator-list)

“The map clause specifies how an original list item is mapped from the current task’s data environment to a corresponding list item in the device data environment of the device identified by the construct.”

Map Clause: Map Types

- to - *allocates data and moves data to the device*
- from – *allocates data and moves data from the device*
- tofrom – *allocates data and moves data to and from the device*
- alloc - *allocates data on the device*
- release – *marks data storage as ”no longer required”*
- delete – *deletes the data from the device*

Example using omp `target`

/*C code to offload Matrix Addition Code to Device*/

```
  …
  int A[N][N], B[N][N], C[N][N];
  /*
    initialize arrays
  */
#pragma omp target
  {
    for (int i = 0; i < N; ++i) {
      for (int j = 0; j < N; ++j) {
        C[i][j] = A[i][j] + B[i][j];
      }
    }
  } // end target
```

Will this work ?

Yes

Is this efficient ?

No

Target construct: Implicit Mapping Rules

- C/C++: Pointer is treated as if it is the base pointer of a zero-length array section is a list item in a map clause.
- Fortran: If a scalar variable has the TARGET, ALLOCATABLE or POINTER attribute then it is treated as if it is is a list item in a map clause with a map-type of tofrom.
- C/C++/Fortran:
  - If a variable is not a **scalar** then it is treated as if it is mapped with a map-type of tofrom.
  - Scalars variables are implicitly **firstprivate**

OpenMP Offload: Example using omp `target`

/*C code to offload Matrix Addition Code to Device with map clause using static arrays*/

```c
…
int A[N][N], B[N][N], C[N][N];
/*
  initialize arrays
*/
#pragma omp target map(to: A, B) map(from: C)
{
  for (int i = 0; i < N; ++i) {
    for (int j = 0; j < N; ++j) {
      C[i][j] = A[i][j] + B[i][j];
    }
  }
} // end target
```

/*C code to offload Matrix Addition Code to Device with map clause using dynamic arrays*/

```c
…
int *A, *B, *C;
/*
  allocate arrays of size N and initialize
*/
#pragma omp target map(to: A[0:N], B[0:N])
map(from: C[0:N])
{
  for (int i = 0; i < N; ++i) {
    for (int j = 0; j < N; ++j) {
      C[i][j] = A[i][j] + B[i][j];
    }
  }
} // end target
```

Array Sections

Array Sections in OpenMP

- An array section designates a subset of the elements in an array.

  [lower-bound : length : stride]

- Must be a subset of the original array.
- Array sections are allowed on multidimensional arrays.
- Must be integers or integer expressions
  - The **length** must evaluate to a non-negative integer and must be explicitly specified when the size of the array dimension is not known
  - The **stride** must evaluate to a positive integer, default 1
  - lower-bound when absent it defaults to 0.

Device Data Persistence

/*C code to offload Matrix Addition Code to Device*/

```
  …
  int A[N][N], B[N][N], C[N][N];
  /*
    initialize arrays
  */
#pragma omp target → Arrays are copied to the device
  {
    for (int i = 0; i < N; ++i) {
      for (int j = 0; j < N; ++j) {
        C[i][j] = A[i][j] + B[i][j];
      }
    }
  } // end target → Arrays are copied back to the host
...
...
```

Device Data Persistence: Offloading Multiple kernels

/*C code for multiple offload kernels */

```
  …
#pragma omp target map(to: A, B) map(from: C)
  {
    for (int i = 0; i < N; ++i) {
      for (int j = 0; j < N; ++j) {
        C[i][j] = A[i][j] + B[i][j];
      }
    }
  }

/*
Some computation using C (no changes to A, B or C)
*/

#pragma omp target map(to: A, B, C) map(from: D)
  {
    for (int i = 0; i < N; ++i) {
      for (int j = 0; j < N; ++j) {
        D[i][j] = A[i][j] + B[i][j] C[i][j];
      }
    }
  }
…
…
```

Is this optimal ?

NO

A and B are unchanged between the two target regions.

OpenMP Device Data Directives

C/C++	Fortran	Description
#pragma omp target data *clause[[[,] clause] ...]* *new-line* *structured-block*	**!$omp target data** *clause[[[,] clause] ...]* *Loosely/tightly-structured-block* **!$omp end target data**	The target data construct maps variables to a device data environment for the extent of the region using the **map** clause.
#pragma omp target enter data [*clause[[,] clause]* ...] *new-line*	**!$omp target enter data** [*clause[[,] clause]*]	A **standalone directive** that specifies that variables are mapped to a device data environment. It does so via a **map** clause
#pragma omp target exit data [*clause[[,] clause]* ...] *new-line*	**!$omp target exit data** [*clause[[,] clause]*]	A **standalone directive** that specifies that variables are unmapped from a device data environment via a **map** clause

Multiple offload kernels using **target data map**

/*C code for multiple offload kernels with structured data mapping using target data map*/

```
  …
#pragma omp target data map(to: A, B)
{
#pragma omp target map(from: C)
  {
    for (int i = 0; i < N; ++i) {
      for (int j = 0; j < N; ++j) {
        C[i][j] = A[i][j] + B[i][j];
      }end-for
    }end-for
  } end target

/*
Some computation on host using C (no changes to A, B or C)
*/

#pragma omp target map(to: C) map(from: D)
  {
    for (int i = 0; i < N; ++i) {
      for (int j = 0; j < N; ++j) {
        D[i][j] = A[i][j] + B[i][j] C[i][j];
      }
    }
  }
}//end target-data
…
…
```

Multiple offload kernels using target enter/exit data

/*C code for multiple offload kernels using target enter/exit data map*/

```
foo(){
   …
#pragma omp target enter data map(to: A, B)

}

main(){
...
...

foo();

#pragma omp target map(from: C)
  {
    for (int i = 0; i < N; ++i) {
      for (int j = 0; j < N; ++j) {
        C[i][j] = A[i][j] + B[i][j];
      }end-for
    }end-for
  } end target
...

bar();
...
 }
```

```
bar(){
...
}
#pragma omp target map(to: C)   {
    for (int i = 0; i < N; ++i) {
      for (int j = 0; j < N; ++j) {
        D[i][j] = A[i][j] + B[i][j] C[i][j];
      }
    }
  }

#pragma omp target exit data map(release: C)
map(from: D)
…
}
```

Allocating Memory on the Device

C/C++	Fortran	Description
void* omp_target_alloc(size_t size, int device_num);	type(c_ptr) function omp_target_alloc(size, device_num) bind(c) use, intrinsic :: iso_c_binding, only : c_ptr, c_size_t, c_int integer(c_size_t), value :: size integer(c_int), value :: device_num	routine allocates memory in a device data environment and returns a device pointer to that memory
void omp_target_free(void *device_ptr, int device_num);	subroutine omp_target_free(device_ptr, device_num) bind(c) use, intrinsic :: iso_c_binding, only : c_ptr, c_int ..	routine frees the device memory allocated by the omp_target_alloc routine.

- The omp_target_alloc routine returns a device pointer that references the device address of a storage location of size bytes.
- The storage location is dynamically allocated in the device data environment of the device specified by device_num.

Accessing device data: is_device_ptr clause

- The is_device_ptr clause indicates that its list items are device pointers
- For C++ list item in an is_device_ptr clause must be:
  - type of pointer or array,
  - reference to pointer or reference to array
- For C it must have a type of pointer or array.
- For Fortran the list item in an is_device_ptr clause must be of type C_PTR
- Support for device pointers created outside of OpenMP is implementation defined
- is_device_ptr clause is not necessary when using ***requires unified_address***

is_device_ptr: How to use it

```
/*C code for example of is_device_ptr*/

int *array_device = NULL;
int *array_host = NULL;

array_device = (int *) omp_target_alloc(BIG_SIZE, omp_get_default_device());

array_host = (int *) malloc(SIZE);

#pragma omp target is_device_ptr(array_device) map(from: array_host[0:N])
{
    /*Extensive work on array_device and only copy back relevant data to array_host */
}
…
…
```

The target update construct

- Syntax

C/C++ : #pragma omp target update clause[[[,] clause] ...] new-line

Fortran: !$omp target update clause[[[,] clause] ...]

- The target update directive makes the corresponding list items in the device data environment consistent with their original list items, according to the specified data-motion-clauses.

target update example 1

/*C code for multiple offload kernels using target data map and target update*/

```
  …
#pragma omp target data map(to: A, B) map(alloc: C, D)
{

#pragma omp target
  {
    for (int i = 0; i < N; ++i) {
      for (int j = 0; j < N; ++j) {
        C[i][j] = A[i][j] + B[i][j];
      }
    }
  }
#pragma omp target update from(C)                    //Updates C device → host
/*
Some computation using C on host (no changes to A, B or C)
*/

#pragma omp target map(from: D)
  {
    for (int i = 0; i < N; ++i) {
      for (int j = 0; j < N; ++j) {
        D[i][j] = A[i][j] + B[i][j] C[i][j];
      }
    }
  }
}//end target-data
…
…
```

target update example 2

/*C code for multiple offload kernels using target data map and target update*/

```
  …
#pragma omp target data map(to: A, B) map(alloc: C, D)
{

#pragma omp target
  {
    for (int i = 0; i < N; ++i) {
      for (int j = 0; j < N; ++j) {
        C[i][j] = A[i][j] + B[i][j];
      }
    }
  }
#pragma omp target update from(C)                    //Updates C device → host
/*
Some changes to A (no changes to B or C)
*/
#pragma omp target update to(A)                      //Updates A host → device

#pragma omp target map(from: D)
  {
    for (int i = 0; i < N; ++i) {
      for (int j = 0; j < N; ++j) {
        D[i][j] = A[i][j] + B[i][j] C[i][j];
      }
    }
  }
}//end target-data
```

Declare target directive

- Syntax

C/C++ : #pragma omp declare target

Fortran: !$omp declare target

- Declare target directives apply to procedures and/or variables to ensure that they can be executed or accessed on a device.

Declare target example

```
/*C code for demonstrating use of declare target*/

#pragma omp begin declare target
int a[N], b[N], c[N];
int i = 0;
#pragma omp end declare target

void foo() {
  for (i = 0; i < N; i++) {
    /*update a, b, and c*/
  }
}

#pragma omp declare target (foo)

int main() {
  foo();
  #pragma omp target update to(a,b,c)

#pragma omp target
  {
    foo();
  }
  #pragma omp target update from( a, b, c)
```

Other Device Memory Routines

C/C++	Fortran	Description
int omp_target_is_present(const void *ptr, int device_num);	integer(c_int) function omp_target_is_present(ptr, device_num) & bind(c) use, intrinsic :: iso_c_binding, only : c_ptr, c_int ..	routine tests whether a host pointer refers to storage that is mapped to a given device.
int omp_target_is_accessible(const void *ptr, size_t size, int device_num);	integer(c_int) function omp_target_is_accessible(& ptr, size, device_num) bind(c) use, intrinsic :: iso_c_binding, only : c_ptr, c_int ..	routine tests whether host memory is accessible from a given device.
int omp_target_memcpy(void *dst,..);	**integer(c_int) function omp_target_memcpy(dst, src, length, & dst_offset, src_offset, dst_device_num, src_device_num) bind(c)** **use, intrinsic :: iso_c_binding, only : c_ptr, c_int, c_size_t ..**	**routine copies memory between host and device pointers.**

Other Device Memory Routines (cont.)

C/C++	Fortran	Description
int omp_target_memcpy_rect(..);	integer(c_int) function omp_target_memcpy_rect(dst,src,element_size, & .	copies a rectangular sub-volume from a multi-dimensional array to another multi-dimensional array.
int omp_target_memcpy_async(...);	integer(c_int) function omp_target_memcpy_async(..	performs asynchronous copy between host and device pointers.
int omp_target_memcpy_rect_async(...);	integer(c_int) function omp_target_memcpy_rect_async(...	asynchronously performs a copy between host and device pointers.
int omp_target_associate_ptr(...);	integer(c_int) function omp_target_associate_ptr(...	routine maps a device pointer to a host pointer
int omp_target_disassociate_ptr(...);	integer(c_int) function omp_target_disassociate_ptr(..	routine removes the associated pointer for a given device from a host pointer.
void * omp_get_mapped_ptr (...);	type(c_ptr) function omp_get_mapped_ptr(...	routine returns the device pointer that is associated with a host pointer for a given device.

Unified Shared Memory

Single address space over CPU and GPU memories

- An implementation to guarantees that all devices accessible through OpenMP API routines and directives use a unified address space.
- A pointer will always refer to the same location in memory from all devices accessible through OpenMP.
- Any OpenMP mechanism that returns a device pointer is guaranteed to return a device address that supports pointer arithmetic, and the is_device_ptr clause is not necessary
- Host pointers may be passed as device pointer arguments to device memory routines and device pointers may be passed as host pointer arguments to device memory routines.

Unified Shared Memory

* Enforced by the requires directive

```
int *a, *b, *c;
/*allocate and initialize arrays a,b, c*/

#pragma omp requires unified_shared_memory

// No data directive or mapping needed for pointers a, b, c
#pragma omp target teams distribute parallel for
  for (int i=0; i < N; i++) {
    c[i] = a[i] + b[i];
  }
```


28

References

- Examples were adapted from: https://github.com/SOLLVE/sollve_vv
- OpenMP Specification 5.1
- https://www.nas.nasa.gov/hecc/assets/pdf/training/OpenMP4.5_3-20-19.pdf

Map: Reference count

- On entry to device environment:
  - If a corresponding list item is not present in the device data environment, then:
    - A new list item corresponding to original list item (on host) is created in the device data environment;
    - The corresponding list item has a reference count that is initialized to zero; and
    - The value of the corresponding list item is undefined;
  - If ref count is not incremented due to map clause, it is incremented by 1

Atomic Operation

- On exit from device environment:
  - if map-type is **delete** ref count is set to 0
  - if map-type is not **delete** the ref count is decremented by 1 (min 0)
  - If the reference count is zero then the corresponding list item is removed from the device data environment.

Atomic Operation