

Joint INTEL / EuroCC/Castiell oneAPI Workshop

Parallelizing heterogeneous applications with Intel[®] OpenMP and OpenMP* offloading

Alina Shadrina

alina.shadrina@intel.com



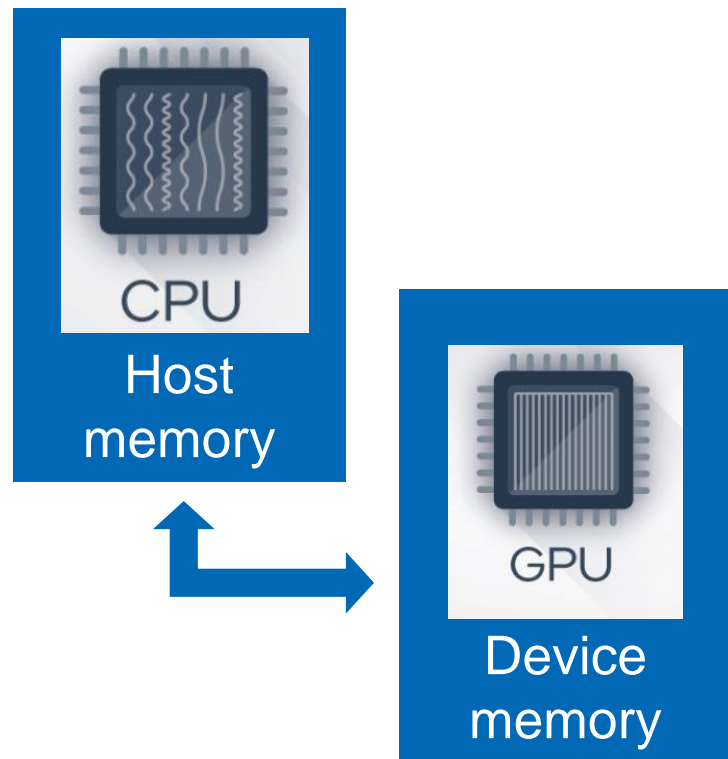
*Other names and brands may be claimed as the property of others.

Agenda

- OpenMP* Offload Compiler Support
- OpenMP* Target Construct
- Managing Device Data

OpenMP* Offload Compiler Support

Device Model



- Host-centric model
- Host and Device have separate memory spaces
- Device data environment
- We need to move data from host to device to access data inside target region
- We need constructs to offload code to device

OpenMP* Offload Compiler Support

- Intel® C++ Compiler

```
icx -qopenmp -fopenmp-targets=spir64 <source>.c
```

```
icpx -qopenmp -fopenmp-targets=spir64 <source>.cpp
```

- Intel® Fortran Compiler

```
ifx -qopenmp -fopenmp-targets=spir64 <source>.f90
```

- Hardware Supported: Intel® Gen9

- Add ``use omp_lib`` to source code

- [OpenMP directives supported in the icx and ifx compilers for GPU and CPU](#)

OpenMP Support:

- [q or Q]openmp
 - -fopenmp (Linux*)
or /Qopenmp (Windows*) (ifx only)
- These options are equivalent in ifx to -qopenmp on Linux* and /Qopenmp on Windows*
- -fopenmp (ifx; deprecated)

OpenMP* Offload Compiler Support

- Ahead-of-Time compilation supported

```
ifx -fiopenmp -fopenmp-targets=spir64_gen -Xopenmp-target-backend "-device *" <source>.f90
```

- -Xopenmp-target-frontend=T"options"
- -Xopenmp-target-backend=T"options"
- -Xopenmp-target-linker=T"options"

Environment variables

- `OMP_TARGET_OFFLOAD` : Control offload on device or host
 - Set `MANDATORY` to start offloading
 - Set `DISABLED` to 'emulate' offloading on CPU (implementation defined!)
- `LIBOMPTARGET_PLUGIN` : Choose runtime backend
 - Choose `OpenCL™` or `Level0`
- `LIBOMPTARGET_DEBUG` : Display debug information
 - Gives you a long and detailed log!
 - Use `1` as value
- `LIBOMPTARGET_PROFILE`: Add profiling info
 - Try `T,usec`
- `LIBOMPTARGET_INFO`: data-mappings and kernel execution
 - 32-bit field to enable or disable different types of information
 - `-1` – enable every bit set

OpenMP Offload Constructs

■ Device Code

- **omp target** [*clause*[[*,*]*clause*]...] *structured-block*
- **omp declare target** [*function-definitions-or-declarations*]
- **omp declare target** [*variable-definitions-or-declarations*]

■ Worksharing

- **omp teams** [*clause*[[*,*]*clause*]...] *structured-block*
- **omp distribute** [*clause*[[*,*]*clause*]...] *for-loops*

■ Memory operations

- **map** ([[*map-type-modifier*[[*,*]*map-type*:]*list*]) *map-type* := **alloc** | **tofrom** | **to** | **from** | **release** | **delete** *map-type-modifier* := **always**
- **omp target data** *clause*[[[*,*]*clause*]...] *structured-block*
- **omp target enter data** *clause*[[[*,*]*clause*]...]
- **omp target exit data** *clause*[[[*,*]*clause*]...]
- **omp target update** *clause*[[[*,*]*clause*]...]

OpenMP Offload Language

C++	Fortran
#pragma omp target <i>[clause[,]clause...]</i> <i>structured-block</i>	!\$omp target <i>[clause[,]clause...]</i> <i>structured-block</i> !\$omp end target
#pragma omp target data <i>[clause[,]clause...]</i> <i>structured-block</i>	!\$omp target <i>[clause[,]clause...]</i> <i>structured-block</i> !\$omp end target data
#pragma omp teams <i>[clause[,]clause...]</i> <i>structured-block</i>	!\$omp teams <i>[clause[,]clause...]</i> <i>structured-block</i>
#pragma omp distribute <i>[clause[,]clause...]</i> <i>structured-block</i>	!\$omp distribute <i>[clause[,]clause...]</i> <i>structured-block</i>

OpenMP* Target Construct

Target construct

```
integer :: a(100), b(100), c(100)
do k=1,100
    a(k) = 1
    b(k) = 1
end do
```

Host code

```
!$omp target
    do k=1,100
        c(k) = a(k) + b(k)
    end do
!$omp end target
```

Device code

```
do k=1,100
    write (*,*) c(k)
end do
```

Host code

target [clause]

- Offloads a code region to a target device
- Sequential and synchronous by default

clause : device, private, firstprivate, map, allocate

Sync: nowait, depend

Target device construct

```
integer :: a(100), b(100), c(100)
do k=1,100
    a(k) = 1
    b(k) = 1
end do
```

Host code

```
!$omp target device (0)
    do k=1,100
        c(k) = a(k) + b(k)
    end do
!$omp end target
```

Device code

```
do k=1,100
    write (*,*) c(k)
end do
```

Host code

target device

- Specify which device to offload to in a multi-device environment
- Device number an integer
 - Assignment is implementation-specific
 - Usually start at 0 and sequentially increments
- Works with **target**, **target data**, **target enter \ exit data**, **target update** directives

OpenMP* Device Parallelism

```
integer :: a(100), b(100), c(100)
do k=1,100
    a(k) = 1
    b(k) = 1
end do
```

Host code

```
!$omp target device (0)
!$omp parallel do
do k=1,100
    c(k) = a(k) + b(k)
end do
!$omp end parallel do
!$omp end target
```

Device code

```
do k=1,100
    write (*,*) c(k)
end do
```

Host code

target [clause]

- Offloads a code region to a target device
- Sequential and synchronous by default

Why NOT parallel for?

- CPU parallelism differs from GPU – shared memory systems
- `omp parallel` for threads will use only 1 Streaming Multiprocessor (SM) to synchronize
- Need a different level of parallelism to step over multiple SM

GPU device architecture

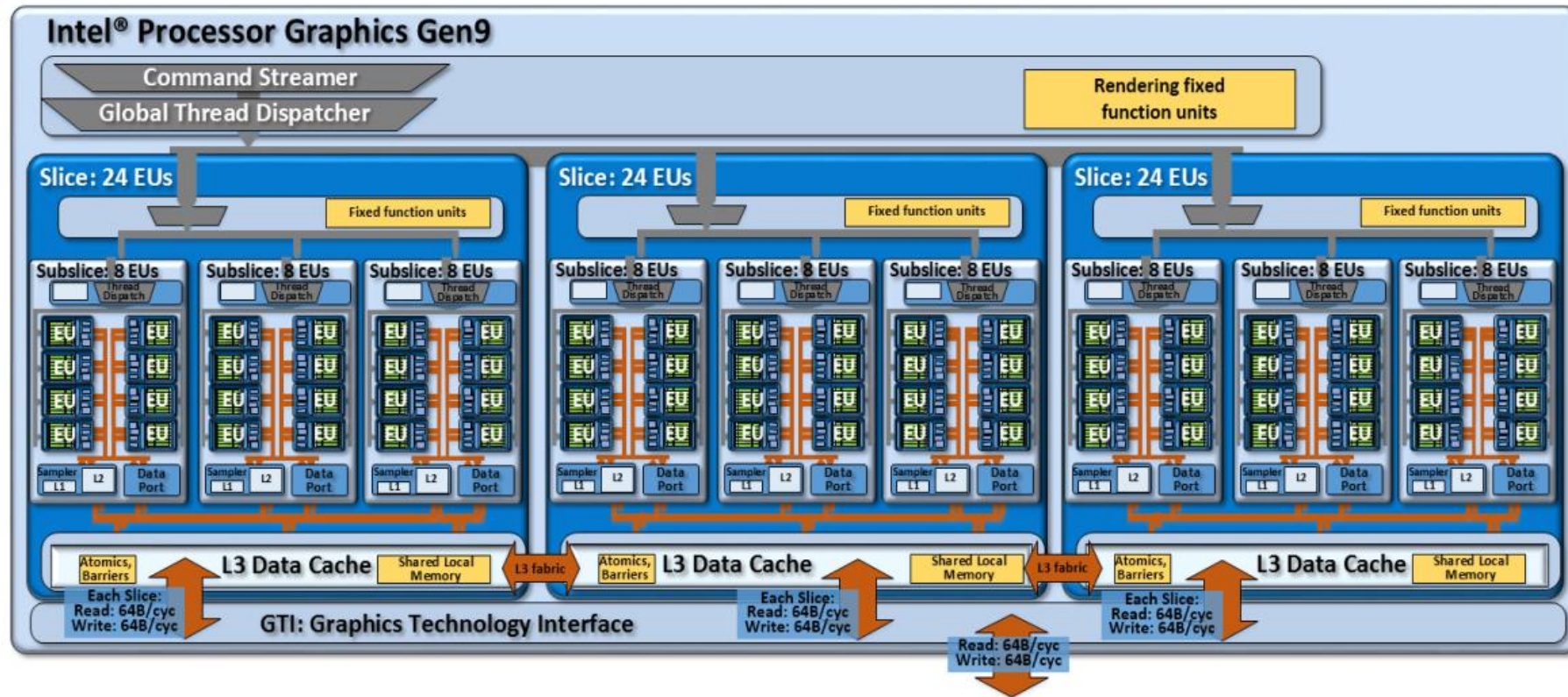


Figure 8: Another potential product design that instantiates the compute architecture of Intel® processor graphics gen9. This design is composed of three slices, of three subslices each for a total of 72 EUs.

OpenMP* Device Parallelism

```
integer :: a(100), b(100), c(100)
do k=1,100
    a(k) = 1
    b(k) = 1
end do
```

Host code

```
!$omp target teams
!$omp parallel do
    do k=1,100
        c(k) = a(k) + b(k)
    end do
!$omp end parallel do
!$omp end target
```

Device code

```
do k=1,100
    write (*,*) c(k)
end do
```

Host code

target [clause]

Offloads a code region to a target device

Sequential by default

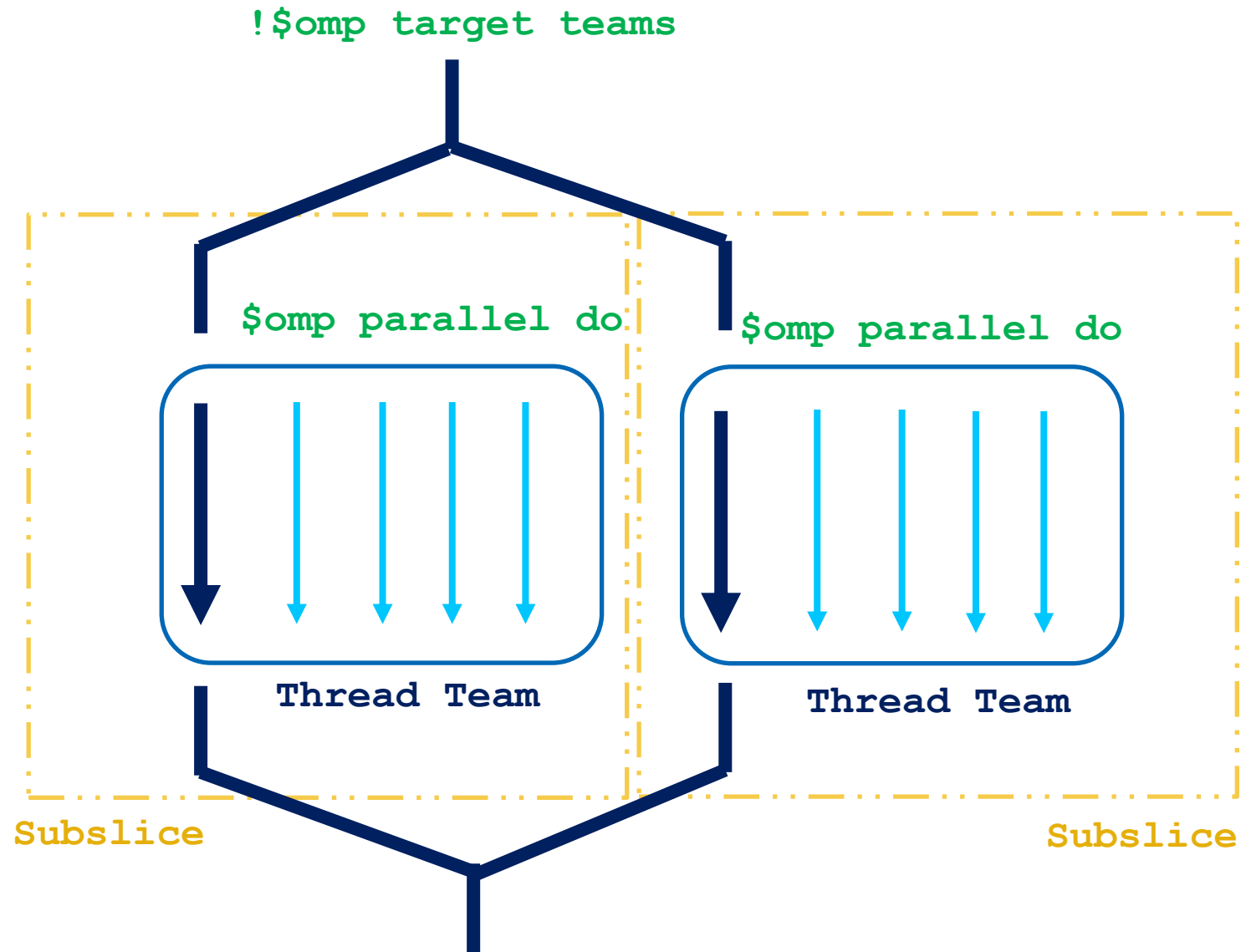
target teams

creates a *league* of teams where the primary thread of each team executes the teams region.

number of teams = number of work groups
(**clinfo**)

Teams Construct

OpenMP	GPU Hardware
SIMD	SIMD Lane (Channel)
Thread	SIMD Thread mapped to an EU
Team	Group of threads mapped to a Subslice
League	Multiple Teams mapped to a GPU



OpenMP* Worksharing

```
integer :: a(100), b(100), c(100)
do k=1,100
    a(k) = 1
    b(k) = 1
end do
```

Host code

```
!$omp target teams distribute parallel do
    do k=1,100
        c(k) = a(k) + b(k)
    end do
!$omp end target teams distribute parallel do
```

Device code

```
do k=1,100
    write (*,*) c(k)
end do
```

Host code

target teams distribute

shortcut for specifying a target construct containing a teams distribute construct and no other statements.

target teams distribute parallel do

parallel worksharing-loop construct is a shortcut for specifying a target construct containing a teams distribute parallel worksharing-loop construct and no other statements

Teams Distribute Construct

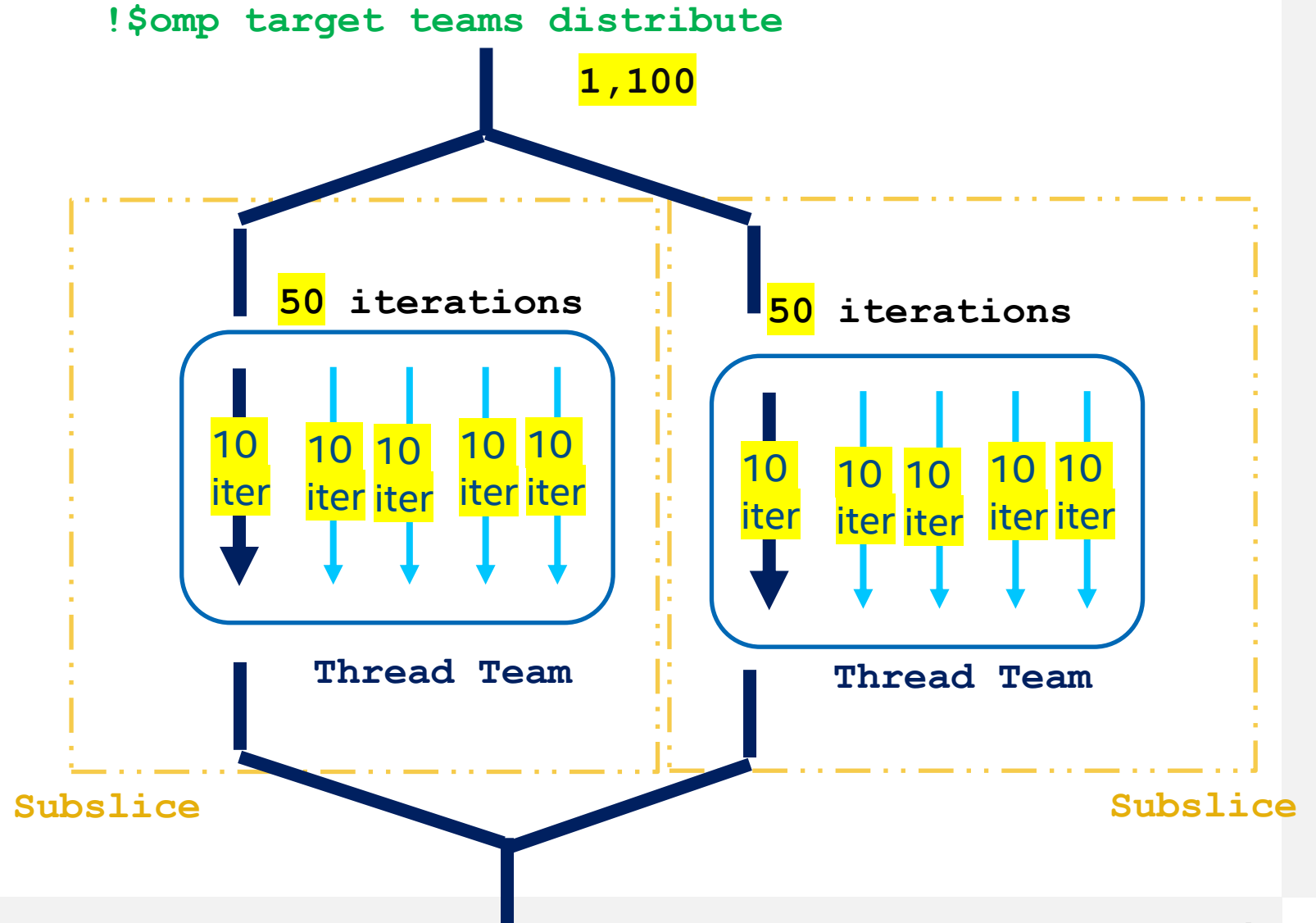
```
!$omp target teams distribute parallel do
```

```
do k=1,100
```

```
    c(k) = a(k) + b(k)
```

```
end do
```

```
!$omp end target teams distribute parallel do
```



Calling functions inside Target region

```
subroutine f(N)
  integer :: N
  !$omp declare target
  ...
  !$omp end declare target
end subroutine
```

Host code

```
!$omp target teams
      call f(N)
!$omp end target
```

Device
code

```
do k=1,100
      write (*,*) c(k)
end do
```

Host code

declare target

compiles a version of the function/subroutine for the target device

Function compiled for both host execution and target execution by default

**device_type(host |
nohost | any)**

OpenMP Offload Constructs

■ Device Code

- ★ • **omp target** *[clause[[[,]clause]...] structured-block*
- ★ • **omp declare target** *[function-definitions-or-declarations]*
- ★ • **omp declare target** *[variable-definitions-or-declarations]*

■ Worksharing

- ★ • **omp teams** *[clause[[[,]clause]...] structured-block*
- ★ • **omp distribute**
[clause[[[,]clause]...] for-loops

■ Memory operations

- **map** *([[map-type-modifier[,]map-type:] list) map-type := **alloc** | **tofrom** | **to** | **from** | **release** | **delete** map-type-modifier := **always***
- **omp target data** *clause[[[,] clause]...] structured-block*
- **omp target enter data**
clause[[[,]clause]...]
- **omp target exit data**
clause[[[,]clause]...]
- **omp target update** *clause[[[,]clause]...]*

Managing Device Data

Basic Principles

Host and devices have separate memory spaces

- Special operation (**mapping**) is needed to access data inside target region
- Data environment is lexically scoped
 - Data environment is destroyed at closing curly brace
 - Allocated buffers/data are automatically released

Target map construct

```
integer :: a(100), b(100), c(100)
do k=1,100
    a(k) = 1
    b(k) = 1
end do
```

Host code

```
!$omp target teams distribute parallel do
    map(to:a) map(to:b) map(tofrom:c)
        do k=1,100
            c(k) = a(k) + b(k)
        end do
    !$omp end parallel do
!$omp end target
```

Device code

```
do k=1,100
    write (*,*) c(k)
end do
```

Host code

target map (map_type)

Map variables to a device data environment and execute the construct on that device.

map_type : to, from, tofrom, alloc, release, delete

modifier: always, close, <mapper identifier>

Dynamically Allocated Data

```
integer :: a(100), b(100), c(100)
do k=1,100
    a(k) = 1
    b(k) = 1
end do
```

Host code

```
!$omp target teams distribute parallel do
    map(to:a[0:N]) map(to:b [0:N]) map(tofrom:c [0:N])
    do k=1,100
        c(k) = a(k) + b(k)
    end do
!$omp end target teams distribute parallel do
```

Device
code

```
do k=1,100
    write (*,*) c(k)
end do
```

Host code

target map (map_type)

When pointers are dynamically allocated, number of elements to be mapped must be explicitly specified

N – the number of elements to be copied

Note:

C++ : array[start : length]

Fortran: array[start : end]

Minimize Copy Overhead

```
integer :: a(100), b(100), c(100)
do k=1,100
    a(k) = 1
    b(k) = 1
end do
```

Host code

```
!$omp target data map(to: a[0:N], b[0:N])
map(tofrom: c[0:N])
    <update c somehow>
!$omp end target data
```

Device code

```
do k=1,100
    write (*,*) c(k)
end do
```

Host code

```
!$omp target data map(to: a[0:N], b[0:N])
map(tofrom: c[0:N])
    <update c somehow>
!$omp end target
```

Device code

- What if we need **a** and **b** in multiple target regions?
- Data movement overhead
- Solution:
 - target enter data
 - target update

Target data enter construct

```
integer :: a(100), b(100), c(100)
do k=1,100
    a(k) = 1
    b(k) = 1
end do
```

Host code

```
!$omp target enter data map(to: a[0:N], b[0:N], c[0:N])
!$omp target
    <update c somehow>
!$omp end target
!$omp target update from (c[0:N])
```

Device code

```
do k=1,100
    write (*,*) c(k)
end do
```

Host code

```
!$omp target
    <update c somehow>
!$omp end target
!$omp target exit data map(from: C[0:N])
```

Device code

target enter requires closing construct, **target exit**

Maps variables

Code execution not offloaded

target update

Copies data between host and device

enter data and exit data are standalone directives

OpenMP Offload Constructs

■ Device Code

- ★ • **omp target** *[clause[[,]clause]...]*
structured-block
- ★ • **omp declare target** *[function-definitions-or-declarations]*
- ★ • **omp declare target** *[variable-definitions-or-declarations]*

■ Worksharing

- ★ • **omp teams** *[clause[[,]clause]...]*
structured-block
- ★ • **omp distribute**
[clause[[,]clause]...] *for-loops*

■ Memory operations

- ★ • **map** (*[[map-type-modifier[,]]map-type:]*
list) *map-type := alloc | tofrom | to |*
from | release | delete *map-type-*
modifier := always
- ★ • **omp target data** *clause[[[,] clause]...]*
structured-block
- ★ • **omp target enter data**
clause[[[,]clause]...]
- ★ • **omp target exit data**
clause[[[,]clause]...]
- ★ • **omp target update** *clause[[[,]clause]...]*

Demo

Code

```
program vector_add
  use omp_lib
  integer :: a(100), b(100), c(100)
  do k=1,100
    a(k) = 1
    b(k) = 1
  end do

  !$omp target teams distribute parallel do map
  (to:a) map(to:b) map(tofrom:c)
    do k=1,100
      c(k) = a(k) + b(k)
    end do
  !$omp end target teams distribute parallel do
  do k=1,10
    write (*, '(1x,i0)', advance='no') c(k)
  end do
  write (*,*) '...'
end program vector_add
```

```
$ ifx -qopenmp -fopenmp-targets=spir64 vectadd.f90
$ ./a.out
 2 2 2 2 2 2 2 2 2 2 ...
$ export OMP_TARGET_OFFLOAD="MANDATORY"
$ export LIBOMPTARGET_PLUGIN=LEVEL0
$ export LIBOMPTARGET_DEBUG=1
$ ./a.out
Libomptarget --> Init target library!
Libomptarget --> Initialized OMPT
Libomptarget --> Loading RTLs...
Libomptarget --> Checking user-specified plugin
'libomptarget.rtl.level0.so'...
Libomptarget --> Loading library
'libomptarget.rtl.level0.so'...
Target LEVEL0 RTL --> Init Level0 plugin!
Target LEVEL0 RTL --> omp_get_thread_limit()
returned 2147483647
Target LEVEL0 RTL --> omp_get_max_teams() returned 0
Target LEVEL0 RTL --> Init Level0 plugin!
Target LEVEL0 RTL --> omp_get_thread_limit()
returned 2147483647
Target LEVEL0 RTL --> omp_get_max_teams() returned 0
Libomptarget --> Successfully loaded library
'libomptarget.rtl.level0.so'!
... .
```

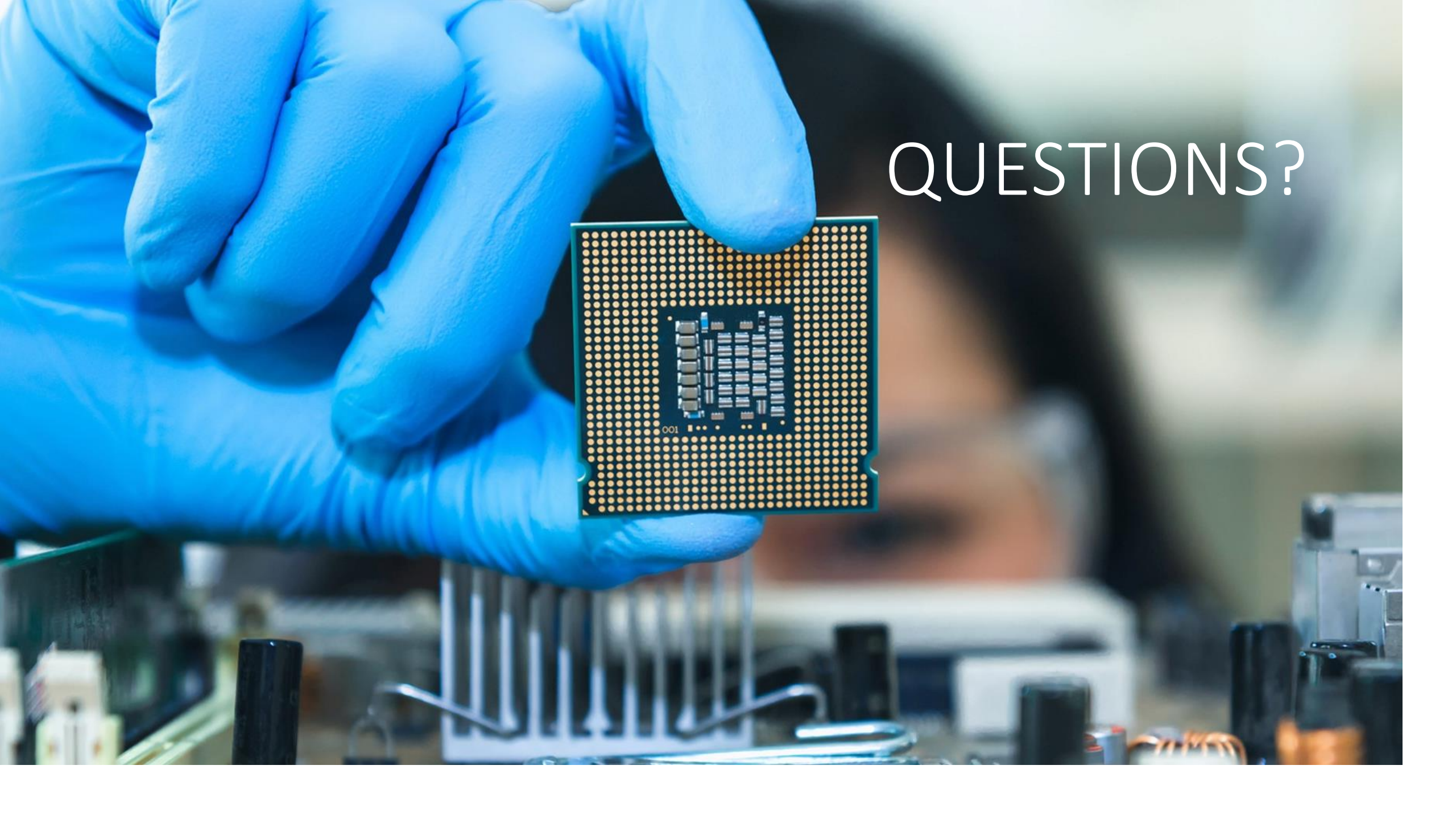
Code

```
program vector_add
  use omp_lib
  integer :: a(100), b(100), c(100)
  do k=1,100
    a(k) = 1
    b(k) = 1
  end do
  !$omp target teams distribute parallel do map
  (to:a) map(to:b) map(tofrom:c)
    do k=1,100
      c(k) = a(k) + b(k)
    end do
  !$omp end target teams distribute parallel do
  do k=1,10
    write (*, '(1x,i0)', advance='no') c(k)
  end do
  write (*,*) '...'
end program vector_add
```

```
$ export LIBOMPTARGET_DEBUG=0
$ export LIBOMPTARGET_INFO=-1
$ ./a.out
Libomptarget device 0 info: Entering OpenMP kernel
at unknown:0:0 with 10 arguments:
Libomptarget device 0 info: tofrom(unknown) [400000]
Libomptarget device 0 info: to(unknown) [400000]
Libomptarget device 0 info: to(unknown) [400000]
Libomptarget device 0 info: firstprivate(unknown) [0]
Libomptarget device 0 info: firstprivate(unknown) [0]
Libomptarget device 0 info: firstprivate(unknown) [0]
Libomptarget device 0 info: firstprivate(unknown) [0]
Libomptarget device 0 info: firstprivate(unknown) [0]
Libomptarget device 0 info: firstprivate(unknown) [0]
Libomptarget device 0 info: firstprivate(unknown) [0]
Libomptarget device 0 info: Creating new map entry
with HstPtrBegin=0x00007ffc6f0441b0,
TgtPtrBegin=0x000000000168b000, Size=400000,
DynRefCount=1, HoldRefCount=0, Name=unknown
Libomptarget device 0 info: Copying data from host
to device, HstPtr=0x00007ffc6f0441b0,
TgtPtr=0x000000000168b000, Size=400000, Name=unknown
```

What else?

- [OpenMP* Offload Basics in DevCloud](#) (with lab!)
- <https://www.openmp.org/specifications/>



QUESTIONS?



Notices & Disclaimers

Performance varies by use, configuration, and other factors. Learn more at [www.Intel.com/PerformanceIndex](https://www.intel.com/PerformanceIndex).

Performance results are based on testing as of dates shown in configurations and may not reflect all publicly available updates. See configuration disclosure for details.

Your costs and results may vary.

Intel technologies may require enabled hardware, software or service activation.

© Intel Corporation. Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Other names and brands may be claimed as the property of others.