LRZ Workshop June 2023

Parallelizing heterogenous applications with Intel ® OpenMP and OpenMP offloading

Advanced Topics

Alina Shadrina

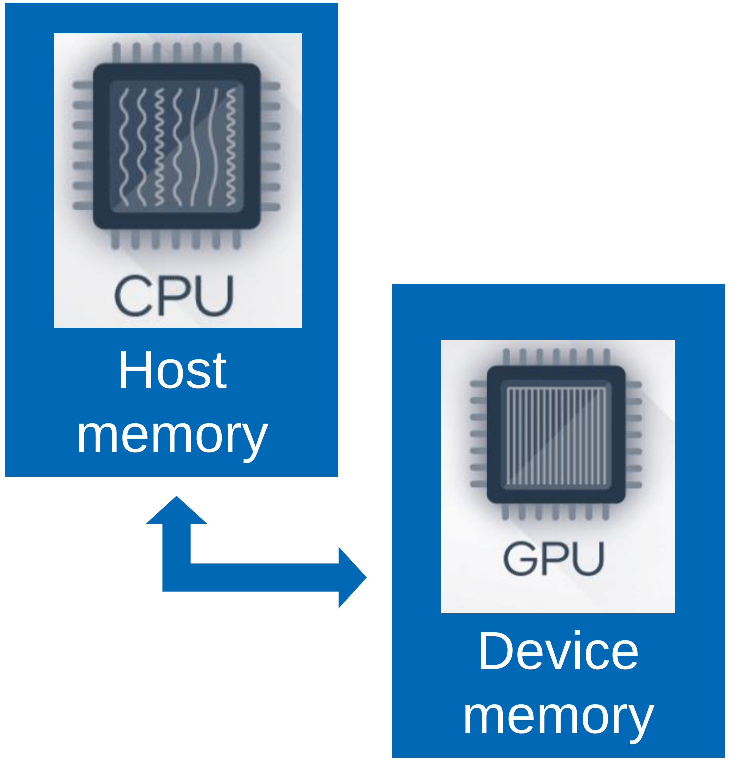
alina.shadrina@intel.com

Agenda

- OpenMP* Offload Compiler Support
- Environment variables
- OpenMP* Target Construct
- Managing Device Data
- Demo
- Using openMP Offload with Intel® oneAPI Math Kernel Library (MKL)

OpenMP* Offload Compiler Support

Device Model

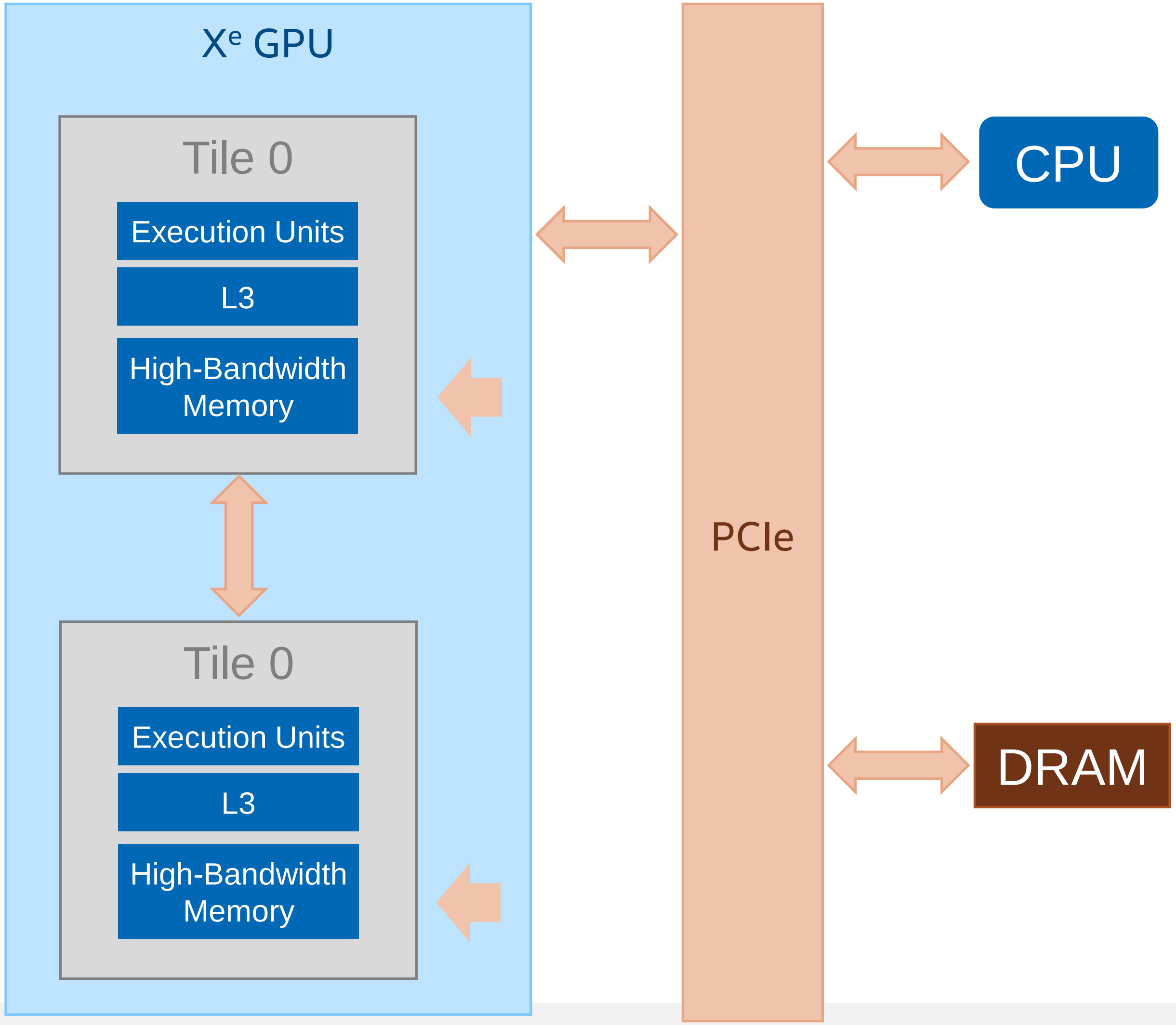


Host memory

Device memory

- 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

Intel X^e Multi-Tile GPU Architecture



- Tiles are independent
  - No global schedulers
  - No global commands affecting all tiles
  - No global state
  - Can work concurrently
- Tiles can communicate over memory
  - Use GPU semaphores for synchronization

OpenMP* Offload Compiler Support

- Intel® C++ Compiler

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

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

- Intel® Fortran Compiler

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

- Hardware Supported: Intel® Gen9
- OpenMP directives supported in the icx and ifx compilers for GPU and CPU
- On Linux*, GCC* 4.8.5 or higher must be installed for host code compilation. This is to avoid any incompatibilities due to a changed C++ Application Binary Interface (ABI).

OpenMP* Offload Compiler Support

- Ahead-of-Time compilation supported

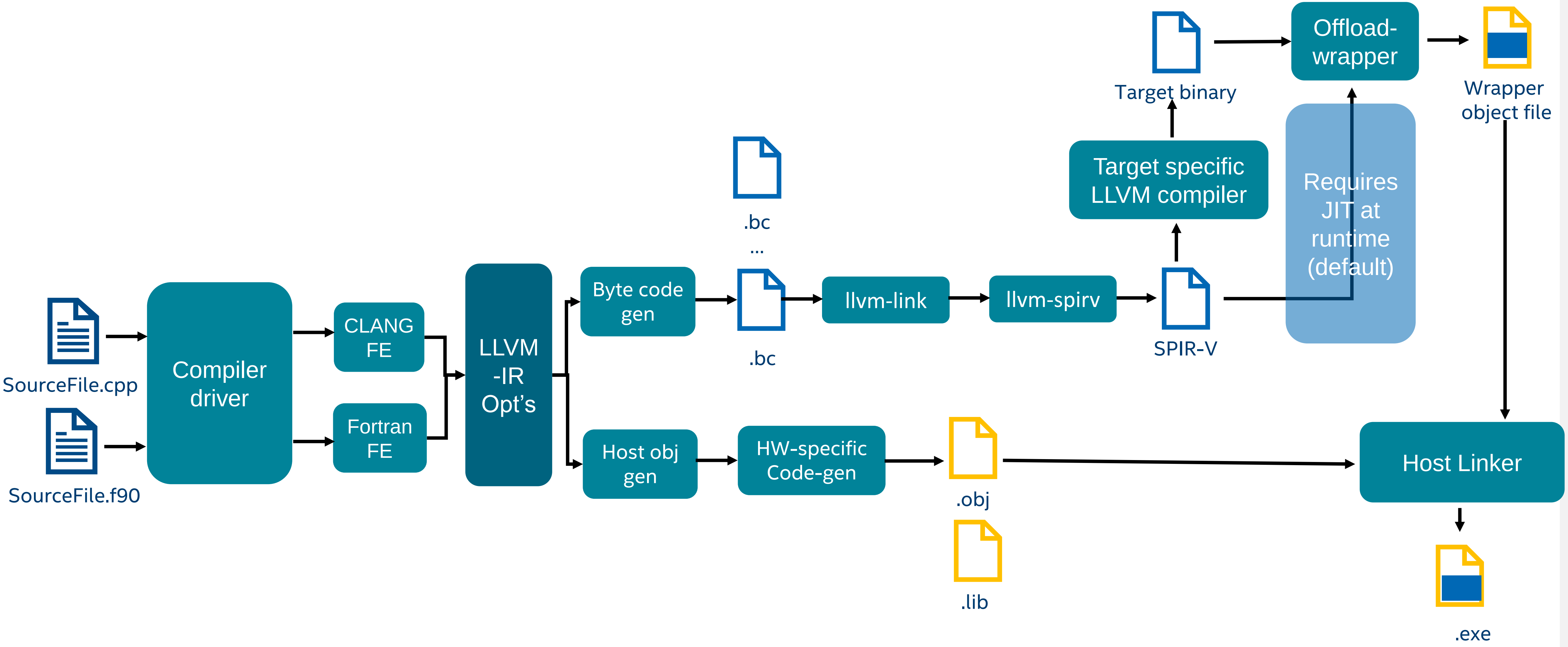
```
icx -fiopenmp -fopenmp-targets=spir64_gen -Xopenmp-target-backend "-device *"
<source>.cpp
```

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

Xopenmp-target

Offload Compilation Flow

-fopenmp-targets=spir64_gen
-Xopenmp-target-backend "-device *"



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_PLUGIN_PROFILE: Add profiling info
  - Try `T,usec`
  - LIBOMPTARGET_PROFILE is deprecated
- LIBOMPTARGET_INFO: data-mappings and kernel execution
  - 32-bit field to enable or disable different types of information
  - -1 – enable every bit set

oneAPI Level Zero Specification

Supported Environment Variables

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 *[clause[[,]clause]…]* *structured-block*	**!$omp target** *[clause[[,]clause]…]* *structured-block* **!$omp end target**
#pragma omp target data *[clause[[,]clause]…]* *structured-block*	**!$omp target** *[clause[[,]clause]…]* *structured-block* **!$omp end target data**
#pragma omp teams *[clause[[,]clause]…]* *structured-block*	**!$omp teams** *[clause[[,]clause]…]* *structured-block*
#pragma omp distribute *[clause[[,]clause]…]* *structured-block*	**!$omp distribute** *[clause[[,]clause]…]* *structured-block*

OpenMP* 5.1 - What's new?

- Fortran 2008 is now fully supported and initial support for Fortran 2018 has been added
- C++11 attributes in addition to pragmas
  - [[omp::directive (parallel for)]] -   #pragma omp parallel for

- New directives

Scope	Improve teams performance
Assume	Optimization invariants
Interop	interoperability
Dispatch	Variant substitution
Error	compiler or runtime to display a messages
nothing	utility

- Deprecated and replaced:
  - omp_target_is_accessible
  - omp_get_mapped_ptr
  - omp_calloc
  - omp_aligned_alloc
  - omp_realloc
  - omp_set_num_teams
  - omp_set_teams_thread_limit
  - omp_get_max_teams
  - omp_get_teams_thread_limit

Press Release: OpenMP ARB releases OpenMP 5.1 with Vital Usability Enhancements

OpenMP* Target Construct
Fortran

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, in_reduction, map, allocate, if
Sync: nowait, depend

`if` - When an `if` clause is present and the `if` clause expression evaluates to *false*, the target region is executed by the **host device in the host data environment.**

2.12.5 `target` Construct

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

2.12.5 `target` Construct

Target Device Construct for Multi-Tile GPUs

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

Host code

```
!$omp target device(0) subdevice (0, 2:5)
        do k=1,100
                c(k) =  a(k) + b(k)
        end do
!$omp end target
```

runs on tiles 2, 3, 4, and 5

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
- How to utilize multi-tile GPU?
- **`SUBDEVICE  ( [level,] start [:length [:stride]] )`**
  - Level - non-negative int constant; default 0
  - Start - non-negative int expression.
  - Length - positive int expression; default 1
  - Stride - positive int expression; default 1

Intel® Fortran Compiler Classic and Intel® Fortran Compiler Developer Guide and Reference: SUBDEVICE

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

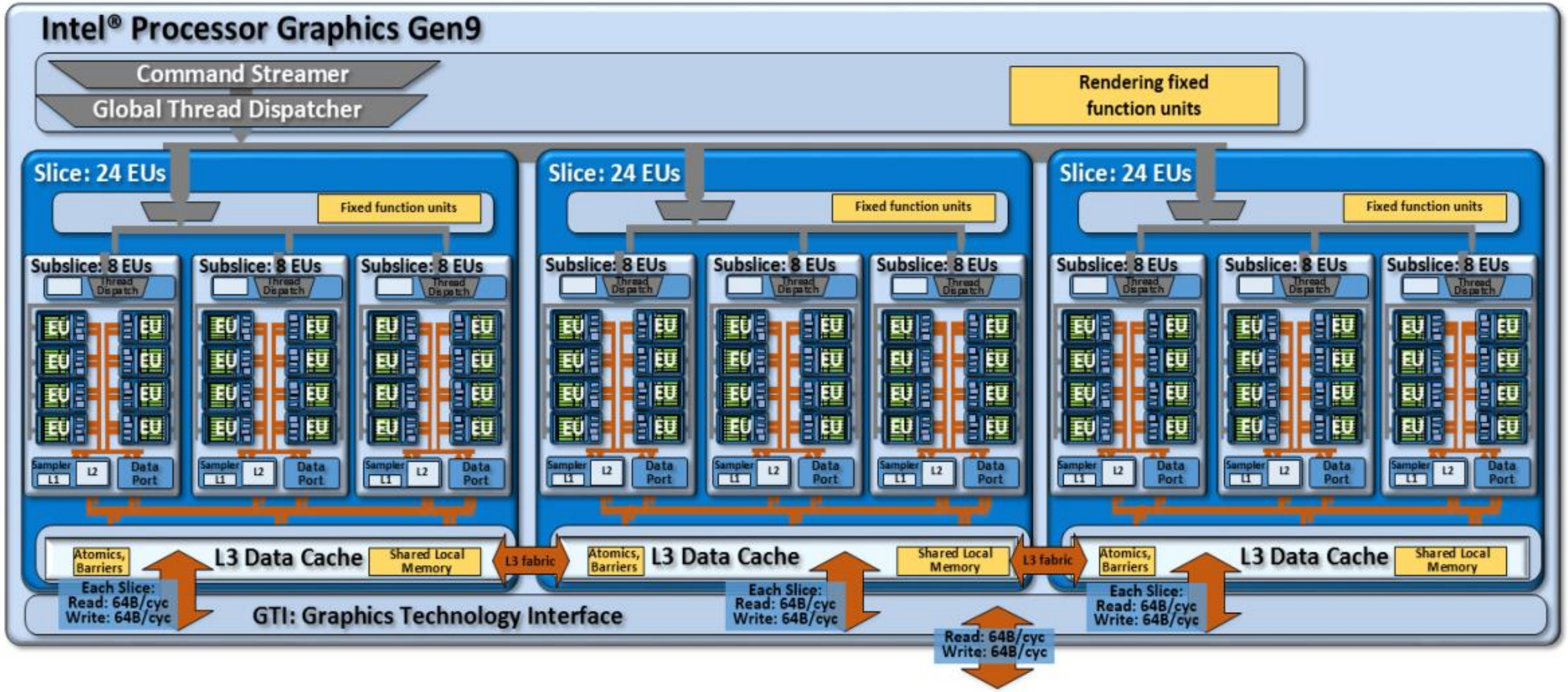


Figure 8: Another potential product design that instantiates the compute architecture of Intel® processor graphics gen9. This design is composed of three slices, of three subslices each for a total of 72 EUs.

[The Compute Architecture of Intel(R) Processor Graphics Gen9. Version 1.0]()

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`

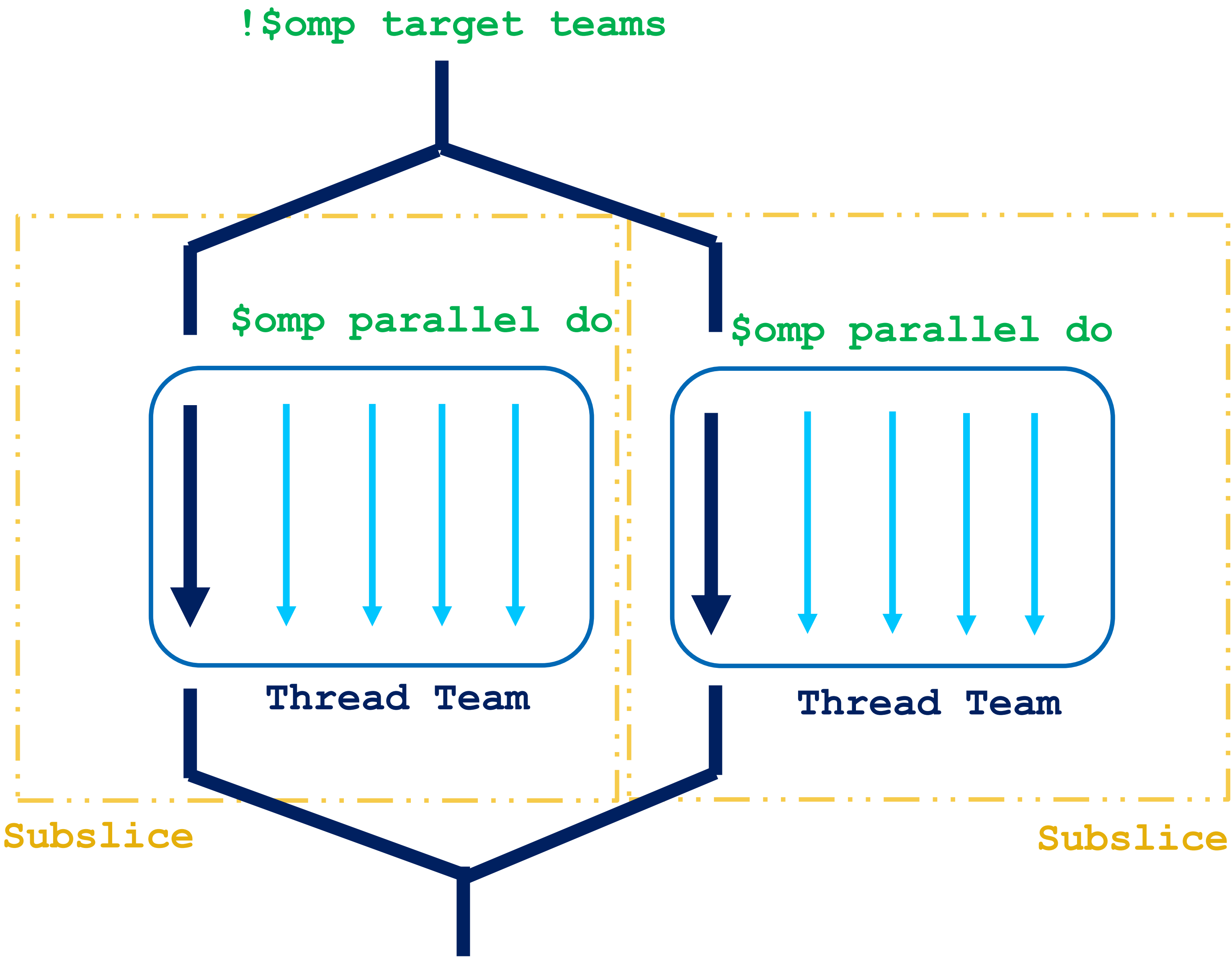
creates a *league* of teams where the primary thread of each team executes the `teams` region.

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

2.13.21 `target teams` Construct

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



2.7 teams Construct

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.

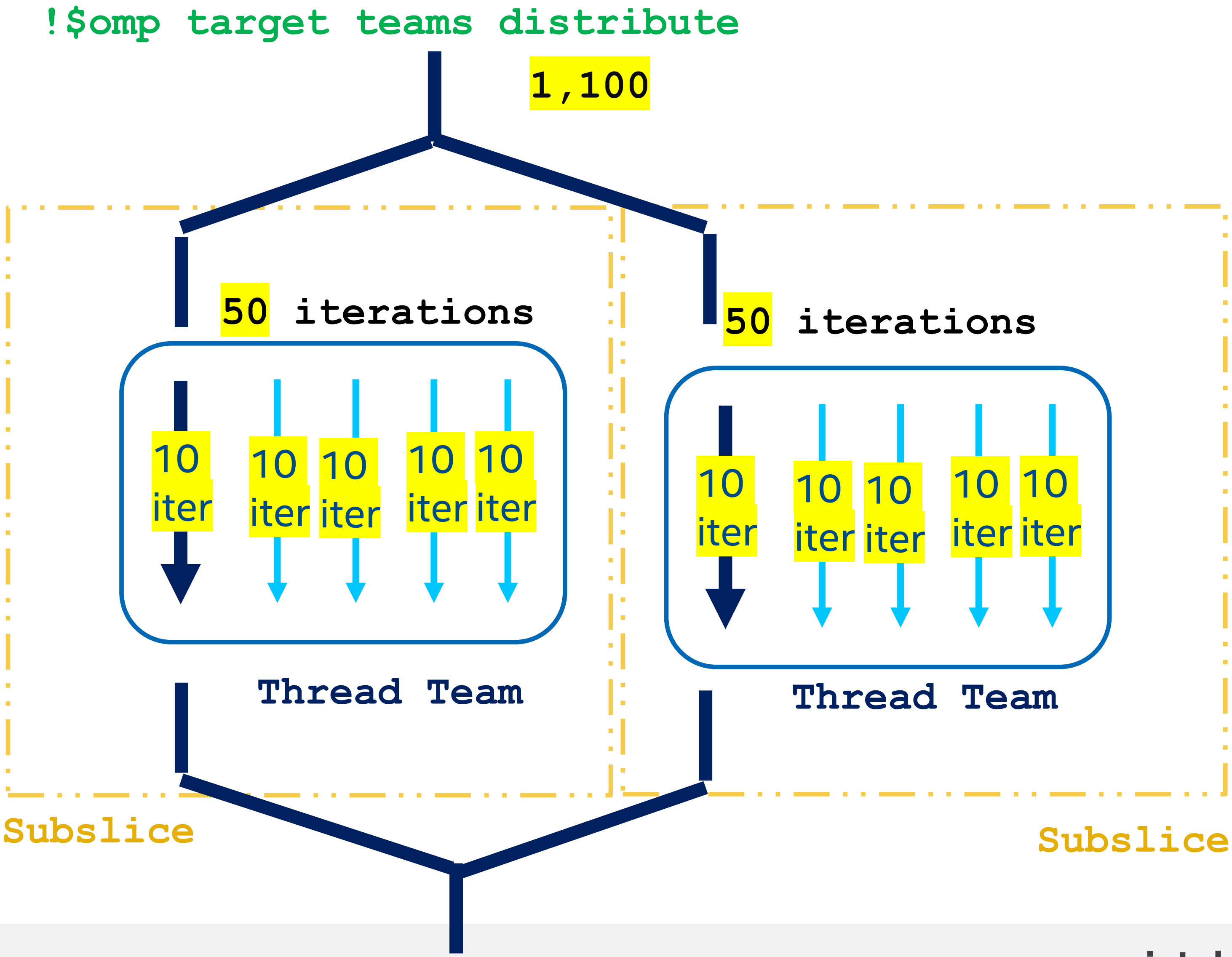
`target teams distribute parallel do`

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

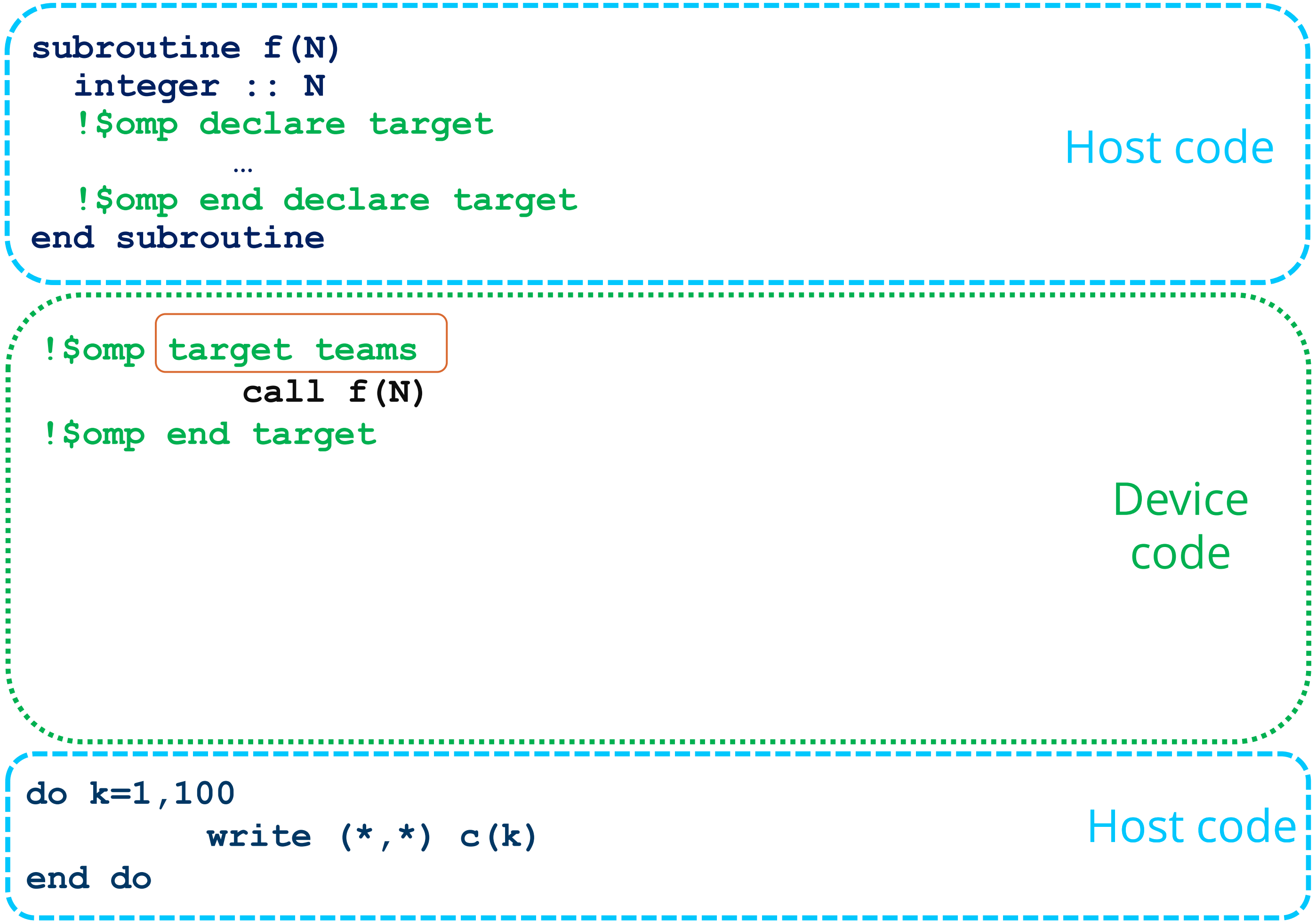
2.13.11 `teams distribute` Construct
2.13.25 Target Teams Distribute Parallel Worksharing-Loop Construct

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)`

2.12.7 `declare target` Directive

Asynchronous Target Regions

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

Host code

```
!$omp task depend(out: a)
    call init_vector(a,  N)
!$omp end task
!$omp task depend(out: b)
    call init_vector(b,  N)
!$omp end task
!$omp target map(to:a, b) map(tofrom:c) nowait depend(in:a, b)
depend(out:c)
     call vector_add(a, b, c, N);
!$omp end target
!$omp targetmap(to:c) map(tofrom:c) nowait depend(in:c)
depend(out:c)
   call vector_increment(c, N)
!$omp end target
!$omp taskwait
```

Device code

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

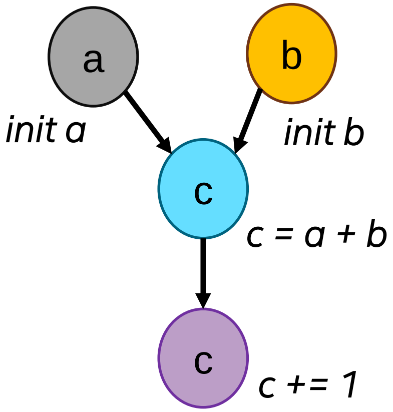
Host code

`target [clause]`

Offloads a code region to a target device

Synchronous by default

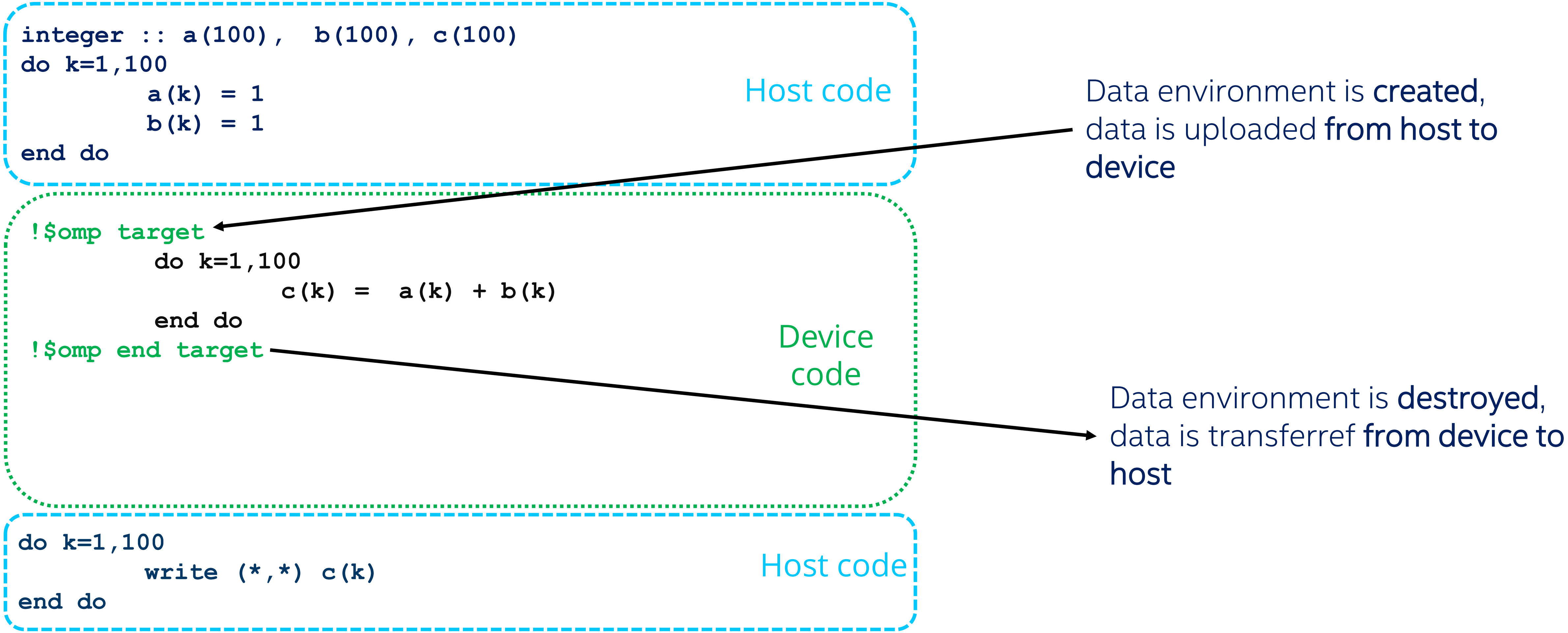
- nowait
- depend ([depend-modifier,]dependence-type : locator-list)



[2.17.11 depend Clause](#)
[15.6 nowait Clause](#)

Managing Device Data
Fortran

Device Data Environment



```
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

Data environment is **created**, data is uploaded **from host to device**

Data environment is **destroyed**, data is transferref **from device to host**

[1.4.6 OpenMP Memory Consistency]

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>

2.19.7 Data-Mapping Attribute Rules, Clauses, and Directives

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

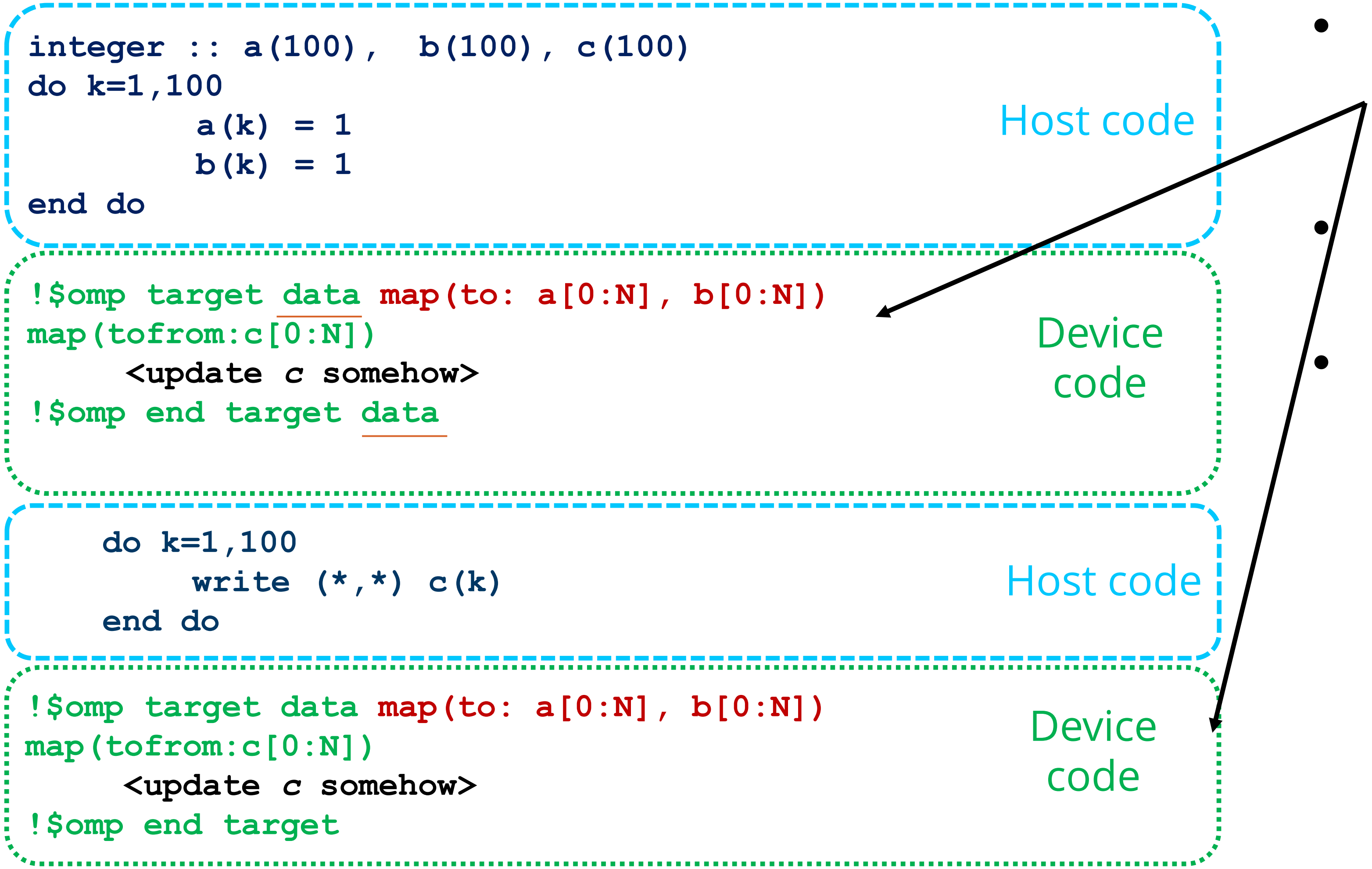
N – the number of elements to be copied

Note:

C++ : array[start : length]
Fortran: array[start : end]

2.19.7 Data-Mapping Attribute Rules, Clauses, and Directives

Minimize Copy Overhead



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

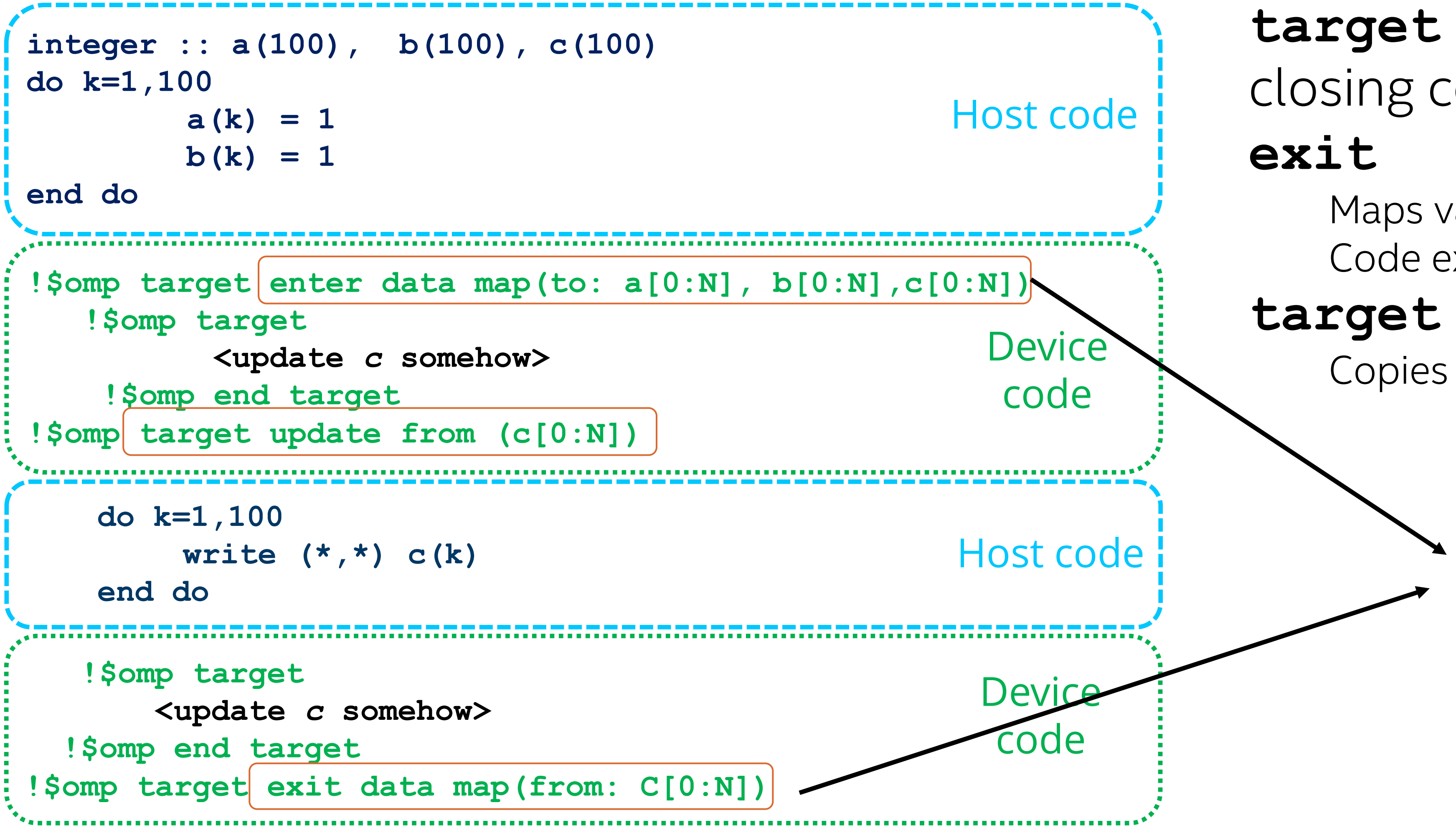
```
!$omp target data map(to: a[0:N], b[0:N])
map(tofrom:c[0:N])
     <update c somehow>
!$omp end target data
```

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

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

- 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
```

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

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

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

`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

[2.12.3 `target enter data` Construct]

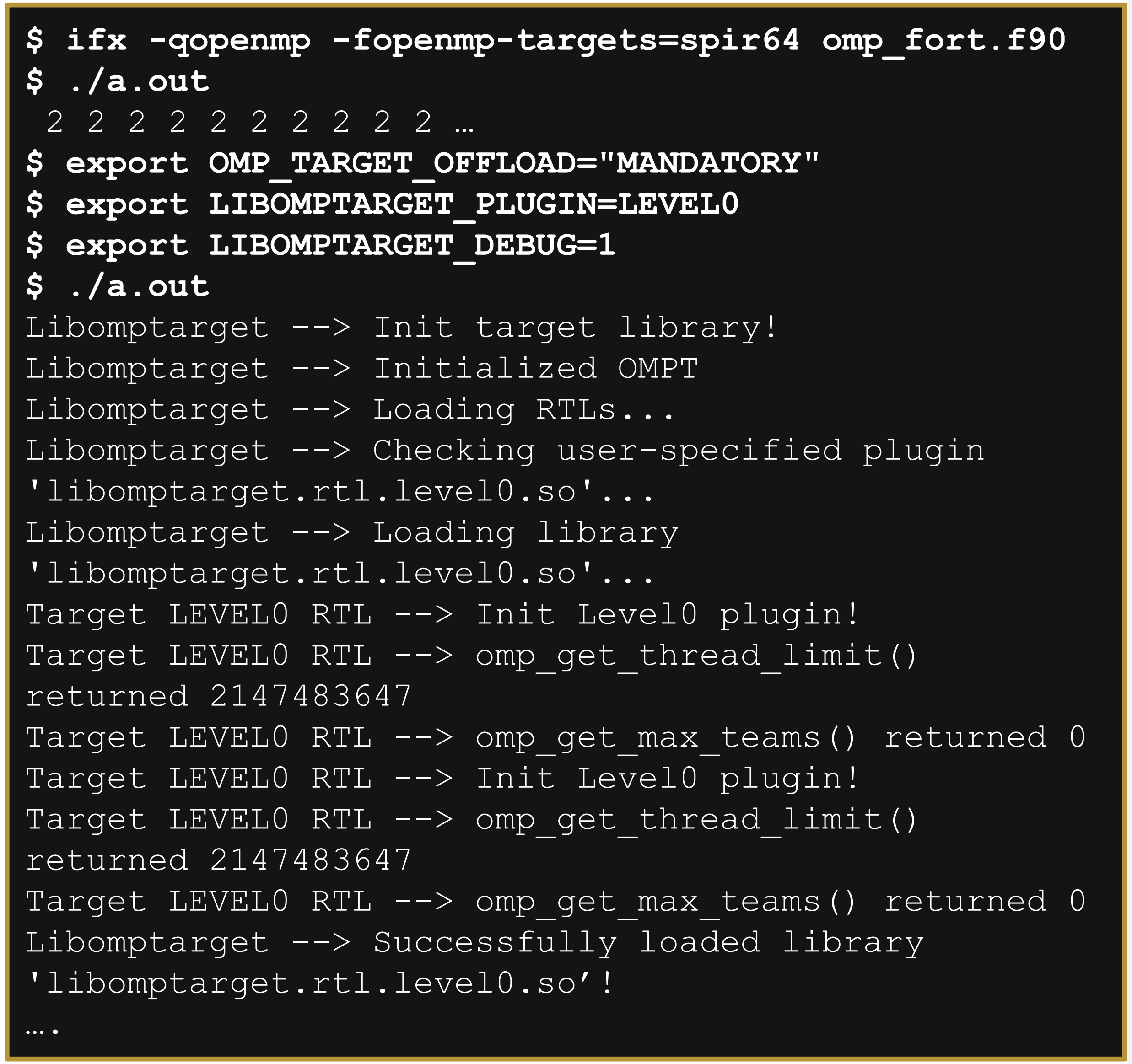
Demo

Fortran Code Sample

```fortran
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
```



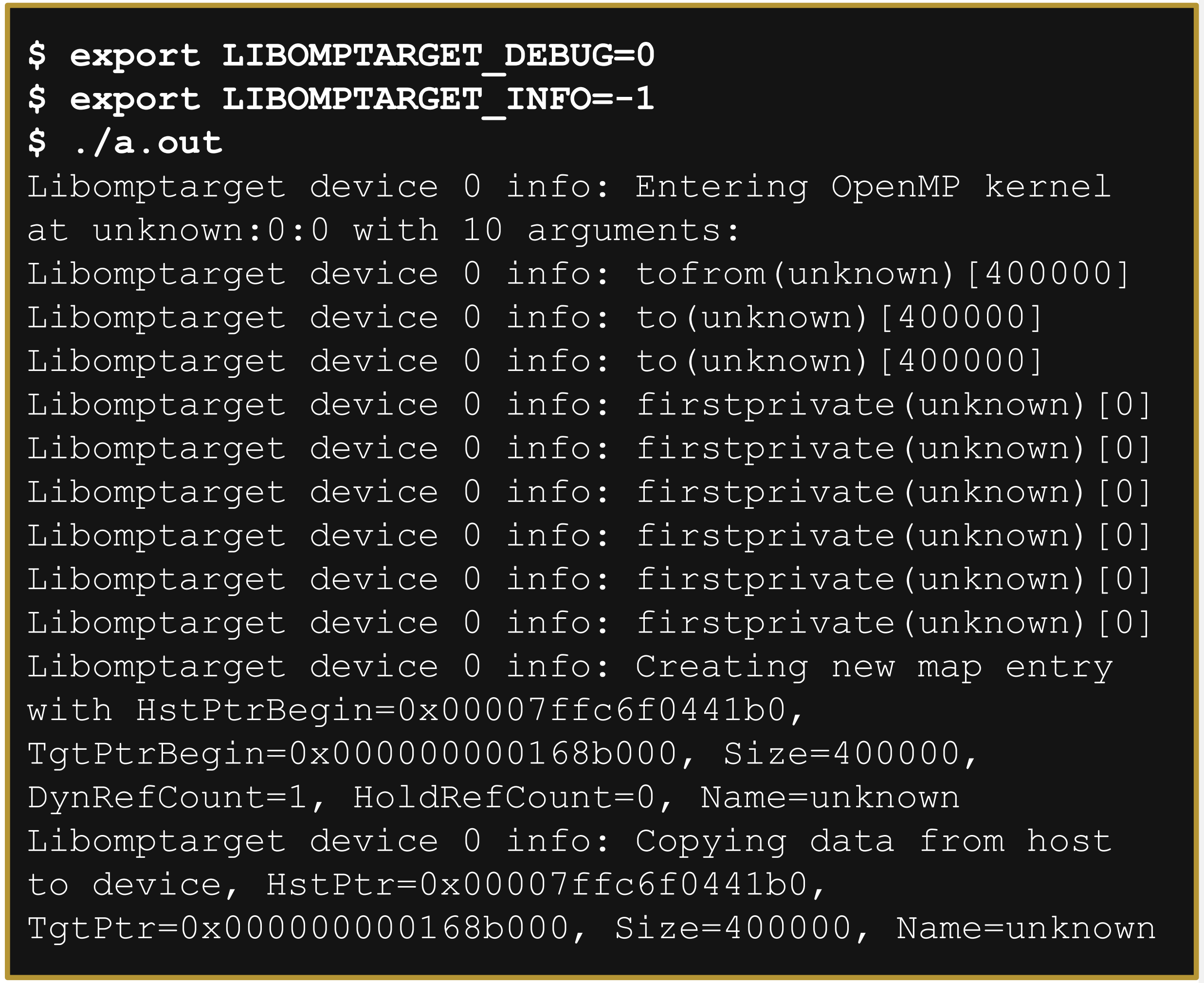
```
$ ifx -qopenmp -fopenmp-targets=spir64 omp_fort.f90
$ ./a.out
 2 2 2 2 2 2 2 2 2 2 …
$ export OMP_TARGET_OFFLOAD="MANDATORY"
$ export LIBOMPTARGET_PLUGIN=LEVEL0
$ export LIBOMPTARGET_DEBUG=1
$ ./a.out
Libomptarget --> Init target library!
Libomptarget --> Initialized OMPT
Libomptarget --> Loading 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’!
….
```

Fortran Code Sample

```fortran
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: 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!)
- openMP Specification
- C/C++ OpenMP* and SYCL* Composability
- Three Quick, Practical Examples of OpenMP* Offload to GPUs

Using openMP Offload with Intel® oneAPI Math Kernel Library (MKL)

Calling MKL OpenMP offload (OpenMP => 5.1)

```
#include <mkl.h>
int main()
{
   /* Allocate memory. */
   MKL_INT n = 1337;
   double *A = malloc(sizeof(double) * n*n );
   double *B = malloc( sizeof(double) * n*n );
   double *C = malloc( sizeof(double) * n*n );
   /* Initialise A and B with your favourite values here. */
   for ( int i = 0; i < n; ++i )
   for ( int j = 0; j < n; ++j )
   {
      A[ i + j*n ] = i*j;
      B[ i + j*n ] = i+j;
   }
   /* Form matrix-matrix product C = A*B */
   cblas_dgemm(CblasColMajor, CblasNoTrans, CblasNoTrans,
               n, n, n, 1.0, A, n, B, n, 0.0, C, n);
   return 0;
}
```

- Starting point: Classic C code calling the standard CBLAS MKL interface.

Shall be executed on GPU

Calling MKL OpenMP offload (OpenMP => 5.1)

```
cblas_dgemm(CblasColMajor, CblasNoTrans, CblasNoTrans,
               n, n, n, 1.0, A, n, B, n, 0.0, C, n);
```

```
#pragma omp target data map(to:A[0:n*n],B[0:n*n]) map(from:C[0:n*n])
{
/* This will run the CPU version! */
   cblas_dgemm(CblasColMajor, CblasNoTrans, CblasNoTrans,
               n, n, n, 1.0, A, n, B, n, 0.0, C, n);
}
```

- Remember: omp target data map only creates a data environment on the GPU.
- A and B get transferred to the GPU, C is allocated on the GPU.
- Cblas_dgemm will be calculated on the CPU.
- C gets overwritten by uninitialized data from the GPU.

Calling MKL OpenMP offload (OpenMP => 5.1)

```
cblas_dgemm(CblasColMajor, CblasNoTrans, CblasNoTrans,
               n, n, n, 1.0, A, n, B, n, 0.0, C, n);
```

```
#include <mkl.h>
#include <mkl_omp_offload.h>
```

```
#pragma omp target data map(to:A[0:n*n],B[0:n*n]) map(from:C[0:n*n])
{
#pragma omp dispatch
cblas_dgemm(CblasColMajor, CblasNoTrans, CblasNoTrans,
               n, n, n, 1.0, A, n, B, n, 0.0, C, n);
}
```

- Include mkl_omp_offload.h header file
- Instruct the compiler to use the offload version by “omp dispatch” (available since OpenMP 5.1)
- Compile with –fopenmp-version=51

Calling MKL OpenMP offload (OpenMP => 5.1)

- Compile:

```
icx -fiopenmp -fopenmp-version=51 -fopenmp-targets=spir64 -qmkl \
   -o dgemm_sample.o -c dgemm_sample.c
```

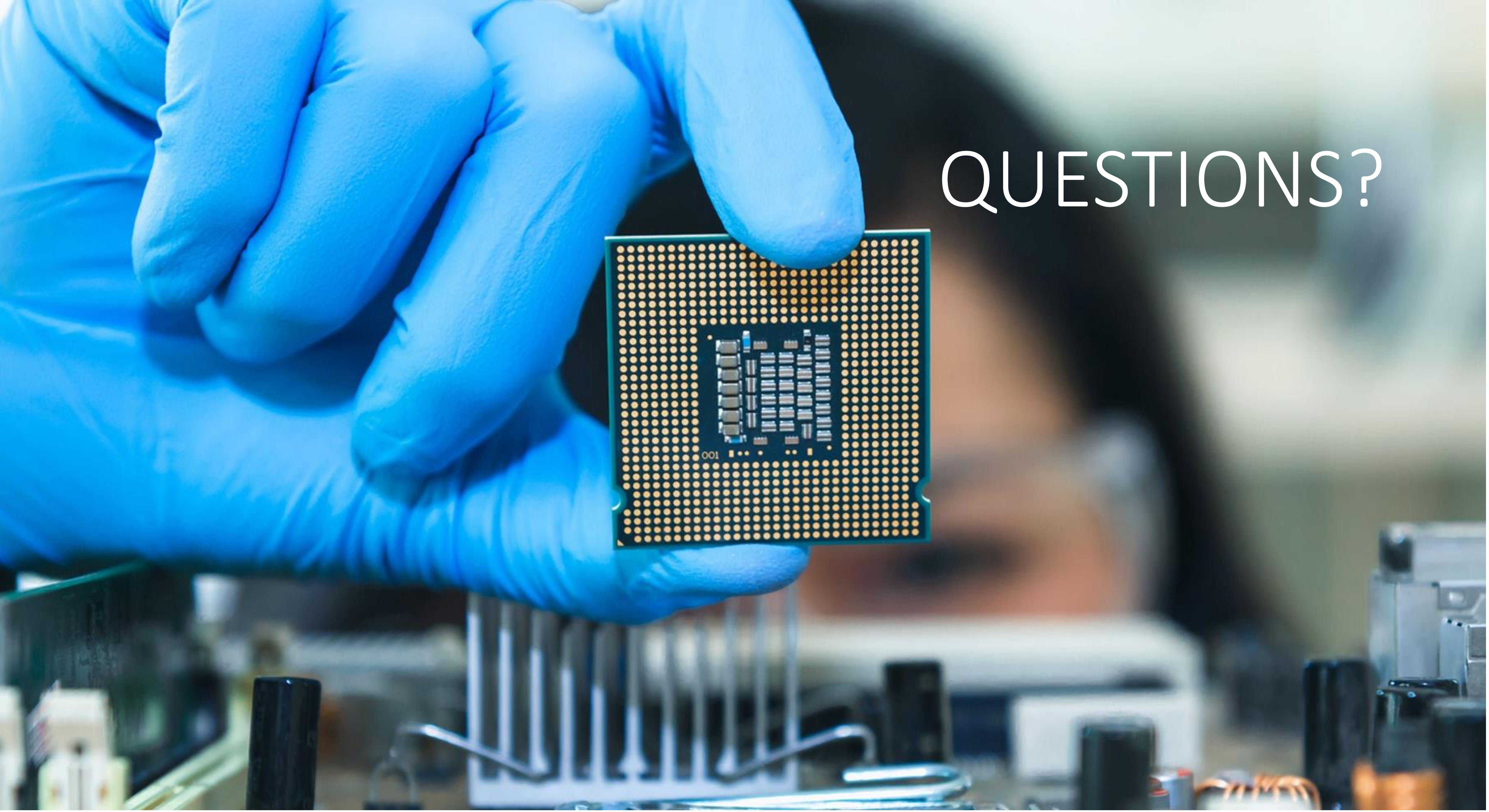
- Link:

```
icx -fiopenmp -fopenmp-version=51 -fopenmp-targets=spir64 -qmkl \
   -o dgemm_sample dgemm_sample.o
```

- Many more options (64/32-bit integers, static/dynamic linking, Linux*/Windows*, etc.) are available through the:

Intel® oneAPI Math Kernel Library Link Line Advisor

QUESTIONS?



intel®

Notices & Disclaimers

Performance varies by use, configuration, and other factors. Learn more at 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.