Introduction to OpenMP Device Offload – Day 2

Swaroop Pophale
Computer Scientist, CSMD, ORNL
Outline

• Recap: Day1 Summary
• Map clause
• Is_device_ptr clause
• OpenMP Device Data Directives
• 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`
- `#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`
Target Directive Clauses

- Clauses allowed on the target directive:
  - `if([ target :] scalar-expression)
  - `device([ device-modifier :] integer-expression)
  - `thread_limit(integer-expression)
  - `private(list)
  - `firstprivate(list)
  - `in_reduction(reduction-identifier : list)
  - `map([[[map-type-modifier[,] [map-type-modifier[,] ...]] map-type:] ] locator-list)
  - `is_device_ptr(list)
  - `has_device_addr(list)
  - `defaultmap(implicit-behavior[:variable-category])
  - `nowait
  - `depend([depend-modifier,] dependence-type : locator-list)
  - `allocate([allocator :] list)
  - `uses_allocators(allocator[([allocator-traits-array]) ,allocator([allocator-traits-array]) ...])
Mapping: 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
Implicit Mapping Rules on Target

- **C/C++**: Pointer is treated as if it is the base pointer of a zero-length array section that had appeared as a list item in a map clause.

- **All**: If a variable is not a scalar then it is treated as if it was mapped with a map-type of tofrom.

- **Fortran**: If a scalar variable has the TARGET, ALLOCATABLE or POINTER attribute then it is treated as if it had appeared in a map clause with a map-type of tofrom.

- **All**: Scalars variables are implicitly firstprivate (not mapped)
The Map Clause

Syntax: \texttt{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 – *decrements the reference count of a variable*
• delete – *deletes the data from the device*
Map Clause: Map-type Modifiers

• always – value of list item is always copied to (for to and tofrom) and from device (for from and tofrom)

• close - is a hint to the runtime to allocate memory close to the target device

• mapper(mapper-identifier) – use user-defined mapper(defined before in program)

• present - clause is ordered before all other map clauses

• iterator(iterators-definition) - to reference iterators
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

- 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.
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
```
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(N*sizeof(int), omp_get_default_device());

array_host = (int *) malloc(N*sizeof(int));
/* initialize array_host */

#pragma omp target is_device_ptr(array_device) map(tofrom: array_host[0:N])
{
    for (int i = 0; i < N; ++i) {
        array_device[i] = i;
        array_host[i] += array_device[i];
    }
}
...
...
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

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

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

```c
... #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*/

...#pragma omp target enter 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 target

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

#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)

target update construct

• Syntax
  – C/C++ : #pragma omp target update clause[ [ [,] clause] ... ] new-line
  – Fortran: !$omp target update clause[ [ [,] clause] ... ]

• Where clause can be:
  – if([ target update :] scalar-expression)
  – device(integer-expression)
  – nowait
  – depend([depend-modifier,) dependence-type : locator-list)
  – to([motion-modifier[,] [motion-modifier[,] ...]: ] locator-list)
  – from([motion-modifier[,] [motion-modifier[,] ...]: ] locator-list)
target update example 1

```c
/*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
```
/*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

<table>
<thead>
<tr>
<th>C/C++</th>
<th>Fortran</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>#pragma omp declare target new-line declaration-definition-seq #pragma omp end declare target new-line OR #pragma omp declare target (extended-list) new-line OR #pragma omp declare target clause[ [ [,] clause] ... ] new-line OR #pragma omp begin declare target [clause[ [,] clause] ... ] new-line declaration-definition-seq #pragma omp end declare target new-line</td>
<td>!$omp declare target (extended-list) OR !$omp declare target [clause[ [,] clause] ... ]</td>
<td>The declare target directive specifies that variables, functions (C, C++ and Fortran), and subroutines (Fortran) are mapped to a device. The declare target directive is a declarative directive.</td>
</tr>
</tbody>
</table>
Declare target example

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

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

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

#pragma omp declare target to(update)

int main() {
    update();
    #pragma omp target update to(a,b,c)
    #pragma omp target
    {
        update();
    }
    #pragma omp target update from( a, b, c)
### Device Memory Routines

<table>
<thead>
<tr>
<th>C/C++</th>
<th>Fortran</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>void* omp_target_alloc(size_t size, int device_num);</td>
<td>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</td>
<td>routine allocates memory in a device data environment and returns a device pointer to that memory</td>
</tr>
<tr>
<td>void omp_target_free(void *device_ptr, int device_num);</td>
<td>subroutine omp_target_free(device_ptr, device_num) bind(c) use, intrinsic :: iso_c_binding, only : c_ptr, c_int ..</td>
<td>routine frees the device memory allocated by the omp_target_alloc routine.</td>
</tr>
<tr>
<td>int omp_target_is_present(const void *ptr, int device_num);</td>
<td>integer(c_int) function omp_target_is_present(ptr, device_num) &amp; bind(c) use, intrinsic :: iso_c_binding, only : c_ptr, c_int ..</td>
<td>routine tests whether a host pointer refers to storage that is mapped to a given device.</td>
</tr>
<tr>
<td>int omp_target_is_accessible(const void *ptr, size_t size, int device_num);</td>
<td>integer(c_int) function omp_target_is_accessible( &amp;ptr, size, device_num) bind(c) use, intrinsic :: iso_c_binding, only : c_ptr, c_int ..</td>
<td>routine tests whether host memory is accessible from a given device.</td>
</tr>
<tr>
<td>int omp_target_memcpy(void *dst,..);</td>
<td>integer(c_int) function omp_target_memcpy(dst, src, length, &amp; 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 ..</td>
<td>routine copies memory between host and device pointers.</td>
</tr>
<tr>
<td>C/C++</td>
<td>Fortran</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------</td>
<td>-------------------------------------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td><code>int omp_target_memcpy_rect(..);</code></td>
<td><code>integer(c_int) function omp_target_memcpy_rect(dst,src,element_size, &amp;)</code></td>
<td>copies a rectangular subvolume from a multi-dimensional array to another multi-dimensional array.</td>
</tr>
<tr>
<td><code>int omp_target_memcpy_async(…);</code></td>
<td><code>integer(c_int) function omp_target_memcpy_async( …)</code></td>
<td>routine asynchronously performs a copy between host and device pointers.</td>
</tr>
<tr>
<td><code>int omp_target_memcpy_rect_async(…);</code></td>
<td><code>integer(c_int) function omp_target_memcpy_rect_async( …)</code></td>
<td>routine asynchronously performs a copy between host and device pointers.</td>
</tr>
<tr>
<td><code>int omp_target_associate_ptr(…);</code></td>
<td><code>integer(c_int) function omp_target_associate_ptr(…)</code></td>
<td>routine maps a device pointer to a host pointer.</td>
</tr>
<tr>
<td><code>int omp_target_disassociate_ptr(…);</code></td>
<td><code>integer(c_int) function omp_target_disassociate_ptr(…)</code></td>
<td>routine removes the associated pointer for a given device from a host pointer.</td>
</tr>
<tr>
<td><code>void * omp_get_mapped_ptr (…);</code></td>
<td><code>type(c_ptr) function omp_get_mapped_ptr( …)</code></td>
<td>routine returns the device pointer that is associated with a host pointer for a given device.</td>
</tr>
</tbody>
</table>
Unified Shared Memory

Single address space over CPU and GPU memories

```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];
  }
```
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

• OpenMP Discussion @ 2021 Exascale Computing Project Virtual Annual Meeting (April 12 – 16, 2021)