HLRS Workshop

# Parallelizing heterogenous applications with Intel ® OpenMP and OpenMP offloading

Alina Shadrina

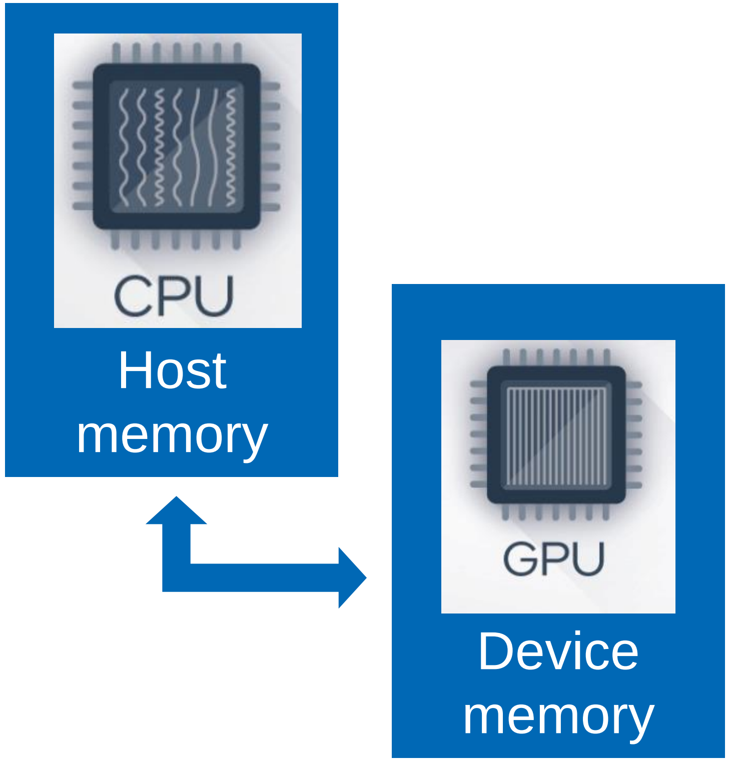
alina.shadrina@intel.com

# Agenda

- OpenMP* Offload Compiler Support
- OpenMP* Target Construct
- Managing Device Data
- Environment variables
- Mixing of OpenMP* and SYCL

# 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 -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](Xopenmp-target)

# 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

[oneAPI Level Zero Specification]

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

# OpenMP* Target Construct

# Target construct

```
float *a, *b, *c;
for (int i=0; i<N; i++){
   a[i] = 1;
   b[i] = 1;
}
```

Host code

```
#pragma omp target
{

 for (int i=0; i<N; i++){
    c[i] = a[i] + b[i];
   }
}
```

Device code

```
for (int i=0; i<N; i++){
   std::cout << c[i] << std::endl;
}
```

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

```
float *a, *b, *c;
for (int i=0; i<N; i++){
   a[i] = 1;
   b[i] = 1;
}
```

Host code

```
#pragma omp target device (0)
{

 for (int i=0; i<N; i++){
    c[i] = a[i] + b[i];
   }
}
```

Device code

```
for (int i=0; i<N; i++){
   std::cout << c[i] << std::endl;
}
```

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

```
float *a, *b, *c;
for (int i=0; i<N; i++){
   a[i] = 1;
   b[i] = 1;
}
```

Host code

```
#pragma omp target parallel for
{

 for (int i=0; i<N; i++){
    c[i] = a[i] + b[i];
   }
}
```

Device code

```
for (int i=0; i<N; i++){
   std::cout << c[i] << std::endl;
}
```

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

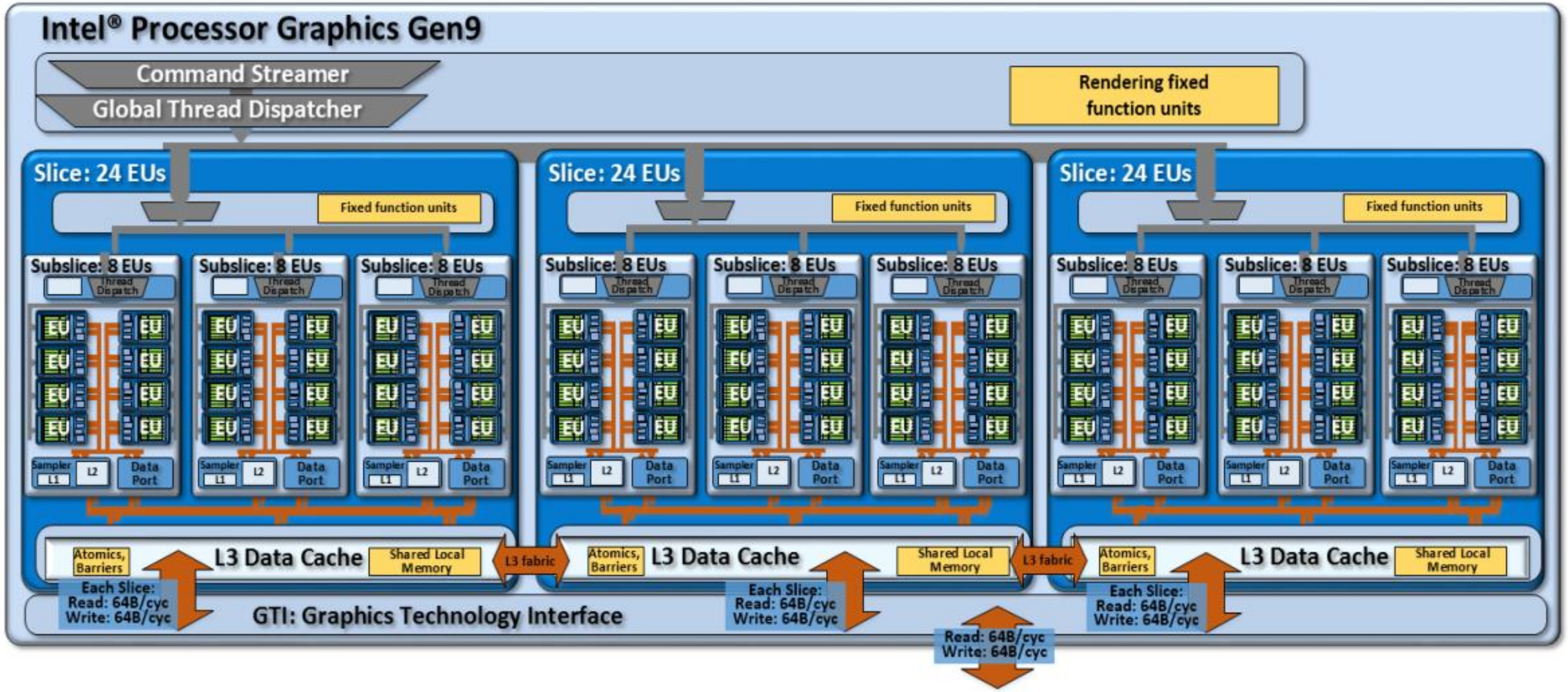



*Figure 8: Another potential product design that instantiates the compute architecture of Intel® processor graphics gen9. This design is composed of three slices, of three subslices each for a total of 72 EUs.*

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

# OpenMP* Device Parallelism

```
float *a, *b, *c;
for (int i=0; i<N; i++){
   a[i] = 1;
   b[i] = 1;
}
```

Host code

```
#pragma omp target teams
{

 for (int i=0; i<N; i++){
    c[i] = a[i] + b[i];
   }
}
```

Device code

```
for (int i=0; i<N; i++){
   std::cout << c[i] << std::endl;
}
```

Host code

## `target [clause]`

Offloads a code region to a target device

Sequential by default

## `target teams`

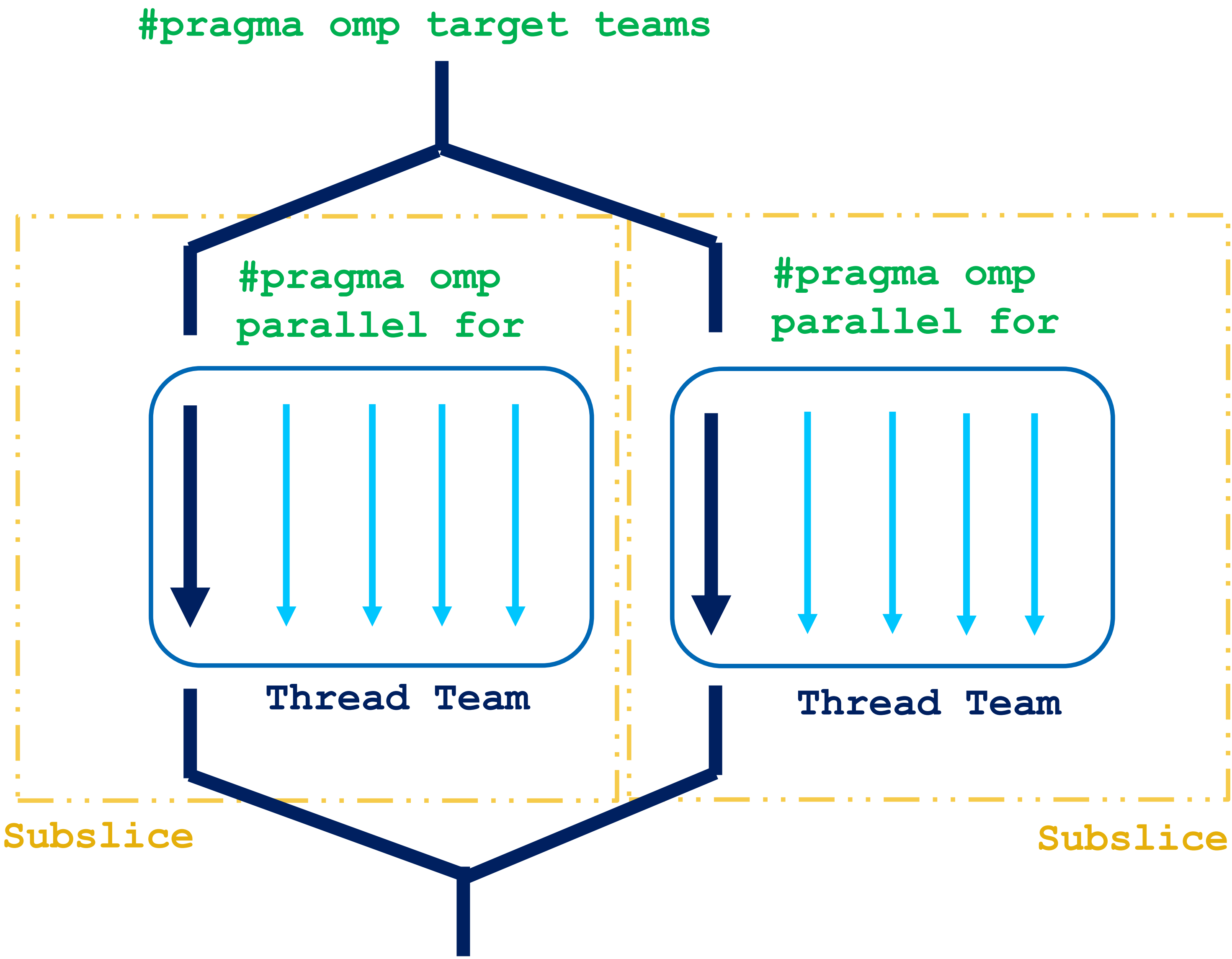
creates a *league* of teams where the primary thread of each team executes the `teams` region.

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 |



# OpenMP* Worksharing

```
float *a, *b, *c;
for (int i=0; i<N; i++){
   a[i] = 1;
   b[i] = 1;
}
```

Host code

```
#pragma omp target teams distribute parallel for
{

 for (int i=0; i<N; i++){
    c[i] = a[i] + b[i];
   }
}
```

Device code

```
for (int i=0; i<N; i++){
   std::cout << c[i] << std::endl;
}
```

Host code

**`target teams distribute`**

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

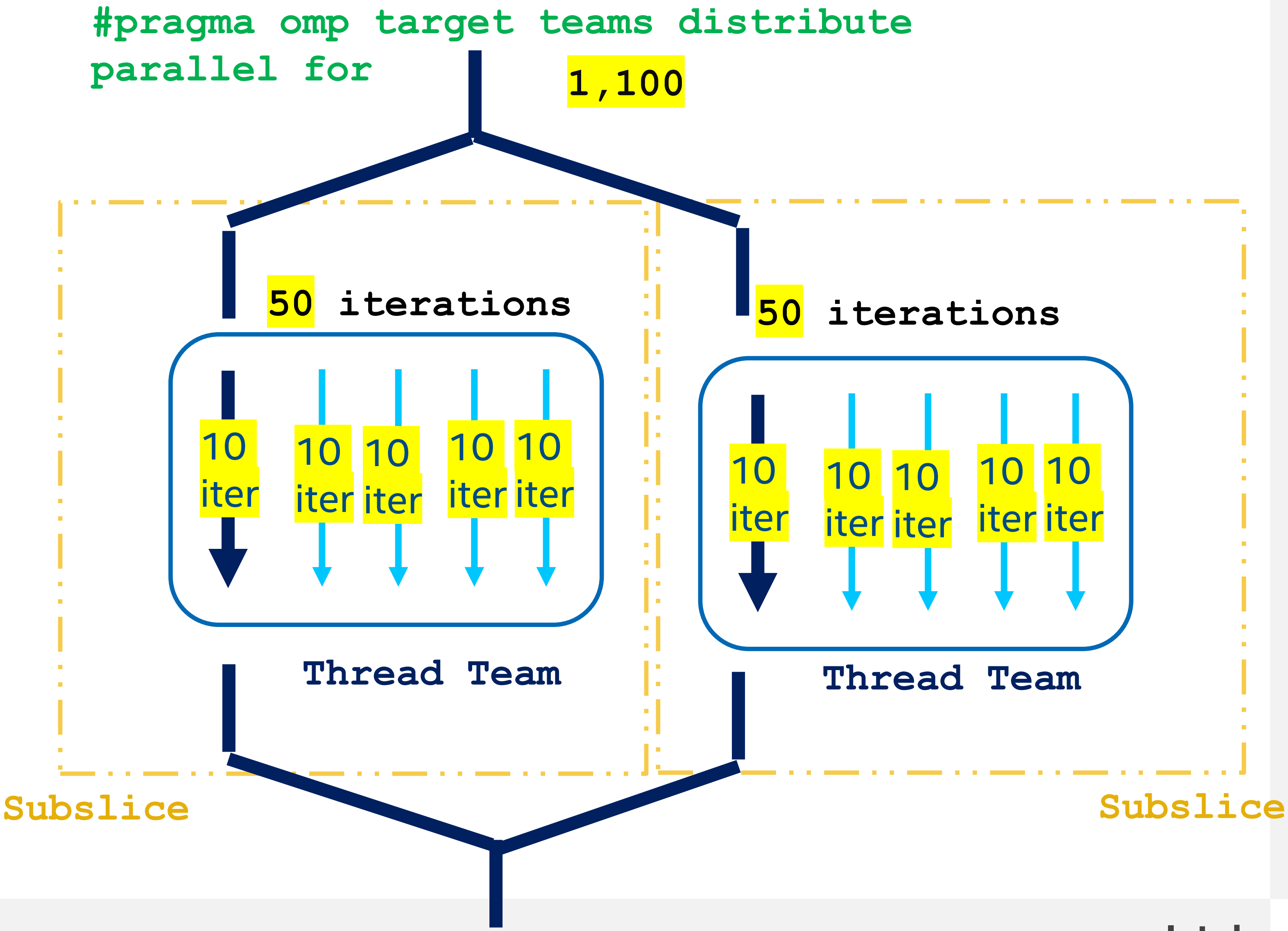
**`target teams distribute parallel for`**

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

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



# Calling functions inside Target region

```
#pragma omp declare target
void vector_add(){
…
}
#pragma omp end declare target
```

Host code

```
#pragma omp target teams
{

  vector_add()


}
```

Device code

```
for (int i=0; i<N; i++){
   std::cout << c[i] << std::endl;
}
```

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

# 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

```
float *a, *b, *c;
for (int i=0; i<N; i++){
    a[i] = 1;
    b[i] = 1;
}
```

Host code

```
#pragma omp target map(to:a) map(to:b)
map(tofrom:c)
{

 for (int i=0; i<N; i++){
     c[i] = a[i] + b[i];
   }
}
```

Device code

```
for (int i=0; i<N; i++){
    std::cout << c[i] << std::endl;
}
```

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

# Dynamically Allocated Data

```
float *a, *b, *c;
for (int i=0; i<N; i++){
   a[i] = 1;
   b[i] = 1;
}
```

Host code

```
#pragma omp target map(to:a[0:N]) map(to:b[0:N])
map(tofrom:c[0:N])
{

 for (int i=0; i<N; i++){
    c[i] = a[i] + b[i];
   }
}
```

Device code

```
for (int i=0; i<N; i++){
   std::cout << c[i] << std::endl;
}
```

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

```
float *a, *b, *c, *d;
for (int i=0; i<N; i++){
   a[i] = 1;
   b[i] = 1;
}
```

Host code

```
#pragma omp target map(to:a[0:N])
map(to:b[0:N]) map(tofrom:c[0:N])
{
…
}
```

Device code

```
for (int i=0; i<N; i++){
   std::cout << c[i] << std::endl;
}
```

Host code

```
#pragma omp target map(to:a[0:N])
map(to:b[0:N]) map(tofrom:d[0:N])
{
…
}
```

Device code

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

# Target data enter construct

```
float *a, *b, *c, *d;
for (int i=0; i<N; i++){
   a[i] = 1;
   b[i] = 1;
}
```

Host code

```
#pragma omp target enter data map(to: a[0:N])
   #pragma omp target
   {
    …
   }
```

Device code

```
   #pragma omp update from (c[0:N])
      for (int i=0; i<N; i++){
           std::cout << c[i] << std::endl;
      }
```

Host code

```
    #pragma omp target
    {
      …
    }
#pragma 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

# 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]
- Workharing
  - ★ **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]…]

# Mixing of OpenMP* and SYCL

# OpenMP* and SYCL DOs and DON'Ts

- USE openMP and SYCL constructs:
  - ✓ in separate files, in the same file, or in the same function with some restrictions
  - ✓ in executable files, in static libraries, in dynamic libraries, or in various combinations.
  - ✓ in a single application but in **different** parts (i.e., functions) of the code

✓ Warning! Oversubscription!

- ✓ using both OpenMP and SYCL a CPU

# OpenMP* and SYCL DOs and DON'Ts

- Restrictions:
  - ❖ OpenMP directives cannot be used inside DPC++/SYCL GPU kernels
  - ❖ DPC++/SYCL code cannot be used inside the OpenMP target regions.
    - ✓ ! it is possible to use SYCL constructs within the OpenMP code that runs on **the host CPU.**
  - ❖ OpenMP and DPC++/SYCL device parts of the program cannot have cross dependencies.
    - ❖ a function defined in the SYCL kernel cannot be called from the OpenMP offloading segment code and vice versa.
  - ❖ The direct interaction between OpenMP and SYCL runtime libraries is not supported at this time.
    - ❖ a device memory object created by OpenMP API is not accessible by DPC++ code

# Demo

## oneAPI Available on

# Intel® DevCloud

A development sandbox to develop, test and run workloads across a range of Intel CPUs, GPUs, and FPGAs using Intel's oneAPI software.

## Get Up & Running In Seconds!

software.intel.com/devcloud/oneapi

intel. DevCloud

- 1 Minute to Code
- No Hardware Acquisition
- No Download, Install or Configuration
- Easy Access to Samples & Tutorials
- Support for Jupyter Notebooks, Visual Studio Code

**Intel® Iris® Xe MAX Graphics cards available now**

devcloud.intel.com/oneapi/documentation/job-submission/#targeting-specific-compute-nodes

# C++ Code Sample

```cpp
#include <iostream>
int main(){
 int N = 100;
 float a[N], b[N], c[N];
 for (int i=0; i<N; i++){
     a[i] = 1; b[i] = 1;
 }

 #pragma omp target teams distribute parallel
for map(to: a, b) map(tofrom: c)
 {
     for (int i=0; i<N; i++){
       c[i] = a[i] + b[i];
     }
 }

 for (int i=0; i<10; i++){
     std::cout << c[i] << " ";
 }
 std::cout << std::endl;
 return 0;
}
```

```
$ icpx -qopenmp -fopenmp-targets=spir64 omp_cpp.cpp
$ ./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
Libomptarget --> Successfully loaded library
'libomptarget.rtl.level0.so'!
Target LEVEL0 RTL --> Looking for Level0 devices...
Target LEVEL0 RTL --> Initialized L0, API 10002
Target LEVEL0 RTL --> Found 1 driver(s)!
Target LEVEL0 RTL --> Found a GPU device, Name =
Intel(R) Iris(R) Plus Graphics 655 [0x3ea5]
1 devices!
…
```

# C++ Code Sample

```cpp
#include <iostream>
int main(){
 int N = 100;
 float a[N], b[N], c[N];
 for (int i=0; i<N; i++){
     a[i] = 1; b[i] = 1;
 }

 #pragma omp target teams distribute
parallel for map(to: a, b) map(tofrom: c)
 {
     for (int i=0; i<N; i++){
        c[i] = a[i] + b[i];
     }
 }

 for (int i=0; i<10; i++){
     std::cout << c[i] << " ";
 }
 std::cout << std::endl;
 return 0;
}
```

```
$ 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)[400]
Libomptarget device 0 info: to(unknown)[400]
Libomptarget device 0 info: to(unknown)[400]
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: alloc(unknown)[32]
Libomptarget device 0 info: Creating new map entry
with HstPtrBegin=0x00007ffe70620c00,
TgtPtrBegin=0x00000000023c5000, Size=400,
DynRefCount=1, HoldRefCount=0, Name=unknown
Libomptarget device 0 info: Copying data from host
to device, HstPtr=0x00007ffe70620c00,
TgtPtr=0x00000000023c5000, Size=400, Name=unknown
Libomptarget device 0 info: Creating new map entry
with HstPtrBegin=0x00007ffe70620a70,
TgtPtrBegin=0x00000000023c5200, Size=400,
DynRefCount=1, HoldRefCount=0, Name=unknown
```

# Mixing openMP* and SYCL Code Sample

```
float computePi(unsigned N) {
 float Pi;
 #pragma omp target map(from : Pi)
 #pragma omp parallel for reduction(+ : Pi)
 for (unsigned I = 0; I < N; ++I) {
         float T = (I + 0.5f) / N;
         Pi += 4.0f / (1.0 + T * T);
 }
 return Pi / N;
}
void iota(float *A, unsigned N) {
  cl::sycl::range<1> R(N);
  cl::sycl::buffer<float, 1> AB(A, R);
  cl::sycl::queue().submit([&](cl::sycl::handler &cgh) {
      auto AA = AB.template get_access<cl::sycl::access::mode::write>(cgh);
      cgh.parallel_for<class Iota>(R, [=](cl::sycl::id<1> I) {
         AA[I] = I;
      });
   });
}

#pragma omp parallel sections {
   #pragma omp section
      iota(Vec.data(), Vec.size());
   #pragma omp section
      Pi = computePi(8192u);
}
```

openMP

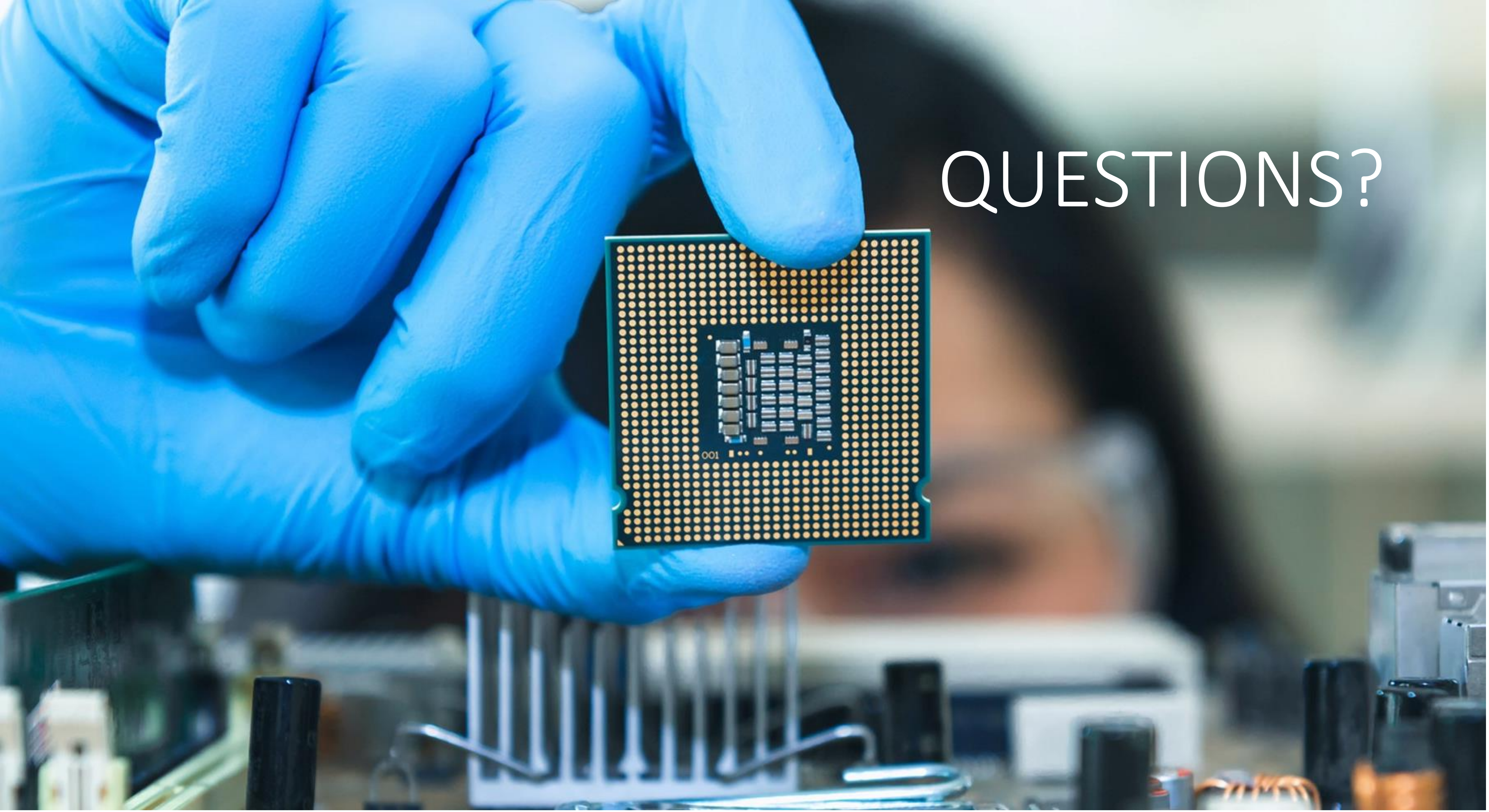
SYCL

CALL

```
$ icpx -fsycl -fiopenmp -fopenmp-targets=spir64 omp_sycl.cpp
$ ./a.out
Vec[512] = 512
Pi = 3.14159
```

# What else?

- OpenMP* Offload Basics in DevCloud (with lab!)
- openMP Specification
- C/C++ OpenMP* and SYCL* Composability
- Fortran Language and OpenMP* Features Implemented in Intel® Fortran Compiler
- OpenMP* Features and Extensions Supported in Intel® oneAPI DPC++/C++ Compiler


QUESTIONS?

# Fortran Code Sample

```
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

```
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
```

# 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.