# GPU Programming with OpenMP
# Part 2: Data mapping




Hans Henrik Brandenborg Sørensen

DTU Computing Center

DTU Compute

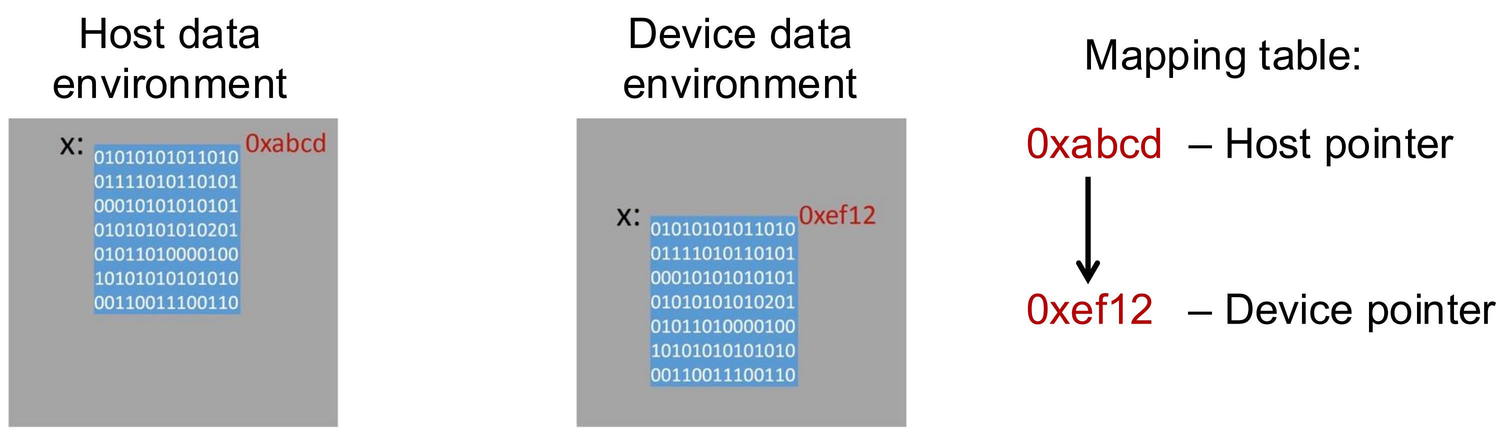
<hhbs@dtu.dk>

# Overview

- OpenMP data mapping
  - `map` clause
  - `target data`
  - `target enter / exit data`
  - `target update`
  - `declare target`
- OpenMP array sections
- OpenMP runtime library
  - Device versions of malloc, free, memcpy
  - Matrix allocation on the device

# OpenMP data environment

- OpenMP maintains a device data environment and a mapping table that records what memory pointers have been mapped from the host




  - The table also maintains the translation between host memory pointers and device memory pointers
  - Presence checks determine if data is already part of the device data environment before a transfer is made

# OpenMP offload basics

- `target` clauses related to data environment
  - `map`
  - `is_device_ptr`
  - `has_device_addr`
  - `defaultmap`
  - ~~`allocate`~~
  - ~~`uses_allocators*`~~

Not currently supported in `nvc++`

# OpenMP offload syntax

- Syntax C/C++:

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

- The `map` clause specifies how a list item is mapped from the host data environment to the corresponding list item in the device data environment

# OpenMP offload syntax

- `map-type` can be
  - `to`
    - Allocates memory and moves data to the device
  - `from`
    - Allocates memory and moves data from the device
  - `tofrom`
    - Allocates memory and moves data to and from the device
  - `alloc`
    - Allocates memory on the device
  - `release`
    - The reference count is decreased by one
  - `delete`
    - Deletes data from the device (the reference count is set to 0)

# OpenMP offload syntax

- `map-type-modifier` can be
  - `always`
    - Always copies the data to and from the device
  - ~~`close`~~
    - A hint to the runtime to allocate memory close to the target device
  - ~~`mapper(mapper-identifier)`~~
    - Use a user-defined mapper
  - ~~`present`~~
    - This clause is ordered before other map clauses w/o present
  - ~~`iterator(iterator-definition)`~~
    - Reference iterators defined by an iterators-definition

Not currently supported in `nvc++`

# OpenMP offload syntax

- Example

```
double a[N], b[N], c[N];
/*
  initialize arrays
*/
#pragma omp target teams loop map(to: a, b) map(from: c)
for (int i = 0; i < N; i++)
    c[i] = a[i] + b[i];
```

  - Arrays a and b are transferred **to** the device before the offload
  - Array c is **allocated** on the device before the offload
  - Array c is transferred **from** the device after the offload

# OpenMP offload syntax

- Example – dynamic allocation

```
double *a, *b, *c;
/*
  allocate and initialize arrays of size N
*/
#pragma omp target teams loop \
    map(to: a[0:N], b[0:N]) map(from: c[0:N])
for (int i = 0; i < N; i++)
    c[i] = a[i] + b[i];

/* free arrays */
```

OpenMP array sections

  - Arrays a and b are transferred **to** the device before the offload
  - Array c is **allocated** on the device before the offload
  - Array c is transferred **from** the device after the offload

# OpenMP array sections

- An array section designates a subset of the elements in an array given by:

  `[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
  - If `lower-bound` is left out it defaults to 0
  - `length` must evaluate to a non-negative integer and must be explicitly specified when the size of the array dimension is not known
  - `stride` must evaluate to a positive integer (default 1)

# OpenMP array sections

- Example – dynamic allocation

```
double *a, *b, *c;
/*
  allocate and initialize arrays of size N
*/
#pragma omp target teams loop \
    map(to: a[0:N/2], b[0:N/2]) map(from: c[0:N/2])
for (int i = 0; i < N/2; i++)
    c[i] = a[i] + b[i];

#pragma omp target teams loop \
    map(to: a[N/2:N/2], b[N/2:N/2]) map(from: c[N/2:N/2])
for (int i = N/2; i < N; i++)
    c[i] = a[i] + b[i];
```

# OpenMP offload syntax

- Another example

```
double a[N], b[N], c[N], d[N];
/*
  initialize arrays
*/
#pragma omp target teams loop map(to: a, b) map(from: c)
for (int i = 0; i < N; i++)
    c[i] = a[i] + b[i];

#pragma omp target teams loop map(to: a, b) map(from: d)
for (int i = 0; i < N; i++)
    d[i] = a[i] - b[i];
```

  - Two offload regions that both use a and b (transferred 2 times)
  - This works, but it is not the best we can do!

# OpenMP offload syntax

- Syntax C/C++:

```
#pragma omp target data [clause]
{
      ...
}
```

- Clause can be
  - `if([ target data :] scalar_expr)`
  - `device(int_expr)`
  - `map(...)`
  - `use_device_ptr(ptr-list)`
  - `use_device_addr(list)`
- Creates persistent data environment within `{ }`

# OpenMP offload syntax

- So the better solution is

```
double a[N], b[N], c[N], d[N];
/*
  initialize arrays
*/
#pragma omp target data map(to: a, b) map(from: c, d)
{
    #pragma omp target teams loop map(to: a, b) map(from: c)
    for (int i = 0; i < N; i++)
        c[i] = a[i] + b[i];

    #pragma omp target teams loop map(to: a, b) map(from: d)
    for (int i = 0; i < N; i++)
        d[i] = a[i] - b[i];
}
```

Presence check: a and b already available!

- Arrays a and b are transferred once and used in both offloads regions

# OpenMP measuring runtimes

- Measuring the transfer time

```
double a[N], b[N], c[N];
/*
  initialize arrays and warmup device
*/
double t = omp_get_wtime();
#pragma omp target data map(to: a, b) map(from: c)
{
    double t = omp_get_wtime();
    #pragma omp target teams loop map(to: a, b) map(from: c)
    for (int i = 0; i < N; i++)
        c[i] = a[i] + b[i];
    printf("Runtime w/o transfer:  %f\n", omp_get_wtime() - t);
}
printf("Runtime with transfer: %f\n", omp_get_wtime() - t);
```

```
$ ./vecadd
Runtime w/o transfer:  0.000565
Runtime with transfer: 0.002533
```

- For reliable runtimes run offload region several times and average

# OpenMP offload syntax

- Syntax C/C++:

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

- Clause can be
  - `if([ target data :] scalar_expr)`
  - `device(int_expr)`
  - `map(...)`
  - `depend([depend-modifier,] depen-type : list)`
  - `nowait`
- Standalone directives that specifies mapping to the data environment of the default device

# OpenMP offload syntax

- Target enter / exit example

main.cpp

```
int main(int argc, char *argv[]) {
    double a[N], b[N], c[N];

    initialize(a, b, c);

    #pragma omp target teams loop \
        map(to: a, b) map(from: c)
    for (int i = 0; i < N; i++)
        c[i] = a[i] + b[i];

    finalize(a, b, c);

    return(0);
}
```

functions.cpp

```
void initialize(double *a, double *b, double *c) {
    for (int i = 0; i < N; i++)
        a[i] = b[i] = i;
    #pragma omp target enter data \
       map(to: a[:N], b[:N]) map(alloc: c[:N])
}

void finalize(double *a, double *b, double *c) {
    #pragma omp target exit data \
       map(release: a[:N], b[:N]) map(from: c[:N])
    for (int i = 0; i < N; i++)
        printf("%f\n", c[i]);
}
```

- Remember to release a and b in order to clean up data environment

# OpenMP measuring runtime

- H2D and D2H transfers

```
$ ./vecadd
H2D transfer time:  0.001761
D2H transfer time:  0.001139
```

main.cpp

```
int main(int argc, char *argv[]) {
    double a[N], b[N], c[N];
    /*
      warm up device
    */
    initialize(a, b, c);

    #pragma omp target teams loop \
        map(to: a, b) map(from: c)
    for (int i = 0; i < N; i++)
        c[i] = a[i] + b[i];

    finalize(a, b, c);

    return(0);
}
```

functions.cpp

```
void initialize(double *a, double *b, double *c) {
    for (int i = 0; i<N; i++)
        a[i] = b[i] = i;
    double t = omp_get_wtime();
    #pragma omp target enter data \
       map(to: a[:N], b[:N]) map(alloc: c[:N])
    printf("H2D transfer time:  %f\n",
           omp_get_wtime() - t);
}

void finalize(double *a, double *b, double *c) {
    double t = omp_get_wtime();
    #pragma omp target exit data \
       map(release: a[:N], b[:N]) map(from: c[:N])
    printf("D2H transfer time:  %f\n",
           omp_get_wtime() - t);}
}
```

- Remember to warm up the device or transfer times will be +0.2 secs

# OpenMP offload syntax

- Syntax C/C++:

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

- Clause can be
  - `if([ target data :] scalar_expr)`
  - `device(int_expr)`
  - `depend([depend-modifier,] depen-type : list)`
  - `nowait`
  - `to([motion-modifier[,] ...]: ] list)`
  - `from([motion-modifier[,] ...]: ] list)`
- Standalone directive that makes the device data environment consistent with their original list items, according to the specified motion clauses

# OpenMP offload syntax

- Target update example

```
#pragma omp target data map(to: a, b) map(from: c, d)
{
    #pragma omp target teams loop map(to: a, b) map(from: c)
    for (int i = 0; i < N; i++)
        c[i] = a[i] + b[i];

    modify_b_on_host(b);
    #pragma omp target update to(b)

    #pragma omp target teams loop map(to: a, b) map(from: d)
    for (int i = 0; i < N; i++)
        d[i] = a[i] - b[i];
}
```

- Array b is modified in the host data environment and needs to be updated in the device data environment before the final offload

# OpenMP offload syntax

- Syntax C/C++:

```
#pragma omp declare target
declarations
#pragma omp end declare target
```

```
#pragma omp declare target [(list) | clause]
```

- Clause can be
  - `to(list)`
  - ~~`link(list)`~~
  - `device_type(host | nohost | any)`
- This directive specifies that global variables and functions (C/C+) are mapped to a device for all device executions (or for a specific one via link)

# OpenMP offload syntax

- Using declare target

```
#pragma omp declare target
double a[N], b[N], c[N];
#pragma omp end declare target

void vecadd() {
    for (int i = 0; i < N; i++)
        c[i] = a[i] + b[i];
}
#pragma omp declare target (vecadd)

int main(int argc, char *argv[]) {
    /* initialize arrays on host */
    vecadd(); // Call host version
    #pragma omp target update to(a, b)
    #pragma omp target
    vecadd(); // Call device version
    #pragma omp target update from(c)
}
```

no mapping – arrays are globally available

# Implicit data mapping

- Why did this run?

```
#define N 16
int main(int argc, char *argv[]) {
    double a[N], b[N], c[N];
    for (int i = 0; i < N; i++)
        a[i] = b[i] = i * 1.0;

    #pragma omp target teams \
        distribute parallel for
     for (int i = 0; i < N; i++)
        c[i] = a[i] + b[i];

    for (int i = 0; i < N; i++)
        printf("%f\n", c[i]);
}
```

arrays are implicitly mapped to the device

arrays are implicitly mapped back to the host

# Implicit data mapping

- If a variable is a scalar then it is implicitly `firstprivate` (i.e., not mapped)
- If a variable is not a scalar (i.e. an array or struct) then implicitly a `map(tofrom:...)` is added
- If a variable is a pointer its actual value has no meaning in the device memory and it “is treated as if it is the base pointer of a zero-length array section that had appeared in a `map` clause”
- However, if the `defaultmap` clause is present in the construct, it takes precedence

# Summary of data mapping

- Map clause on a target construct
  - Maps variables for a single target region
  - Enclosed region executes on device and maps data
- Target data
  - Map variables across multiple target regions
  - Region does not execute on device, only maps data
- Declare target
  - Mapping variables and functions for the whole execution of the program (“globally mapped”)
- Target enter/exit/update data
  - Map variables in stand-alone clauses

# OpenMP runtime library

# OpenMP offload runtime library

- Managing memory yourself

| *Name* | *Functionality* |
| --- | --- |
| `void* omp_target_alloc(size_t, int dev_num)` | allocate memory on device |
| `void omp_target_free(void*, int dev_num)` | free memory on device |
| `int omp_target_memcpy(...,int dev_num)` | memcpy to and from device |
| `int omp_target_memcpy_rect(...,int dev_num)` | memcpy to and from device of a rectangular subvolume |
| `omp_target_associate_ptr(...)`<br>`omp_target_disassociate_ptr(...)`<br>`omp_target_is_present(...)` | combining device ptr with host ptr to be used in map clause<br>(we use is_device_ptr clause) |

# OpenMP offload syntax

- Syntax C/C++:

```
void* omp_target_alloc(size_t size,
                       int dev_num);
```

- This routine allocates memory of `size` bytes in the device data environment of device `dev_num` and returns a device pointer to that memory
- Syntax C/C++:

```
void omp_target_free(void *dev_ptr,
                     int dev_num);
```

- This routine frees the memory at `dev_ptr` in the device data environment of device `dev_num`

# OpenMP offload syntax

- Syntax C/C++:

```
int omp_target_memcpy(void *dst,
                      void *src,
                      size_t length,
                      size_t dst_offset,
                      size_t src_offset,
                      int dst_dev_num,
                      int src_dev_num);
```

- This routine copies `length` bytes of memory at offset `src_offset` from `src` in the device data environment of device `src_dev_num` to `dst` starting at `offset dst_offset` in the device data environment of device `dst_dev_num`

# OpenMP offload syntax

- The `is_device_ptr` clause indicates that its list items are device pointers
  - Inside the target construct, each list item is privatized and the new list item is initialized to the device address to which the original list item refers
- Support for device pointers created outside of OpenMP, specifically outside of any OpenMP mechanism that returns a device pointer, is implementation defined

CUDA device pointers are fully supported in `nvc++`

# OpenMP offload syntax

- Example – dynamic allocation

```
double *a, *b, *c, *a_d, *b_d, *c_d; // Notation _d is convention for device pointers.
/* allocate and initialize arrays on host */

int dev_num = omp_get_default_device();
a_d = (double*)omp_target_alloc(N * sizeof(double), dev_num);
b_d = (double*)omp_target_alloc(N * sizeof(double), dev_num);
c_d = (double*)omp_target_alloc(N * sizeof(double), dev_num);

omp_target_memcpy(a_d, a, N * sizeof(double), 0, 0, dev_num, omp_get_initial_device());
omp_target_memcpy(b_d, b, N * sizeof(double), 0, 0, dev_num, omp_get_initial_device());

#pragma omp target teams loop is_device_ptr(a_d, b_d, c_d)
for (int i = 0; i < N; i++)
    c_d[i] = a_d[i] + b_d[i];

omp_target_memcpy(c, c_d, N * sizeof(double), 0, 0, omp_get_initial_device(), dev_num);
...
omp_target_free(a_d, dev_num);
omp_target_free(b_d, dev_num);
omp_target_free(c_d, dev_num);
```

clause `is_device_ptr` required here

# OpenMP offload runtime library

- Example matrix allocation – host version

```c
/* Routine for allocating two-dimensional array */
double **malloc_2d(int m, int n) {
    if (m <= 0 || n <= 0)
        return NULL;

    double **A = (double**)malloc(m * sizeof(double *));
    if (A == NULL)
        return NULL;

    A[0] = (double*)malloc(m * n * sizeof(double));
    if (A[0] == NULL) {
        free(A);
        return NULL;
    }

    for (int i = 0; i < m; i++)
      A[i] = A[0] + i * n;

    return A;
}
```

```c
void free_2d(double **A) {
    free(A[0]);
    free(A);
}
```

# OpenMP offload runtime library

- Example matrix allocation – device version

```
/* Routine for allocating two-dimensional array on the device */
double **malloc_2d_dev(int m, int n, double **data) {
    if (m <= 0 || n <= 0)
        return NULL;

    double **A = (double**)omp_target_alloc(m*sizeof(double *), omp_get_default_device());
    if (A == NULL)
        return NULL;

    double *a = (double*)omp_target_alloc(m*n * sizeof(double), omp_get_default_device());
    if (a == NULL) {
        omp_target_free(A, omp_get_default_device());
        return NULL;
    }
    #pragma omp target is_device_ptr(A, a)
    for (int i = 0; i < m; i++)
      A[i] = a + i * n;

    *data = a; return A;
}
```

```
void free_2d_dev(double **A,
                 double *data) {
    omp_target_free(data,
        omp_get_default_device());
    omp_target_free(A,
        omp_get_default_device());
}
```

# OpenMP offload runtime library

- Example matrix allocation

```
double **A = malloc_2d(N, N);                              // Allocate A on host

double *data;
double **A_d = malloc_2d_dev(N, N, &data);                 // Allocate A_d on device

/* initialize A on host */

omp_target_memcpy(data, A[0], N * N * sizeof(double),    // Copy data from A to A_d
                  0, 0, omp_get_default_device(), omp_get_initial_device());

double sum = 0.0;
#pragma omp target teams distribute parallel for reduction(+:sum) is_device_ptr(A_d)
    for (int i = 0; i < N; i++)
        for (int j = 0; j < N; j++)
            sum += A_d[i][j] * A_d[i][j];
printf("Frobenius norm: %f\n", sqrt(sum));

free_2d(A);                                                // Free A on host
free_2d_dev(A_d, data);                                    // Free A_d on device
```

# Exercises

- Finish up the first two exercises
  - ex1_nvallocinfo
  - ex2_helloworld
- Begin the third exercise
  - ex3_mandelbrot

# End of lecture