OAK RIDGE National Laboratory

OpenMP Offload 101

August 5th , 2025

Swaroop Pophale (CSMD), Reuben Budiardja (NCCS), Wael Elwasif (CSMD), Suzanne Parete-Koon (NCCS)

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

U.S. DEPARTMENT OF ENERGY

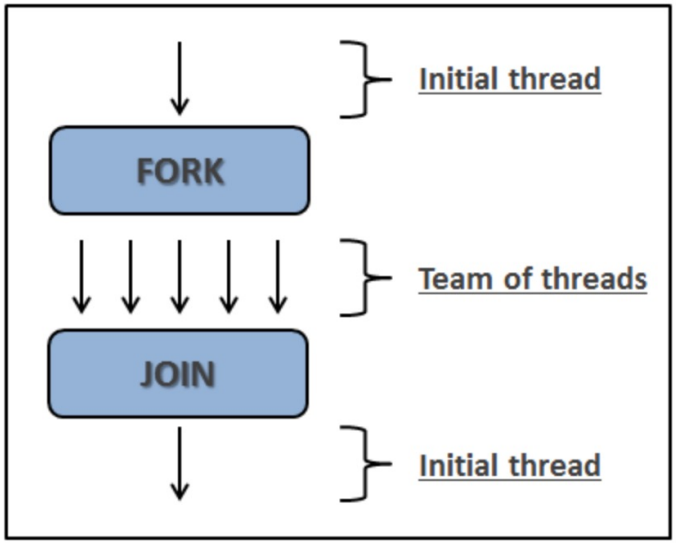
Agenda

Time	Activity
1:30 – 2:20	**OpenMP Offload 101** • Intro to OpenMP Offload • Preview of OpenMP 6.0 *Swaroop Pophale*
2:20 – 3:00	**Hands-On Session** • Intro to hands-on and exercises *Wael Elwasif*

OpenMP Programming Model

It is an Application Program Interface (API) to allow programmers to develop threaded parallel codes on shared memory computational units.

- Directives are understood by OpenMP aware compilers (others are free to ignore)
- Generates parallel threaded code
  - Original thread becomes thread "0"
  - Share resources of the original thread (or rank)
  - Data-sharing attributes of variables can be specified based on usage patterns
- Current status
  - Version 5.2 (November 2021)
  - Version 6.0 (November 2024)



Reference: Somewhere from the web

Recap: OpenMP Worksharing

Serial

```
for (int i = 0; i < N; ++i)
{
    C[i] = A[i] + B[i];
}
```

- 1 thread/process will execute each iteration sequentially
- Total time = time_for_single_iteration * N

Parallel

```
#pragma omp parallel
for (int i = 0; i < N; ++i)
{
    C[i] = A[i] + B[i];
}
```

- Say, OMP_NUM_THREADS = 4
- 4 threads will execute each iteration sequentially (overwriting values of C)
- Total time = time_for_single_iteration * N

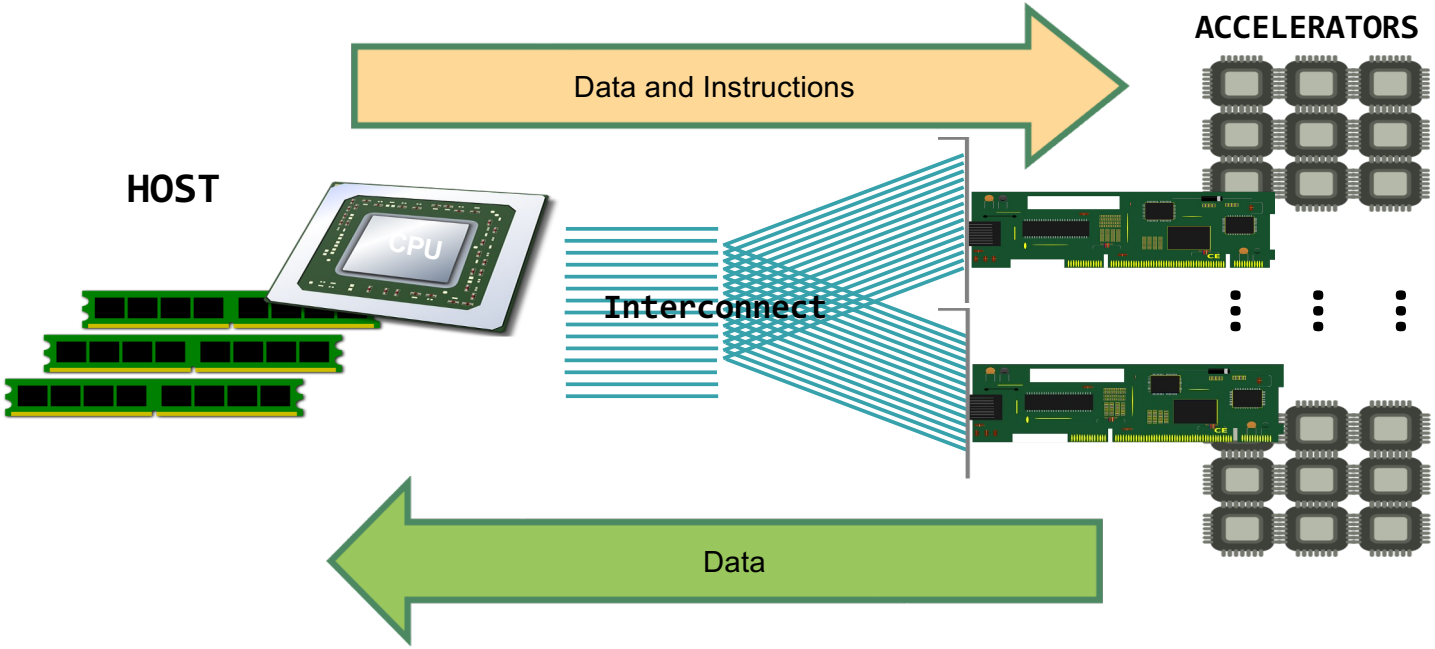
Parallel Worksharing

```
#pragma omp parallel for
for (int i = 0; i < N; ++i)
{
    C[i] = A[i] + B[i];
}
```

- Say, OMP_NUM_THREADS = 4
- 4 threads will distribute iteration space (roughly N/4 per thread)
- Total time = time_for_single_iteration * N/4

OpenMP Offload

- OpenMP offload constructs are a set of directives for C, C++, and Fortran that were introduced in OpenMP 4.0 and further enhanced in later versions. Accelerators



OpenMP Offload Terminology

- **Host device**
  - The device on which the OpenMP program begins execution.
- **Target device**
  - A device with respect to which the current device performs an operation, as specified by a device construct or an OpenMP device memory routine.
- **Parent device**
  - For a given target region, the device on which the corresponding target construct was encountered.
    - A host device may not always be the parent.

NOTE: This presentation adheres to OpenMP Specification 5.2

OpenMP Offload: Steps

- **Identification** of compute kernels
  - CPU initiates kernel for execution on the device
- Expressing **parallelism** within the kernel
- Manage **data transfer** between CPU and Device
  - relevant data needs to be moved from host to device memory
  - kernel executes using device memory
  - relevant data needs to be moved from device to main memory

Identification of Kernels to Offload

- Look for compute intensive code and that can benefit from parallel execution
  - Use performance analysis tools to find bottlenecks/computationally intensive kernels
  - Track independent work units with well defined data accesses
- Keep an eye on platform specs
  - GPU memory is a precious resource
- Confirm via Profiling
  - Tools like rocprof and HPCToolkit

- More information regarding rocprof can be found at: https://docs.olcf.ornl.gov/systems/frontier_user_guide.html#optimization-and-profiling
- More information on HPCToolkit can be found at: http://hpctoolkit.org

How to Offload using OpenMP ?

`target` directive

Syntax:

C/C++: **`#pragma omp target`** *`[clause[ [,] clause] ...`*

`structured-block`

Fortran: **`!$omp target`** *`[clause[ [,] clause] ... ]`*

`loosely/tightly-structured-block`

`!$omp end target`

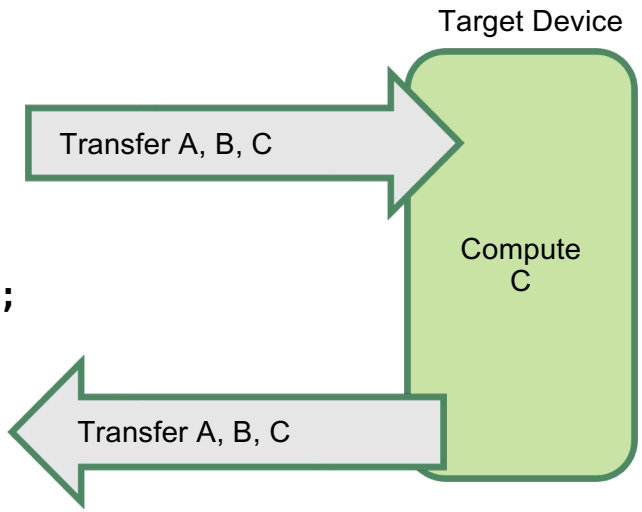
- A device data environment is created for the structured block
- The code region is mapped to the device and executed.

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
```

Kernel



The target construct is a task generating construct

Clauses on `target` directive

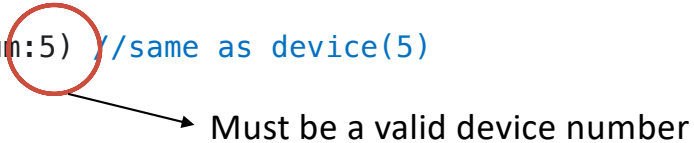
- Clauses allowed on the target directive:
  - device([device-modifier :] integer-expression)
  - if([target :] scalar-expression)
  - nowait
  - thread_limit(integer-expression)
  - private(list)
  - firstprivate(list)
  - in_reduction(reduction-identifier : list)
  - map([[map-type-modifier[,] [map-type-modifier[,]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

`device` clause on `target` directive

- Use
  - Specify which device should execute the kernel
    - takes device_num or ancestor modifiers

/*C code depicting use of device clause */

```
#pragma omp target device(device_num:5) //same as device(5)
  {
    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
```



`device` clause to target multiple non-host devices

```
/*C code to offload Matrix Addition Code to Multiple Devices*/
```

```
  …
int num_dev = omp_get_num_devices();
/*
Calculate start array index for each device and elements per device
*/

for (int dev = 0; dev < num_dev; ++dev)
{
 #pragma omp target map(tofrom: C[lb:len:1]) device(dev)
 {
   for (int i = lb; i < lb+len; ++i) {
     C[i] += A[i] + B[i] ;
    }
  } // end of omp target
}//end-for
```

`if` clause on `target` directive

- Use
  - Conditional execution on target device

/*C code demonstrating conditional offloading */

```
#pragma omp target if (N > 1024)
{
    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
```

`if` clause on `target` directive

When using if leads to non-portable behavior

```
#pragma omp begin declare target
void foo(double *d, double x)
{
  #pragma omp metadirective \
              when( construct={target}: distribute parallel for) \
              otherwise(parallel for)
  for(int i = 0; i < N; i++)
    // some work on array d and x;
}
#pragma omp end declare target
int main()
{
   double d[N];

   #pragma omp target teams // compiler A and compiler B call foo_target version
   foo(d, 10);

   #pragma omp target teams if(target:0) // compiler A calls foo_host but compiler B calls
foo_tgt_host
   foo(d, 15);

   foo(d, 20); // compiler A and compiler B calls foo_host
   …
}
```

`nowait` clause on `target` directive

- Use
  - target task may be deferred

/*C code with nowait on target */
/*Use-case: Free host thread*/

```
…
#pragma omp parallel
{
  #pragma omp masked
  #pragma omp target nowait
    {
     /*independent work unit*/
    } // end target
  #pragma omp for
  for (int i = 0; i < N; ++i) {
    C[i] = A[i] + B [i]
  }
  } // end parallel
```

`map` clause on `target` directive

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
  - decrements the reference count of a variab
- delete
  - reference count of a variable is set to 0
  - deletes the data from the device

If a map-type is not specified, the map-type defaults to tofrom
E.g. #pragma omp target map (A)

`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`
  - **hint** to the runtime to allocate memory close to the target device
- `present`
  - sets a requirement that the corresponding list item already exists in the device data environment
  - If the list item is not present it causes a runtime error

`map` clause: other modifiers

- `mapper`
  - Provides a mechanism to override implicit mapping and provide custom mapping
  - A `declare mapper` directive must be used to create a unique mapper
  - Use case: Nested structure elements or nested structures (deep copy)

```
/*C code to show use of mappers*/
```

```
  …
typedef struct S{
  size_t len;
  double *data;
} S_t;

#pragma omp declare mapper(X: S_t s)  map(s,
s.data[0:s.len])
…
int main(){
  S_t A;
  //allocate and initialize elements of A
#pragma omp target map(mapper(X))
{
      //work using array elements of A
} // end target
…
```

`map` clause: other modifiers (cont.)

- `iterator`
  - Defines a set of iterators, each of which is an iterator-identifier and an associated set of values
  - Expands based on the values assigned
  - Example:

```
int *A[N];
  for (int i = 0; i<N; i++){
          A[i] = (int *) malloc(sizeof(int));
          A[i][0] = i;
  }
  #pragma omp target map(iterator(it = 0:N), tofrom: A[it][:1]) map(to: test_lst)
```

Mapping Rules: Reference Count

- On entry to device environment:
  - If a corresponding storage block is not present in the device data environment, then:
    - A new storage block corresponding to original list item (on host) is created in the device data environment;
    - Reference count of this storage block is initialized to zero; and
  - The ref count is then incremented by 1
  - For every map clause list item in the storage block
    - If ref count of the storage block is 1 - new list item is created in the storage block
    - If ref count of the list item is 1 or `always map-type-modifier` is present, and `map-type` is `to` or `tofrom` the list item value on the target device is updated to the value on the host device

Atomic Operation

Mapping Rules: Reference Count (cont.)

- On exit from device environment:
  - If ref count is 1 or `always map-type-modifier` is specified, and `map-type` is `from` or `tofrom` original list item (on the host device) is updated
  - if ref count is finite and:
    - map-type is `delete` → ref count is set to 0
    - if map-type is not `delete` → 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

Allocating Memory on the Target 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.

`is_device_ptr` clause on `target` directive

- `is_device_ptr` clause
  - indicates that its list item has a valid device pointer for the device data environment
  - can be pointer returned by omp_target_alloc
  - or previously mapped ptr
- For C++ the list item must be:

  Check for NULL !
  - 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 must be of type C_PTR
- Support for device pointers created outside of OpenMP is implementation defined

Using `is_device_ptr`

```
/*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];
    }
}
…
…
omp_target_free(array_device, omp_get_default_device());
…
```

`has_device_addr` and `defaultmap` clause on `target` directive

- `has_device_addr` clause
  - indicates that the list items, like array sections, already have device addresses
  - list item must have a valid device address for the device data environment
  - if not, leads to unspecified behavior
- `defaultmap` clause
  - is used to change implicitly determined data-mapping and data-sharing attribute rules of variables referenced in the target region
  - Example: `defaultmap(tofrom: scalar)`

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.

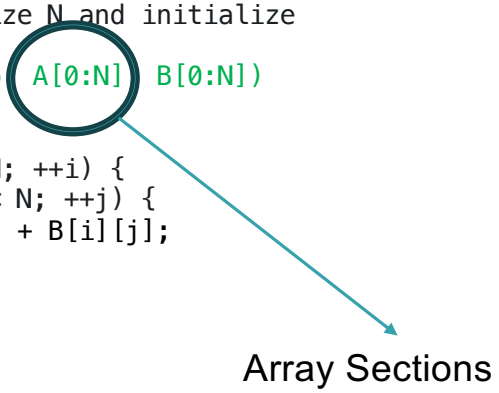
OpenMP Offload: Example using `omp target`

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

```
…
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*/

```
…
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
```



Target Device Data Persistence

/*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++ API	Fortran API	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

target data map directive usage

/*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) //kernel 1
  {
    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) //kernel 2
  {
    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 enter/exit data` directive usage

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

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

void 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)

}
```

```
int 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();
…
}


…
```

`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 list items consistent according to the specified data-motion-clauses.
  - `update to` makes makes the corresponding list items in the target device data environment consistent with their original list items
  - `udate from` makes makes the original list item in the host device data environment consistent with their corresponding list items on the target device data environment

target update directive usage

/*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
  {
    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 update from(D)                //Updates D; device → host
```

Declare Target directives

- Applied to procedures and/or variables to ensure that they can be executed or accessed on a device.
- A variable declared in the directive must:
  - have a mappable type
  - have static storage duration
- Two variations:
  - `declare target` directive (clauses: `enter, link, device_type, indirect`)
  - `begin declare target` directive (clauses: `device_type, indirect`)
    - Must be paired with end `declare target` directive

begin/end 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 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 Query 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.
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.

Device Memory Copy Routines

C/C++	Fortran	Description
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.
int omp_target_memcpy_async(…);	integer(c_int) function omp_target_memcpy_async(..	routine asynchronously performs a copy between host and device pointers.
int omp_target_memcpy_rect(..);	integer(c_int) function omp_target_memcpy_rect(dst,src,element_size, & .	copies a rectangular subvolume from a multi-dimensional array to another multi-dimensional array.
int omp_target_memcpy_rect_async(…);	integer(c_int) function omp_target_memcpy_rect_async(…	routine asynchronously copies a rectangular subvolume from a multi-dimensional array to another multi-dimensional array.

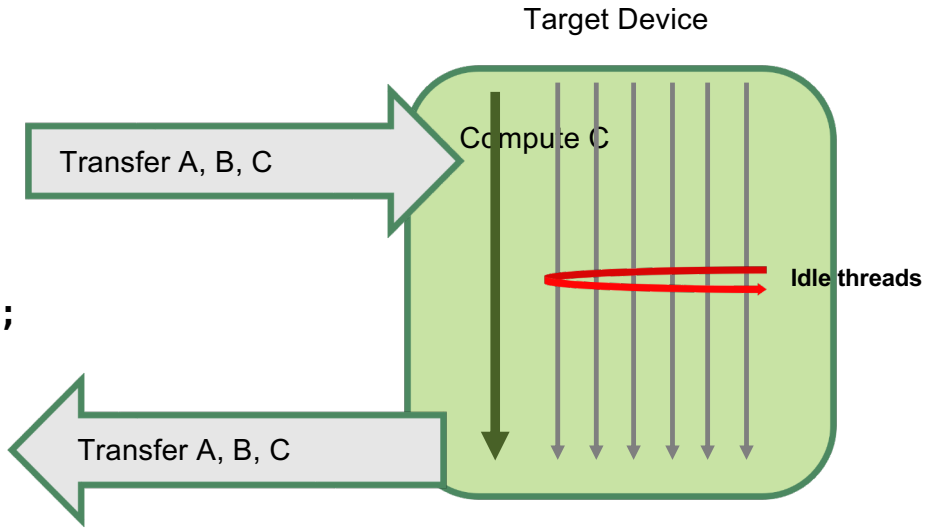
Other Device Memory Routines

C/C++	Fortran	Description
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.

Expressing Parallelism on non-host device

/*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
```



Expressing Parallelism Using combined constructs

- Combined construct
  - A construct that is a shortcut for specifying one construct immediately nested inside another construct. A combined construct is semantically identical to that of explicitly specifying the first construct containing one instance of the second construct and no other statements.
  - Example:

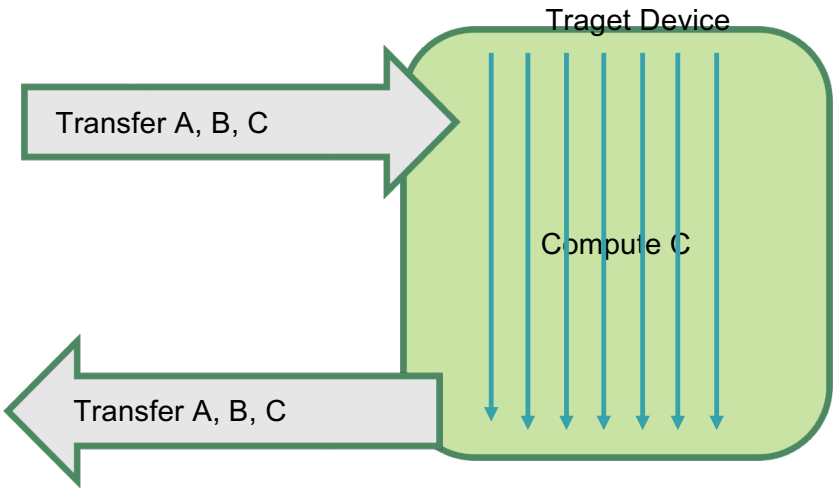
```
#pragma omp parallel
{
  #pragma omp for

}
// is equivalent to
#pragma omp parallel for
```

target + parallel construct

/*C code using target parallel*/

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



OpenMP `teams`

- When a `teams` construct is encountered, a league of teams is created.
- Each team is an initial team, and the initial thread "0" in each team executes the teams region.
  - Initial team numbers are consecutive whole numbers (zero to one less than the number of initial teams)
- The number of teams created is determined by the `num_teams` clause. Once the teams are created, the
  - these remain constant for the duration of the teams region.
- `The teams` region must be strictly nested within:
  - the implicit parallel region that surrounds the whole OpenMP program

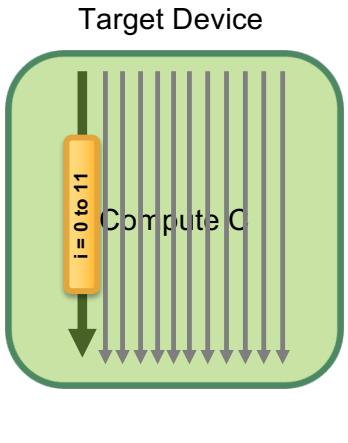
`distribute` construct

- It is a loop associated construct that binds to the set of initial threads executing an enclosing teams region
  - distribute construct must be strictly nested inside a teams region
- The iterations are distributed across the initial threads of all initial teams that execute the teams region to which the distribute region binds
- Clauses permitted on distribute construct are allocate, collapse, dist_schedule, firstprivate, lastprivate, order, and private

Expressing Parallelism: Increasing device utilization

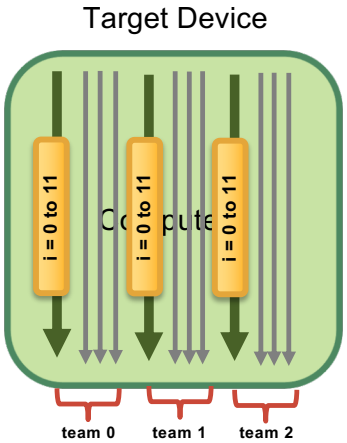
target

```
#pragma omp target
for (int i = 0; i < 12; ++i)
{
    C[i] = A[i] + B[i];
}
```



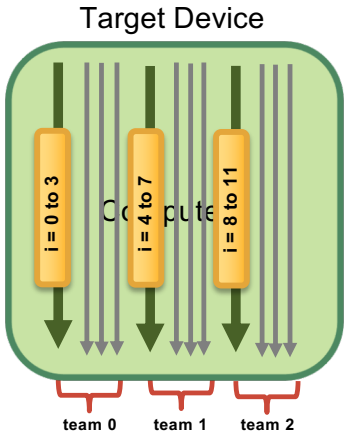
target teams

```
#pragma omp target teams
num_teams(3)
for (int i = 0; i < 12; ++i)
{
    C[i] = A[i] + B[i];
}
```



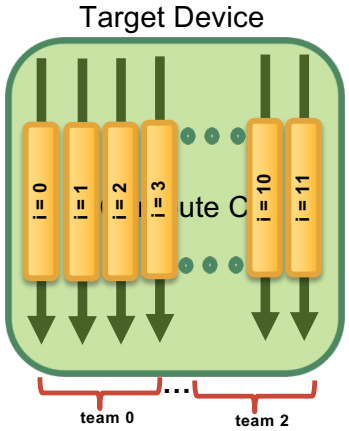
target teams distribute

```
#pragma omp target teams
distribute num_teams(3)
for (int i = 0; i < 12; ++i)
{
    C[i] = A[i] + B[i];
}
```



target teams distribute parallel

```
#pragma omp target teams
distribute parallel
for[simd] num_teams(3)
for (int i = 0; i < 12; ++i)
{
    C[i] = A[i] + B[i];
}
```



`loop` construct

- Properties:
  - logical iterations of the associated loops may execute concurrently
  - bind clause determines the binding region
    - orphaned loop needs explicit binding
  - can be nested inside another `loop` construct
  - all iterations are guaranteed to complete at the end of `loop`
    - except when bound to `teams` construct
      - iterations are guaranteed to complete before the end of the teams region.

Offloading using target teams + loop

/*C code with loop enclosed by teams region */

```
…
#pragma omp target teams
{
  #pragma omp loop //implicit bind(team)
  for (int i = 0; i < N; ++i) {
    C[i] = A[i] + B [i]
  }
} // end target teams
```

/*C code with orphaned loop */

```
…
void fun1(){
#pragma omp loop bind(teams)
for (int i = 0; i < N; ++i) {
    C[i] = A[i] + B [i]
}
#pragma omp target teams
{
  fun1();

} // end target teams
```

Other Device Execution Directives - `simd`

C/C++	Fortran	Description
#pragma omp target simd [clause[[,] clause] ...] new-line loop-nest	**!$omp target simd** [clause[[,] clause] ...] loop-nest [**!$omp end target simd**]	Semantics are identical to explicitly specifying a target directive immediately followed by SIMD directive.
#pragma omp target parallel for simd \ clause[[,] clause] ...] new-line loop-nest	**!$omp target parallel do simd** [clause[[,] clause] ...] loop-nest [**!$omp end target parallel do simd**]	Semantics are identical to explicitly specifying a target directive immediately followed by a parallel worksharing-loop SIMD directive.
#pragma omp target teams distribute simd \ [clause[[,] clause] ...] new-line loop-nest	**!$omp target teams distribute simd** [clause[[,] clause] ...] loop-nest [**!$omp end target teams distribute simd**]	Semantics are identical to explicitly specifying a target directive immediately followed by a teams distribute simd directive
**#pragma omp target teams distribute parallel for simd ** [clause[[,] clause] ...] new-line loop-nest	**!$omp target teams distribute parallel do simd** [clause[[,] clause] ...] loop-nest [**!$omp end target teams distribute parallel do simd**]	Semantics are identical to explicitly specifying a target directive immediately followed by a teams distribute parallel worksharing-loop SIMD directive.

Summary: Useful runtime routines for device

C/C++	Fortran	Where to call ?		Description
		Host	Target region	
int omp_get_num_procs(void);	integer function omp_get_num_procs()	✓	✓	returns the number of processors available to the device
void omp_set_default_device(int device_num);	subroutine omp_set_default_device(device_num) integer device_num	✓	✗	sets the value of the default-device-var ICV of the current task to device_num
int omp_get_default_device(void);	integer function omp_get_default_device()	✓	✗	returns the default target device
int omp_get_num_devices(void);	integer function omp_get_num_devices()	✓	✗	returns the number of non-host devices available for offloading code or data.
int omp_get_device_num(void);	integer function omp_get_device_num()	✓	✓	returns the device number of the device on which the calling thread is executing
int omp_is_initial_device(void);	logical function omp_is_initial_device()	✓	✓	returns true if the current task is executing on the host otherwise, it returns false.
int omp_get_initial_device(void);	integer function omp_get_initial_device()	✓	✗	return the device number of the host device

Summary: Useful runtime routines for `teams` region

C/C++	Fortran	Where to call ?		Description
		Host	Target region	
int omp_get_num_teams(void);	integer function omp_get_num_teams()	✓	✓	returns the number of initial teams in the current teams region.
int omp_get_team_num(void);	integer function omp_get_team_num()	✓	✓	returns the initial team number of the calling thread
void omp_set_num_teams(int num_teams);	subroutine omp_set_num_teams(num_teams) integer num_teams	✓	✓	the number of threads to be used for subsequent teams regions that do not specify a num_teams clause
int omp_get_max_teams(void);	integer function omp_get_max_teams()	✓	✓	returns an upper bound on the number of teams that could be created by a teams construct
void omp_set_teams_thread_limit(int thread_limit);	subroutine omp_set_teams_thread_limit(thread_limit) integer thread_limit	✓	✓	defines the maximum number of OpenMP threads per team

What's New in 6.0 ?

New features in 6.0

- Added OMP_AVAILABLE_DEVICES env variable
  - export OMP_AVAILABLE_DEVICES="kind(gpu)"
  - export OMP_AVAILABLE_DEVICES="kind(gpu)&&vendor(A),*"
- OMP_DEFAULT_DEVICE extended to support device selection by traits
  - export OMP_DEFAULT_DEVICE="kind(gpu)&&vendor(A),invalid"
- **local** clause for **declare_target** directive

New features in 6.0

- **groupprivate** directive for privatization for a contention group

C++ Example

```
void foo(int &sum, int tid)
{
   static int x[100];
   #pragma omp groupprivate(x)
   //initialize and use x
   }
}
#pragma omp declare_target enter(foo)
```

```
int main()
{
   …
   #pragma omp target teams num_teams(4)
thread_limit(100)
   #pragma omp parallel
   foo(sums[omp_get_team_num()],
omp_get_thread_num());
```

New features in 6.0

- **target_data** as a composite directive

Code

```
#pragma omp target_data map(a)
{
  #pragma omp target map(a) nowait
    do_stuff_with_a(a);
}
```

Equivalent code according to 6.0

```
#pragma omp target_enter_data map(a)
depend(out: t_)    // T1
#pragma omp task transparent mergable
depend(inout:t_) default(shared)  //T2
#pragma omp taskgroup
{
  #pragma omp target map(a) nowait  // T2_1
    do_stuff_with_a(a);
}
  #pragma omp target_exit_data map(a)
depend(in: t_)       // T3
```

New features in 6.0

- Single environment variable for host and non-host devices
- Modification to the **declare_target** directive list items must now be placed at the same scope as their declaration.
- **self_maps** requirement clause added
- New device routines available on host device - omp_get_device_num_teams, omp_set_device_num_teams, omp_get_device_teams_thread_limit, and omp_set_device_teams_thread_limit routine
- Full support for C23, C++23, and Fortran 2023
- Support for free agent and structured threads via OMP_THREADS_RESERVE

References

- Examples were adapted from:
  - OpenMP Examples Document - https://www.openmp.org/wp-content/uploads/openmp-examples-6.0.pdf
  - OpenMP Validation and Verification Suite - https://github.com/OpenMP-Validation-and-Verification/OpenMP_VV
- OpenMP Specifications
  - https://www.openmp.org/wp-content/uploads/OpenMP-API-Specification-5-2.pdf
  - https://www.openmp.org/wp-content/uploads/OpenMP-API-Specification-6-0.pdf

Thank You