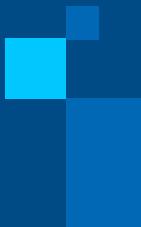


Transition to the Intel® Fortran Compiler

Q2 2023



intel.[®]

Our Fortran Compilers 2023 and Beyond



Agenda

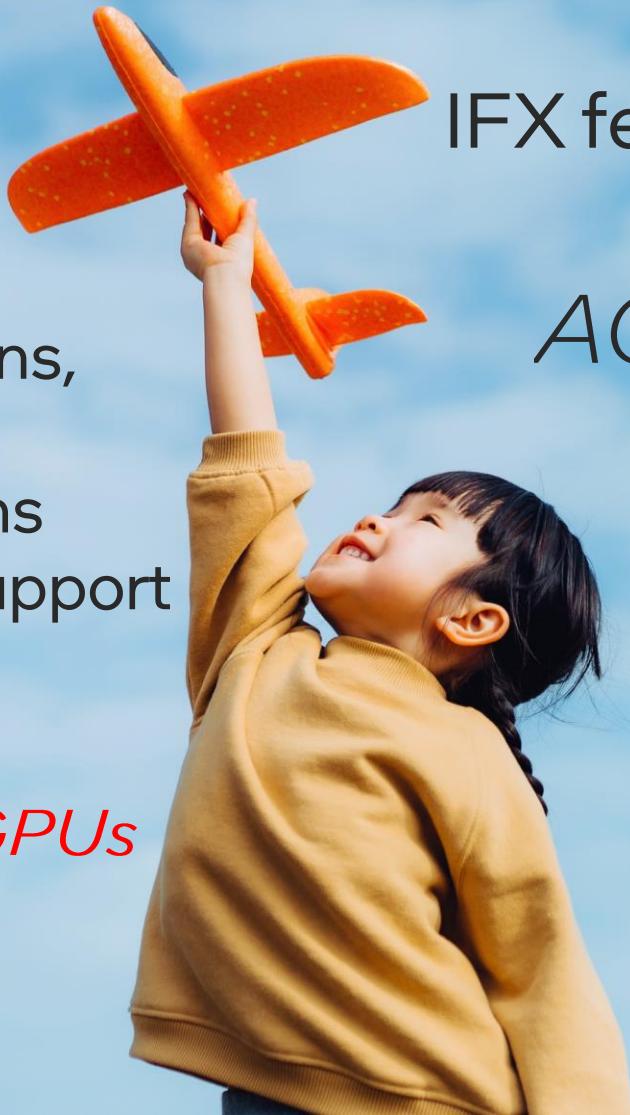
- Intel Fortran Compiler 2023 Overview
- Getting Started and Porting From IFORT to IFX
- IFX OpenMP Features
- Q&A

Celebrating IFX Feature Completion!

Full Fortran 2018
IFORT directives, options,
& behaviors
Legacy DEC extensions
Microsoft* Visual Studio support

AND MORE!

Acceleration with Intel GPUs



Our Goal:
IFX feature parity with IFORT
with 2023.0.0
ACCOMPLISHED!

Our Fortran Solution 2023

Intel® Fortran Compiler (ifx)

Our Fortran compiler tuned for

4th Gen Intel® Xeon® Scalable processors (code-named Sapphire Rapids), Intel® Xeon® CPU Max Series (code-named Sapphire Rapids HBM) and the Intel® Data Center GPU Max Series (code-named Ponte Vecchio)

Fortran Language Feature parity with IFORT
With Comparable Performance

Intel® Fortran Compiler Classic (ifort)

Dependable, proven features
and performance for pre-2023 Intel CPU products

Because you need advanced Fortran language features and the absolute best performance for your applications on Intel solutions

CHOICE! Continuity! Features! Performance!

IFX: Driving a New Era in Accelerated Computing

IFX: ALL that you like in IFORT *PLUS*

- OpenMP* 5.x Standards, offload to Intel GPUs from Fortran
 - An open, portable Standard maintains your investment*
 - Best in class OpenMP features and support*
- F18 DO CONCURRENT supports automatic offload to Intel GPUs

Protecting your Fortran Investment

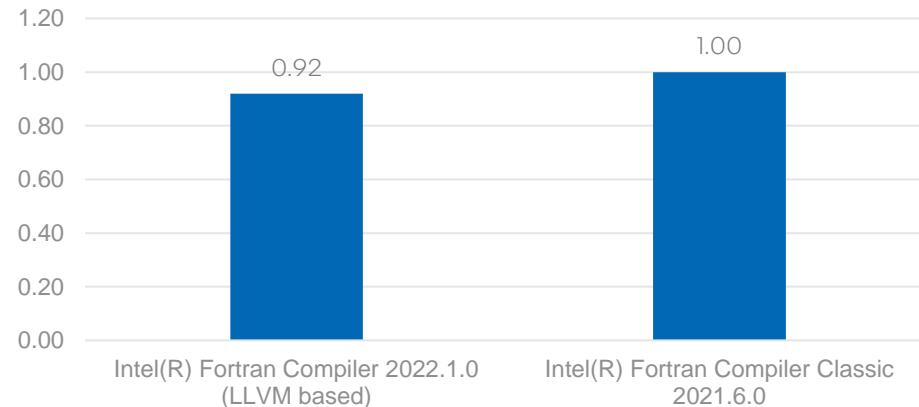
Same Fortran parser/analyzer you know and love from IFORT

- Supports legacy DEC extensions, *all F2018*, ifort directives and features
 - The majority of IFORT compiler directives and options you have used for years. And Microsoft Visual Studio* integration for Windows*
-
- *Binary compatible, mix and match ifx and ifort*

Intel® Fortran Compilers Build Time Performance on Linux*

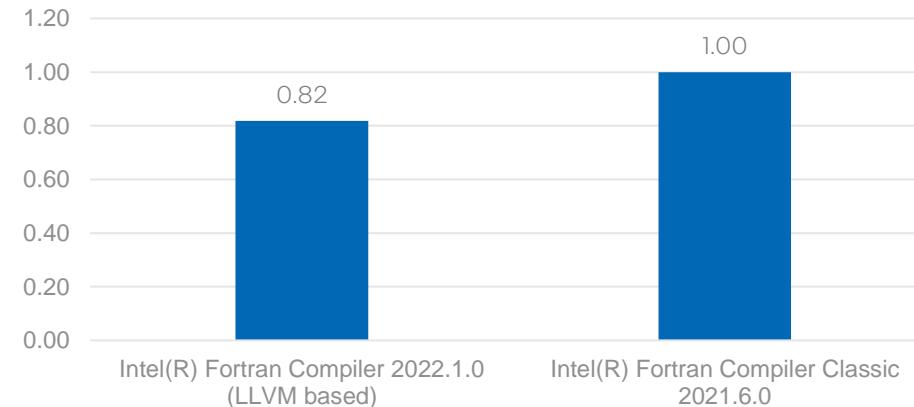
Build time performance advantage relative between Intel compilers on Intel® Core™ i7-8700K Processor

Build Time Performance for SPECrate 2017
Integer Suite @ 64 Bit (est.)
(Lower is Better)



Estimated: Built time measurement of the geometric mean of the Fortran workloads from the SPECrate* 2017 Integer suite

Build Time Performance for SPECrate 2017
Floating Point Suite @ 64 Bit (est.)
(Lower is Better)



Estimated: Built time measurement of the geometric mean of the Fortran workloads from the SPECrate* 2017 Floating Point suite

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. No product or component can be absolutely secure.

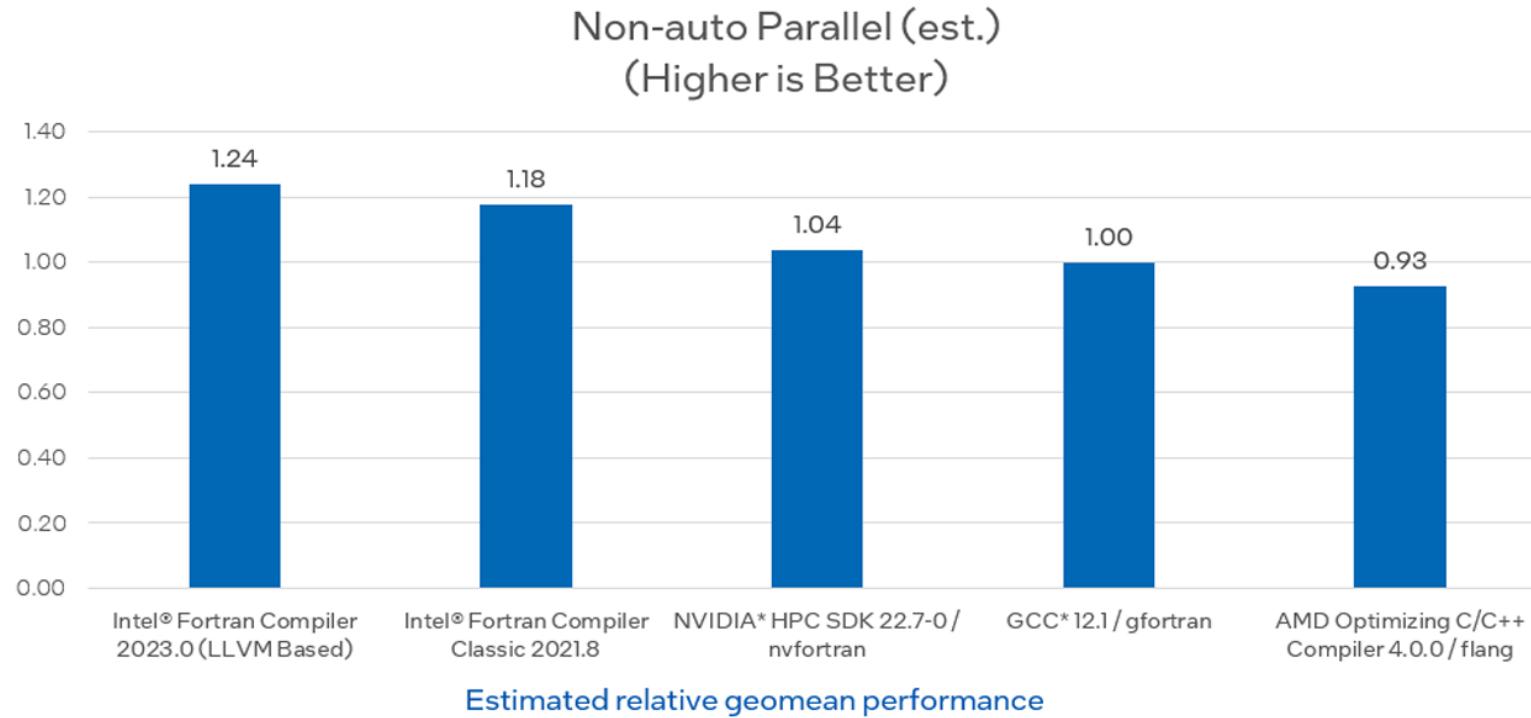
Your costs and results may vary. Intel technologies may require enabled hardware, software, or service activation.

More information on the SPEC benchmarks can be found at: <http://www.spec.org>

Configuration: Testing by Intel as of Mar 16, 2022. Configuration: Intel(R) Core(TM) i7-8700K CPU @ 3.70GHz, 16G x2 DDR4 2666. Red Hat Enterprise Linux release 8.0 (Ootpa), 4.18.0-80.el8.x86_64. Software: Intel(R) Fortran Compiler for applications running on Intel(R) 64, Version 2022.1.0 Build 20220316. Intel(R) Fortran Intel(R) 64 Compiler Classic for applications running on Intel(R) 64, Version 2021.6.0 Build 20220226_000000. Compiler switches: Intel(R) 64 Compiler Classic: ifort -O2 -xCORE-AVX512, Intel(R) Fortran Compiler: ifx -O2 -xCORE-AVX512.

Intel® Fortran Compiler Boosts Application Performance on Linux*

Performance Advantage Measured by Polyhedron* Fortran Benchmark on Intel® Core™ i9-12900K Processor



Testing Date: Performance results are based on testing by Intel as of December 2, 2022 and may not reflect all publicly available security updates.

Configuration Details and Workload Setup: Intel® Core™ i9-12900K CPU @ 5.2GHz, i9-12900K, 16G x2 DDR5 4800. Software: Intel® Fortran Compiler for applications running on Intel® 64, Version 2023.0.0 Build 20221201, Intel® Fortran Compiler Classic for applications running on Intel 64, Version 2021.8.0 Build 20221119_000000, GCC 12.1.0 / gfortran, AMD® Optimizing C/C++ Compiler 4.0.0 / flang - AMD clang version 14.0.6 (CLANG: AOCC_4.0.0-Build#434 2022_10_28) (based on LLVM Mirror, Version 14.0.6), NVIDIA® HPC SDK 22.7.0 / nvfortran. Red Hat Enterprise® Linux release 8.4 (Ootpa), 4.18.0-372.9.1.el8.x86_64. Non-auto Parallel compiler switches: Intel® Fortran Compiler: -Ofast -xlderlake -fno -nostandard-realloc-lhs. Intel® Fortran Compiler Classic: -fast -xCORE-AVX2 -nostandard-realloc-lhs. GCC / gfortran: -Ofast -mfpmath=sse -fno -march=lderlake -funroll-loops. AMD® Optimizing C/C++ Compiler / flang: compile: -O3 -ffast-math -march=znver3 -fveclib=AMDLIB -fno -mlv -unroll-aggressive -mlv -unroll-threshold=500; link: -Wl,-mlv -Wl,-inline-recursion=4 -Wl,-mlv -Wl,-Lsr-in-nested-loop -Wl,-mlv -Wl,-enable-iv-split -fno -mlv -Wl,-region-vectorize -Wl,-mlv -Wl,-function-specialize -Wl,-mlv -Wl,-align-all -nofallthru-blocks=6 -Wl,-mlv -Wl,-reduce-array-computations=3 -O3 -ffast-math -march=znver3 -fveclib=AMDLIB -fno -lamdlib -flang -lamdlib -lm. NVIDIA HPC SDK / nvfortran: -fast -Mipa=fast,inline -Mallocate=O3 -Mfprelaxed -Mstack_arrays.

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. No product or component can be absolutely secure.

Performance varies by use, configuration, and other factors. Learn more at www.intel.com/PerformanceIndex. Your costs and results may vary.

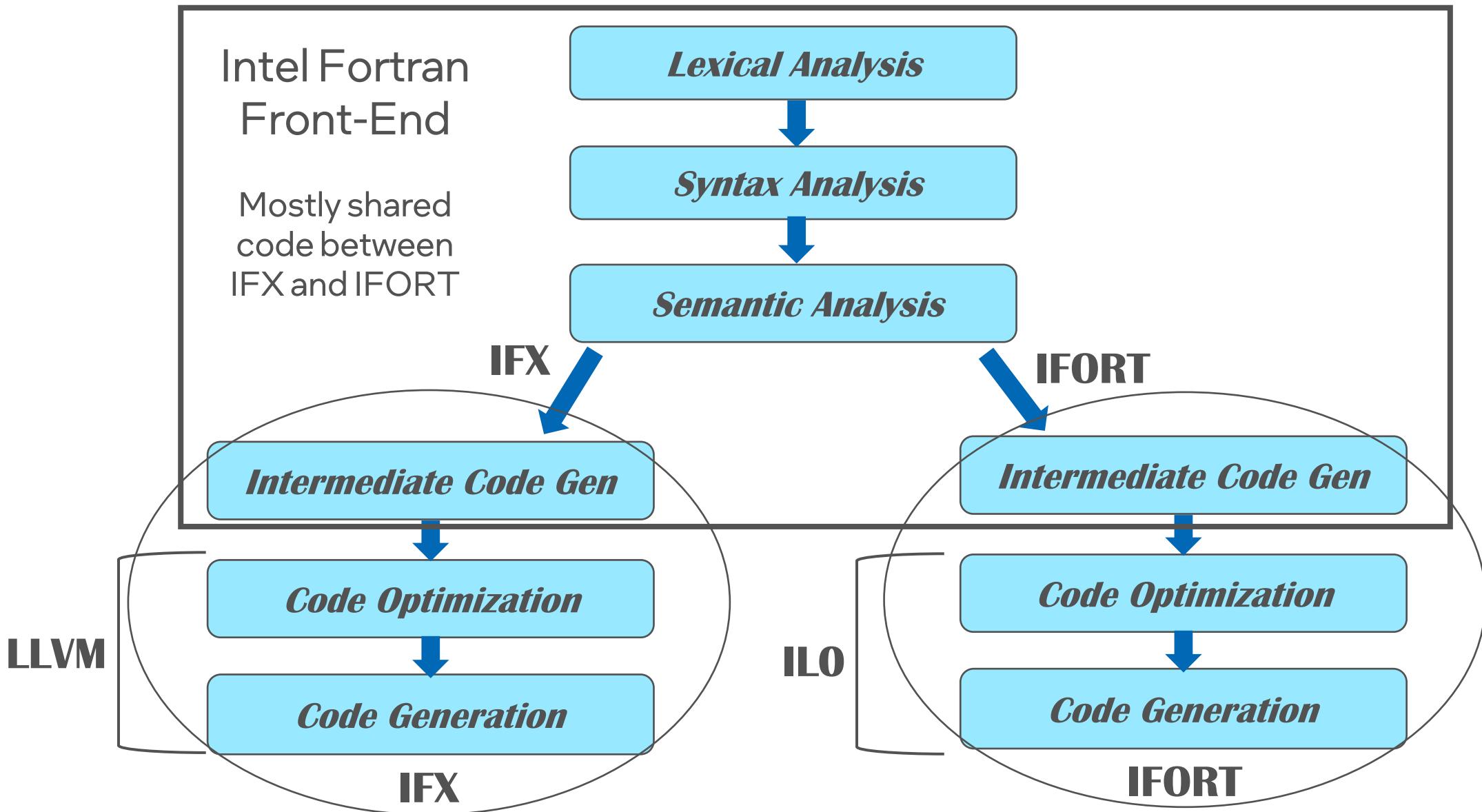
Creating
World
Changing
Technologies



Intel Fortran Compiler 2023

Compiler Options

How IFX Relates to IFORT



Important High-Level Understanding of IFX

Although both IFORT and IFX use the same language parser (Fortran Front End)
EVERYTHING after that is DIFFERENT

- Key takeaway
 - Optimization, vectorization, optimization reports, inlining, unrolling, interprocedural optimization, profile guided optimization, floating point control, code generation is different. SO ...
- Examine the compiler options you are using
 - Remove most of the “exotic” performance options
 - Start with a simple subset like `-O2 -xhost`

Be aware that the default, out of box optimization used by ifx is aggressive, just like ifort

Specify the following options with ifx to turn off default optimizations:

- On Linux: `-O0 -fno-fast-math`
- On Windows: `/O0 /fp:precise`

Compiler Options for IFX First Use

Consider these safe options.

Use these to validate IFX correctness with your application before moving to optimized code tests.

	ifx Linux and ifx Windows	
Disable optimization	-O0	/O0
Check for compile-time warnings	-warn all	/warn:all
Runtime checks	-check all	/check:all
Print stack traceback on crash	-g -traceback	/debug:full /traceback
Create symbols for debugging	-g	/debug:full
Turn off default OpenMP SIMD or !dir\$ SIMD	-qno-openmp-simd /Openmp-simd-	-no-simd /simd-
Obey Fortran semantics for expressions	-standard-semantics.	/standard-semantics
Use precise floating-point settings	-fp-model=precise	/fp:precise

Common Optimization Compiler Options

	Linux* ifx (ifort)
Disable optimization	-O0
Optimize for speed (no code size increase)	-O1
Optimize for speed (default)	-O2
High-level loop optimization	-O3
Create symbols for debugging	-g
Multi-file inter-procedural optimization	-ipo translates to -flio=full
Profile guided optimization (multi-step build)	-fprofile-generate (-prof-gen) -fprofile-use (-prof-use)
Optimize for speed across the entire program ("prototype switch")	-fast is same as "-ipo -O3 -static -fp-model fast" (-ipo -O3 -no-prec-div -static -fp-model fast=2 -xHost)
Recognize OpenMP directives	-qopenmp or -fopenmp (-qopenmp)

IFX Performance Options

Option	Description/Use
-Ofast	Sets compiler options: -O3 -no-prec-div -fp-model fast=2
-flto[=[full thin]]	Enables whole program link time optimization (LTO). -flto by itself sets flto=full. Thin see https://clang.llvm.org/docs/ThinLTO.html
-align arrayNbyte	N byte data alignment, use 32 on AVX2 or 64 on AVX-512
-ffast-math	Compatibility ICX macro option, same as IFX -fp-model=fast=2
-funroll-loops	Compatibility ICX option Unroll loops, same as IFX -unroll
-nostandard-realloc-lhs	Determines whether the compiler uses the current Fortran Standard rules or the old Fortran 2003 rules when interpreting assignment statements.

NOTE -nostandard-realloc-lhs should be used with caution. Assumes F2003 rules that require the LHS to be allocated AND with the correct shape to hold the RHS. If not, segfault or corrupted data. See the [Fortran Developer Guide and Reference](#) for more information

IFX Essentials: Options Support

- NEW: -qopenmp-simd is ON by default at -O1 and above
 - Turn off with -qno-openmp-simd or /Qopenmp-simd-
- IFORT options that are implemented are accepted quietly (no message)
- IFORT options that are not implemented generate a warning
 - ifx: command line warning #10148: option '-simd' not supported
- ifx -qnextgen-diag or ifx /Qnextgen-diag
 - Prints a long list of IFORT options TO BE supported
 - And prints a long list of IFORT options that are REMOVED

IFX -x and Intel Optimizations

- Classic compiler IFORT performs Intel-specific optimizations at -O2. Additional vectorization optimizations done if -x used.
- *LLVM compilers IFX will only do Intel specific optimizations with -x or -ax options*
 - Without -x or -ax you get default LLVM optimizations and vectorization
 - IMPLICATION: You ONLY get Intel optimizations and performance with IFX, if and only if, you use -x or -ax options.
 - -O2 is NOT enough with IFX
 - -xhost tunes for the computer where the compile is done
- **New!** ifx **-xsapphirerapids**

Notes on Intel® AVX-512 Processor Targeting

- `-x` and `-ax` SUGGESTION or REQUEST to the compiler, not imperative
- IFX sometimes will use AVX2 instead of AVX-512
- Use `-mprefer-vector-width=512`

```
ifx -mprefer-vector-width=512 ...
```

Same as IFORT compiler: `-qopt-zmm-usage=high`

Example:

```
ifx -xsapphirerapids -mprefer-vector-width=512
```

Optimization Report

- `-qopt-report [=n]` tells the compiler to generate an optimization report
 - ifx has three levels of reports. `n=3` is the max level
 - Includes Loop Optimizations, OpenMP parallelization and Register Allocation messages
- `-qopt-report-phase [=list]` specifies one or more optimizer phases for which optimization reports are generated.
 - loop: the phase for loop nest optimization
 - vec: the phase for vectorization
 - par: the phase for auto-parallelization
 - all: all optimizer phases
- `-qopt-report-filter=string` specifies the indicated parts of your application and generate optimization reports for those parts of your application.

Under
development

More Optimization Report

- `-qopt-report [=n]` also creates a YAML file
 - Contains additional optimization information from LLVM
 - Use `opt-viewer.py` (from `llvm/tools/opt-viewer`) to create html files from the YAML file
 - Information applies to host only

Interprocedural Optimization

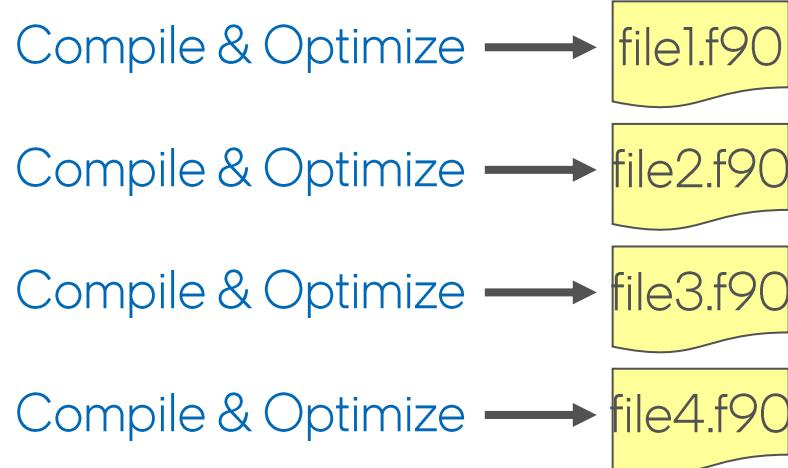
ifort

-ip	Only between modules of one source file
-ipo	Modules of multiple files/whole application

ifx

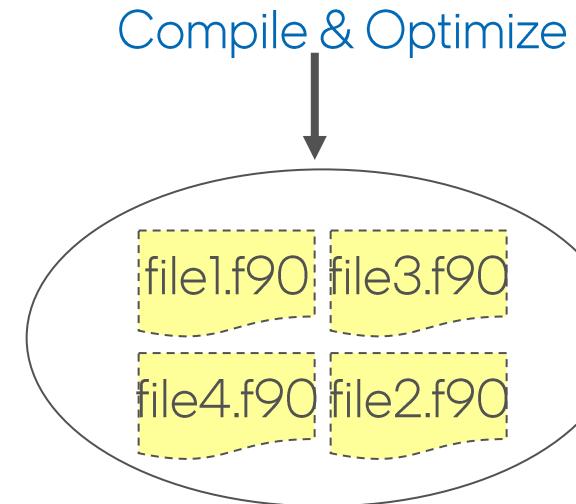
-ipo (mapped to -flto)	Link Time Optimization in IFX
------------------------	-------------------------------

Without IPO



Add -ipo to both compiling and linking steps

With IPO



Link Time Optimization (LTO): tinyurl.com/clang-lto

Other Good Things to Know

- Binaries and libraries generated with ifort can be linked with binaries and libraries built with ifx, and .mod files generated with one compiler can be used by the other (64-bit targets only).
 - EXCEPT when built with `-ipo`
- Profile Guided Optimization (PGO)
 - Two flavors
 - Sampling: `-fprofile-sample-generate` and `-fprofile-sample-use`
 - Instrumented: `-fprofile-instr-generate` and `-fprofile-instr-use`
 - More information
 - <https://clang.llvm.org/docs/UsersManual.html#profile-guided-optimization>

Known Issues

- COMPLEX data type performance
 - ifx 2023.x.x and older poor performance for COMPLEX data types
 - COMPLEX(KIND=4) same as Intel's COMPLEX*8
 - COMPLEX(KIND=8) same as Intel's COMPLEX*16 or DOUBLE COMPLEX
 - COMPLEX(KIND=16) same as Intel's COMPLEX*32
 - Improved performance in a future release

OpenMP* TARGET

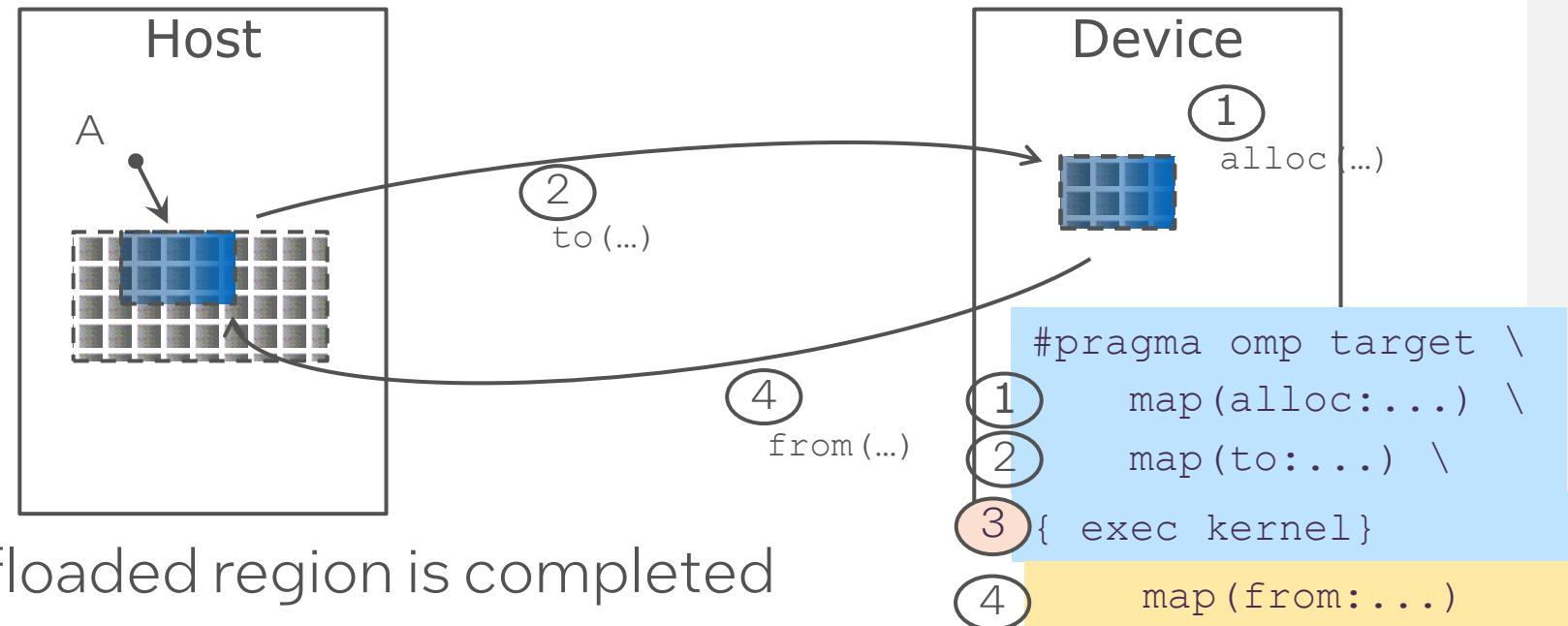
Status of IFX OpenMP Implementation, 2023.x

- All of OpenMP 4.5
- OpenMP 5.0/5.1
 - ~84% of 49 major features in 5.0/5.1
- Remaining features in progress
- OpenMP 5.2
 - In early stages
- OpenMP 6.0 – 2 features
 - INTEROP on DISPATCH
 - PREFER_TYPE in the APPEND_ARGS clause of DECLARE VARIANT

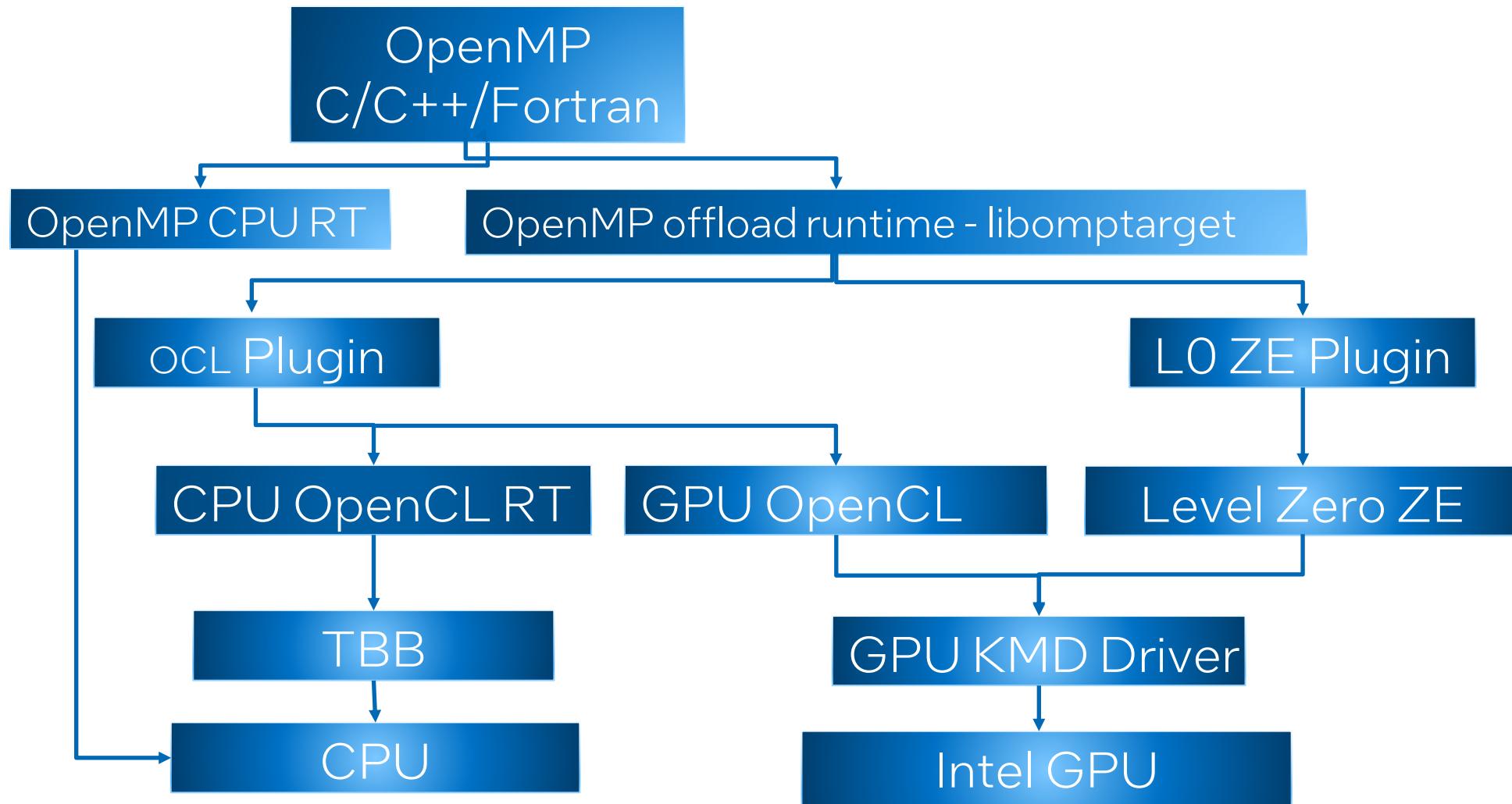
<https://www.intel.com/content/www/us/en/developer/articles/technical/fortran-language-and-openmp-features-in-ifx.html>

Offloading and Device Data Mapping

- Use *target* construct to
 - Transfer control from the host to target device
 - Map variables between the host and target device data environments
- Host thread waits until offloaded region is completed
 - Use other OpenMP tasks for asynchronous execution
- The **map** clauses determine how an *original variable* in a data environment is mapped to a *corresponding variable* in a device data environment
- OpenMP also provides Unified Shared Memory IF you want the data mapping to be automatic



High Level Architecture



Example: Simple Matrix Multiply Offload

Transfer control **and data** from the host to the device

Syntax

```
!$omp target [clause[,...] clause]  
structured-block
```

Clauses for TARGET

teams, distribute,
device(*scalar-integer-expression*),
map({alloc | to | from | tofrom}:list),
if(*scalar-expr*)

These OMP pragmas cause the loop to execute on a target device (i.e., GPU)

```
program matrix_multiply  
use omp_lib  
implicit none  
integer, parameter :: N=1000  
integer :: i, j, k, my_thread_id  
real, allocatable, dimension(:,:) :: a, b, c, c_validate  
allocate( a(N,N), b(N,N), c(N,N), c_validate(N,N) )  
! Initialize the arrays A and B, set C to 0.0 (not shown)  
!.... offload data & compute matrix multiply on the GPU  
!.... send 'a' and 'b' but do not move them back (no change)  
!.... 'c' goes to GPU and brought back from GPU (changed)  
!$omp target map(to: a, b ) map(tofrom: c )  
!$omp parallel do  
  
do j=1,N  
    do i=1,N  
        do k=1,N  
            c(i,j) = c(i,j) + a(i,k) * b(k,j)  
        enddo  
    enddo  
enddo  
!$omp end parallel do  
!$omp end target
```

Auto Offload “DO CONCURRENT” to Intel GPUs

No need to change DO CONCURRENT. Simply add the following compiler option:

```
-fopenmp-target-do-concurrent
```

```
ifx -qopenmp -fopenmp-targets:spir64 \  
      -fopenmp-target-do-concurrent source.f90
```

```
subroutine add_vec (a, b, c, N)  
real, dimension(:) :: a,b,c  
integer :: N
```

```
do concurrent (i=1:N) shared ( a,b,c )  
    c(i)=a(i)+b(i)
```

```
end do
```

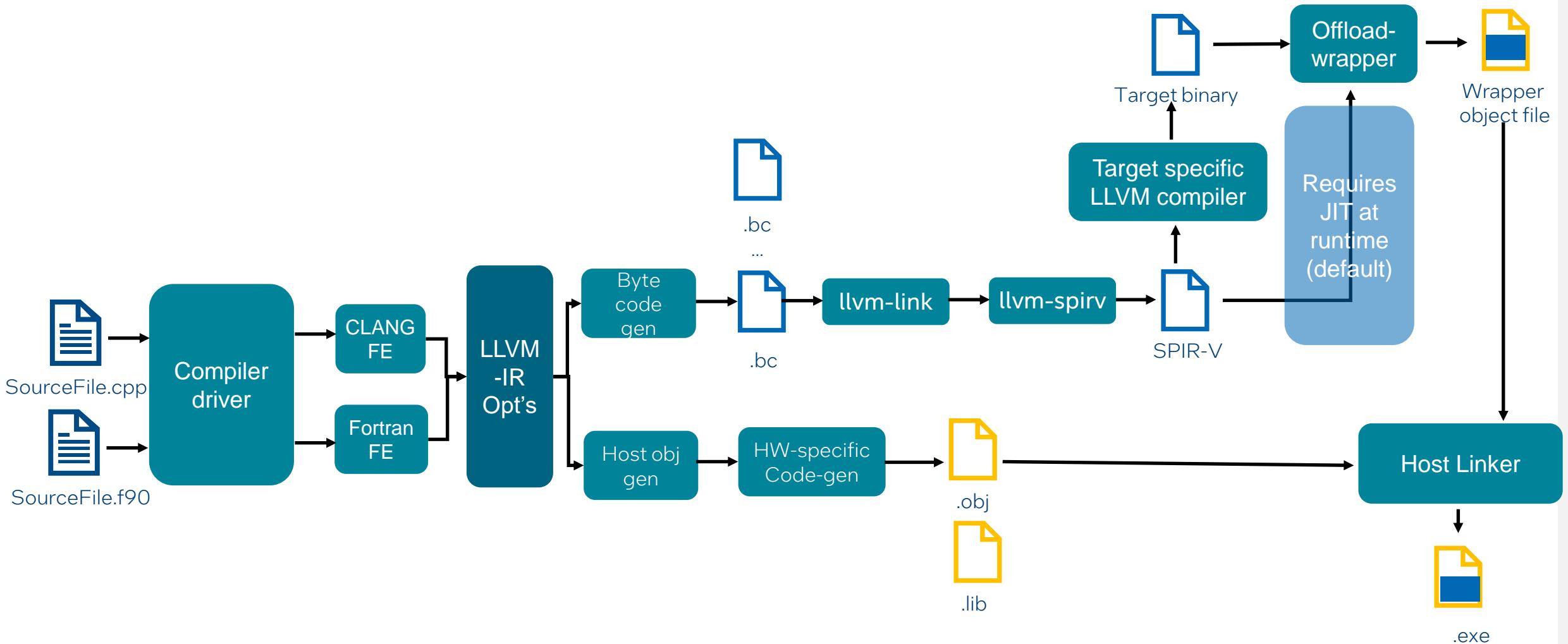
```
end subroutine add_vec
```

```
Compute  
on GPU
```

OpenMP* Compilation and Runtime 101

Offload Compilation Flow

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



IFX Essential: OpenMP TARGET Compiler Options

- `-qopenmp -fopenmp-targets:spir64`
 - OpenMP directives recognized
 - Fat binary produced for host and Intel GPU
 - spir64 stands for "64-bit Standard, Portable Intermediate Representation"

OpenMP* Offload Compiler Support

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

Passing options to device compiler

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

OpenMP TARGET Essential Environment Variables

Essential Environment Variables

- Select Target Device with Environment variable

`OMP_TARGET_OFFLOAD = mandatory | disabled | default`

- mandatory – The target region runs code on GPU or other accelerator
- disabled – The target region code runs on CPU
- default - The target region runs on GPU if device is available, else will fall back to the CPU

- Select Plugin/Driver

`LIBOMPTARGET_PLUGIN= [OPENCL | LEVEL0]`

`LIBOMPTARGET_DEVICETYPE= gpu | cpu` (only works for OpenCL)

- Dumps offloading runtime debugging information.

`LIBOMPTARGET_DEBUG= [1 | 2]`

`LIBOMPTARGET_INFO` (see LLVM Runtimes document URL below)

<https://openmp.llvm.org//design/Runtimes.html>

More Essential Environment Variables

- Profile GPU kernel start/complete time and data-transfer time.
 - `export LIBOMPTARGET_PLUGIN_PROFILE=T`

```
LIBOMPTARGET_PLUGIN_PROFILE(LEVEL0) for OMP DEVICE(0) Intel(R) Graphics [0x020a], Thread 0
```

Kernel 1 : __omp_offloading_3b_dd7d2220_MAIN__136											
Name	Host Time (msec)					Device Time (msec)					Count
	Total	Average	Min	Max	Total	Average	Min	Max	Count		
Compiling	: 2586.27	2586.27	2586.27	2586.27	0.00	0.00	0.00	0.00	1.00		
DataAlloc	: 5.93	0.42	0.00	1.29	0.00	0.00	0.00	0.00	14.00		
DataRead (Device to Host)	: 11.70	11.70	11.70	11.70	2.39	2.39	2.39	2.39	1.00		
DataWrite (Host to Device)	: 29.86	3.32	0.09	9.72	8.23	0.91	0.00	2.74	9.00		
Kernel 1	: 7660.45	7660.45	7660.45	7660.45	7638.77	7638.77	7638.77	7638.77	1.00		
Linking	: 0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00		
OffloadEntriesInit	: 807.01	807.01	807.01	807.01	0.00	0.00	0.00	0.00	1.00		

- Other LLVM OpenMP Runtime ENV vars are accepted.
 - <https://openmp.llvm.org//design/Runtimes.html>

Notice & 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 backup for configuration details. No product or component can be absolutely secure.
- 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.

The Intel logo is displayed in white against a solid blue background. The word "intel" is written in a lowercase, sans-serif font. A small, solid blue square is positioned above the letter "i". The letter "i" has a vertical stroke extending upwards from its top loop. The letter "t" has a vertical stroke extending downwards from its top loop. The letters "n", "e", and "l" are standard lowercase forms.

Backup Slides

More details on OpenMP Offload

Support for Compilers IFORT and IFX

- Summary: Same support model we have used for years
- Current version fully supported
- 2 previous versions supported, but only the last Update release to that version
- AND available but unsupported - next older version, last Update only, provided for download on Intel® Registration Center, but not supported
- This means ...
 - IFORT will continue to be supported per our usual model.
 - We will ensure you have Fortran compiler solutions that are Best-in-Class
- <https://www.intel.com/content/www/us/en/developer/articles/release-notes/intel-parallel-studio-xe-supported-and-unsupported-product-versions.html>

IFX OpenMP Features and Support



IFX Fortran Language &
OpenMP Features Support

[https://www.intel.com/content/www/us/en/developer/
articles/technical/fortran-language-and-openmp-
features-in-ifx.html](https://www.intel.com/content/www/us/en/developer/articles/technical/fortran-language-and-openmp-features-in-ifx.html)



Porting Guide, ifort to ifx

Kept up to date with tips and techniques to help you move from ifort to ifx

<https://www.intel.com/content/www/us/en/developer/articles/guide/porting-guide-for-ifort-to-ifx.html>



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

<https://www.intel.com/content/www/us/en/develop/documentation/fortran-compiler-oneapi-dev-guide-and-reference/top/compilation/supported-environment-variables.html>

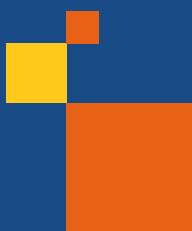


Parallelizing heterogenous applications with Intel® OpenMP and OpenMP offloading

Advanced Topics

Alina Shadrina

alina.shadrina@intel.com



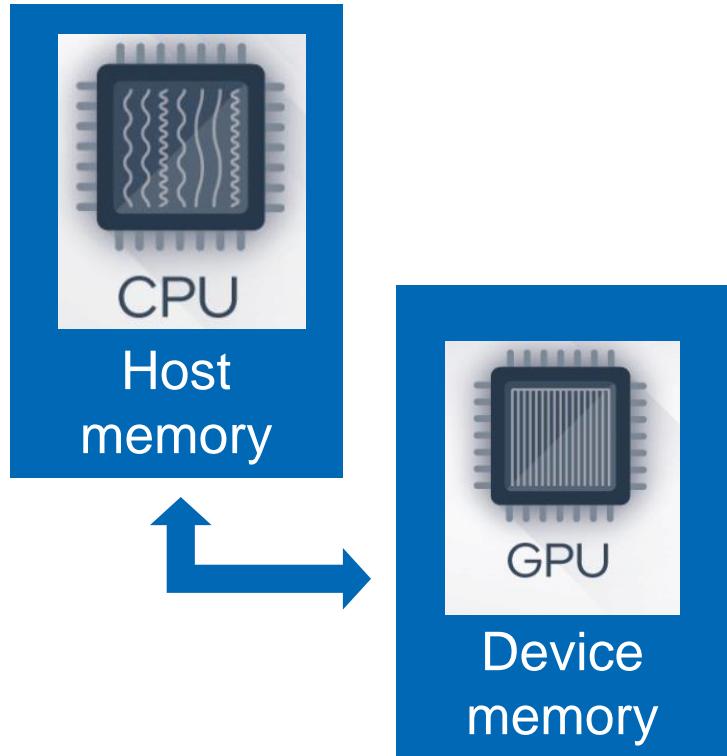
intel®

Agenda

- OpenMP* Offload Compiler Support
- Environment variables
- OpenMP* Target Construct
- Managing Device Data
- Demo

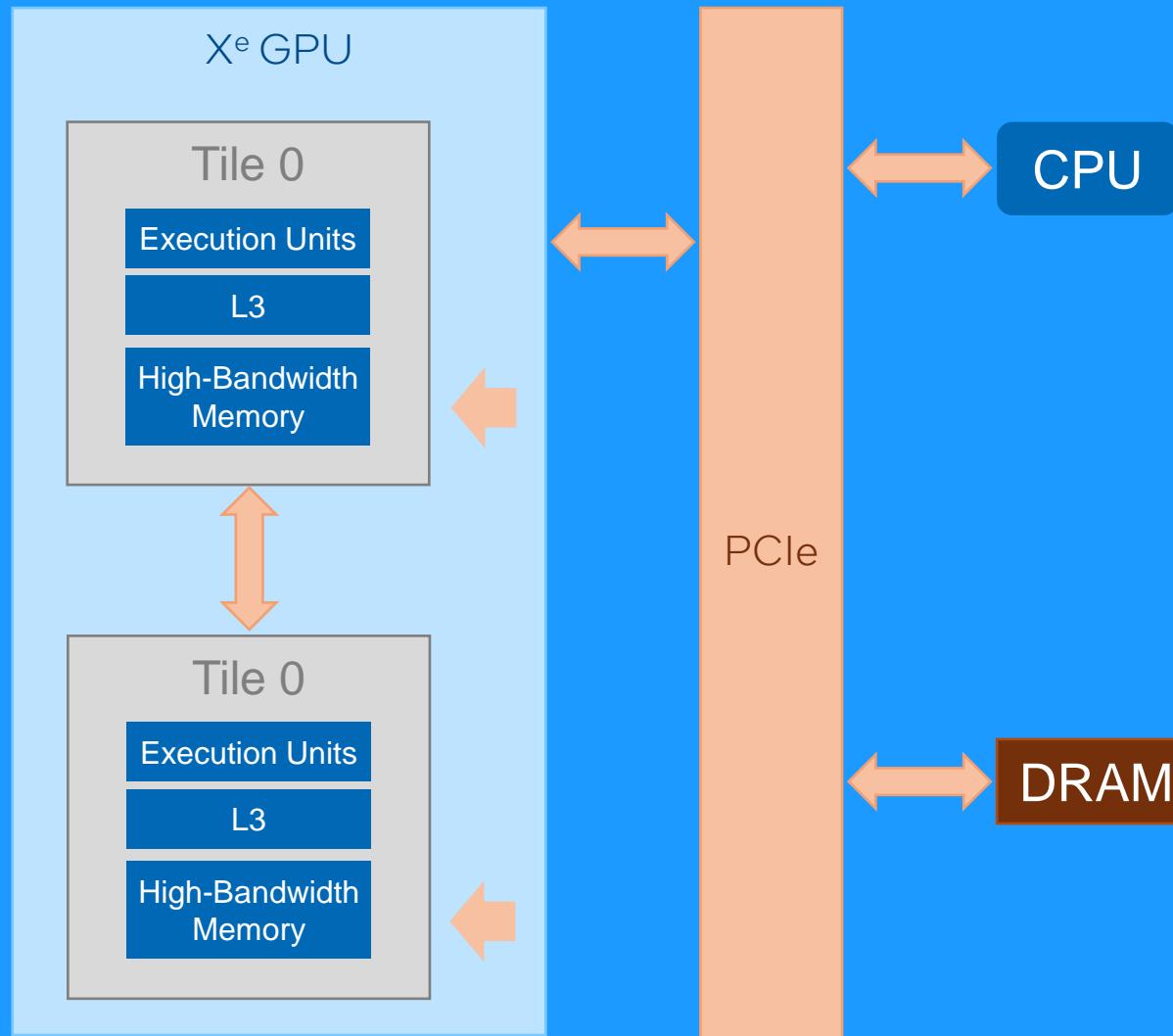
OpenMP* Offload Compiler Support

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

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

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

- Intel® Fortran Compiler

```
ifx -fopenmp -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).

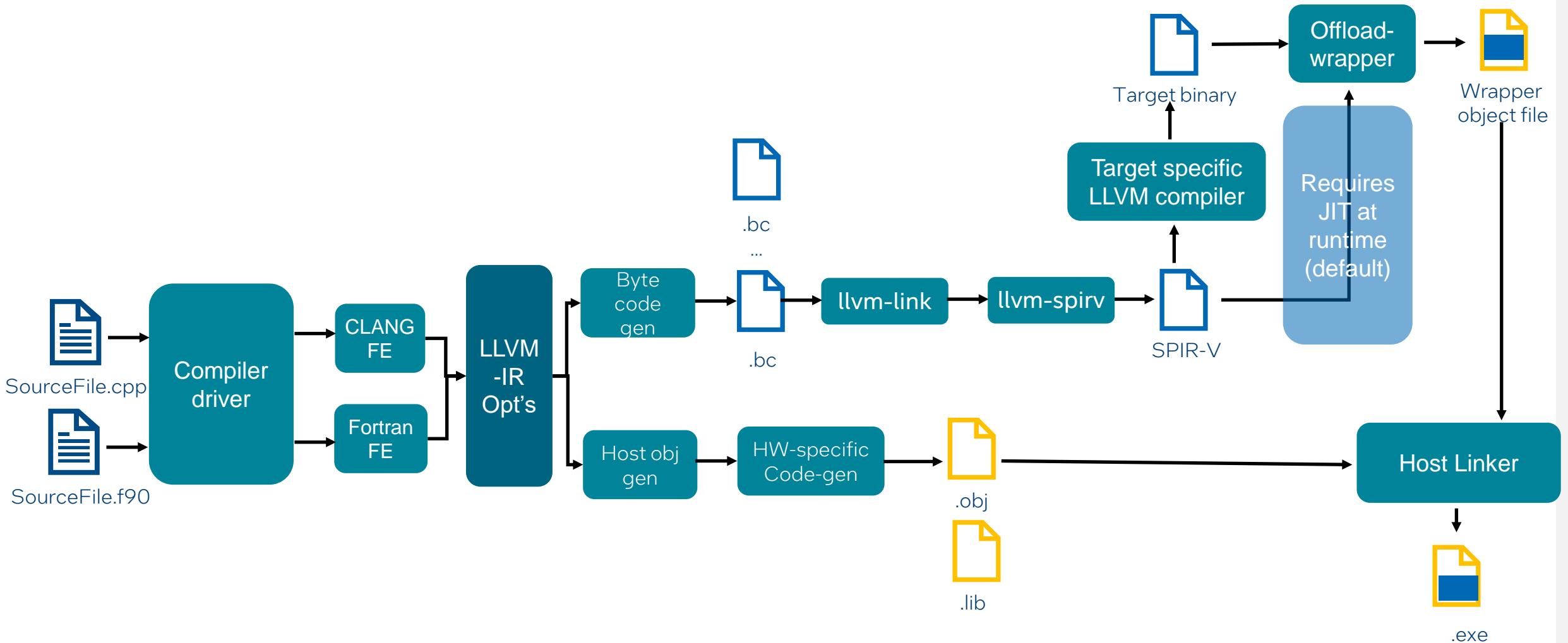
OpenMP* Offload Compiler Support

- Ahead-of-Time compilation supported

```
ifx -fopenmp -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"

Offload Compilation Flow



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 Level10
- `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

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

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**.

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

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
```

Device code

runs on tiles 2, 3, 4, and 5

```
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

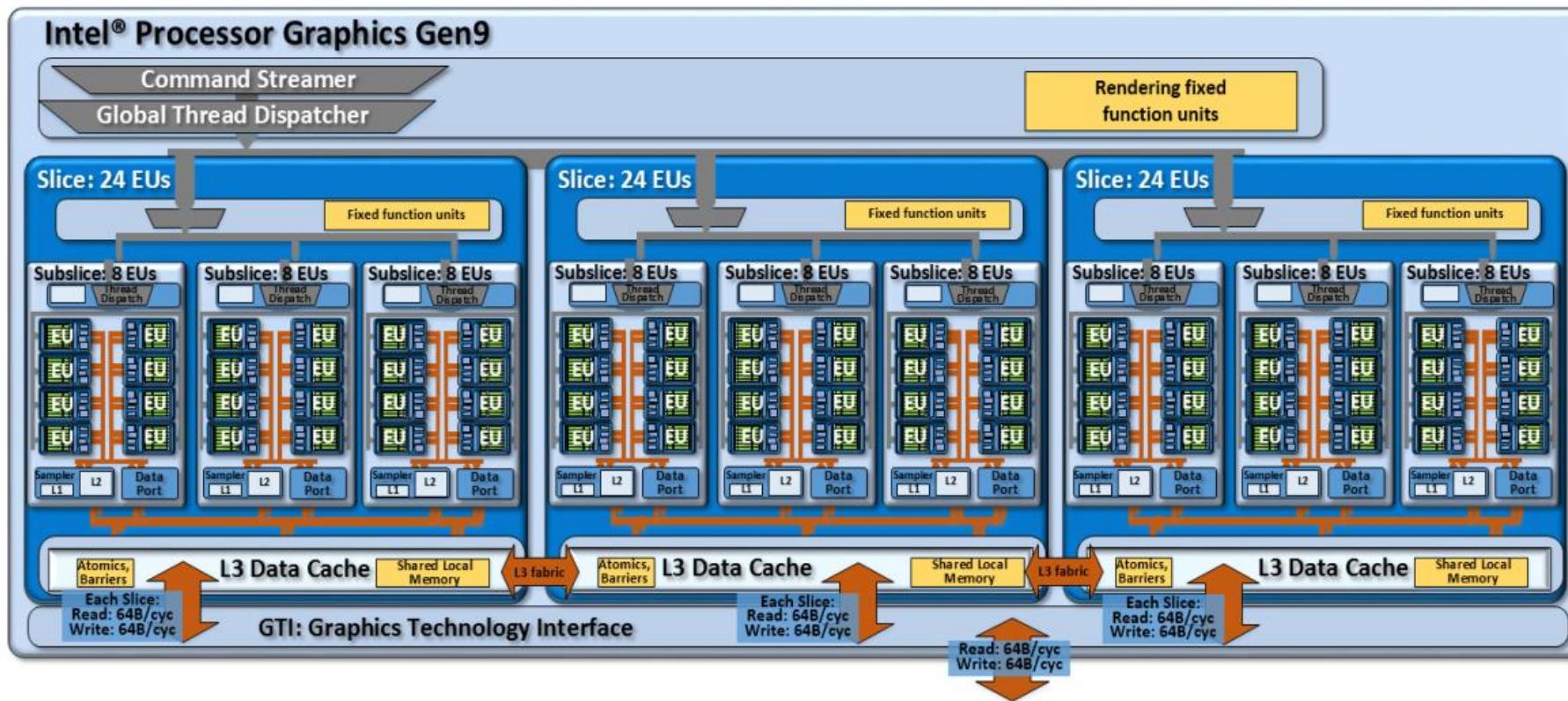


Figure 8: Another potential product design that instantiates the compute architecture of Intel® processor graphics gen9. This design is composed of three slices, of three subslices each for a total of 72 EUs.

[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

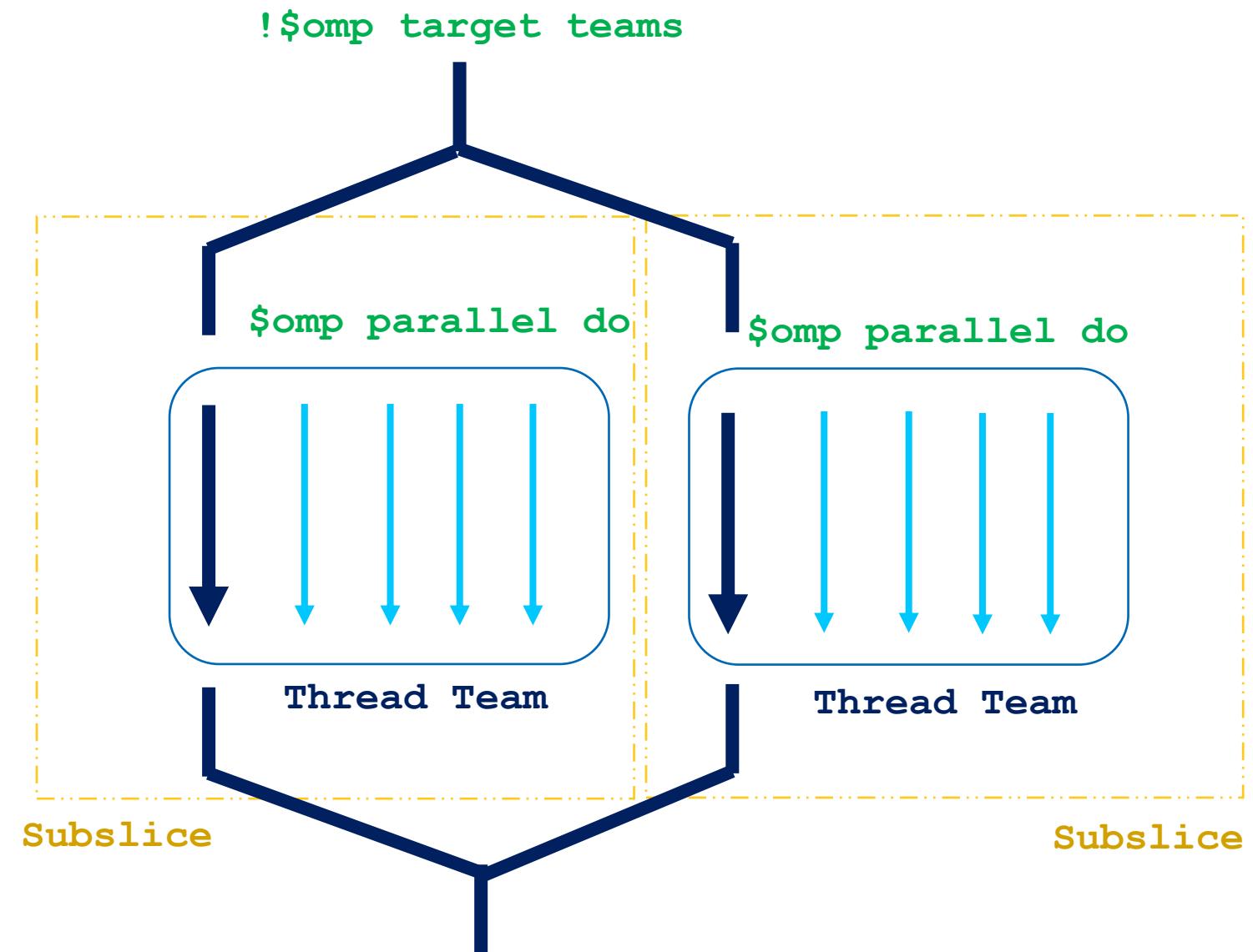
target teams

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

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

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

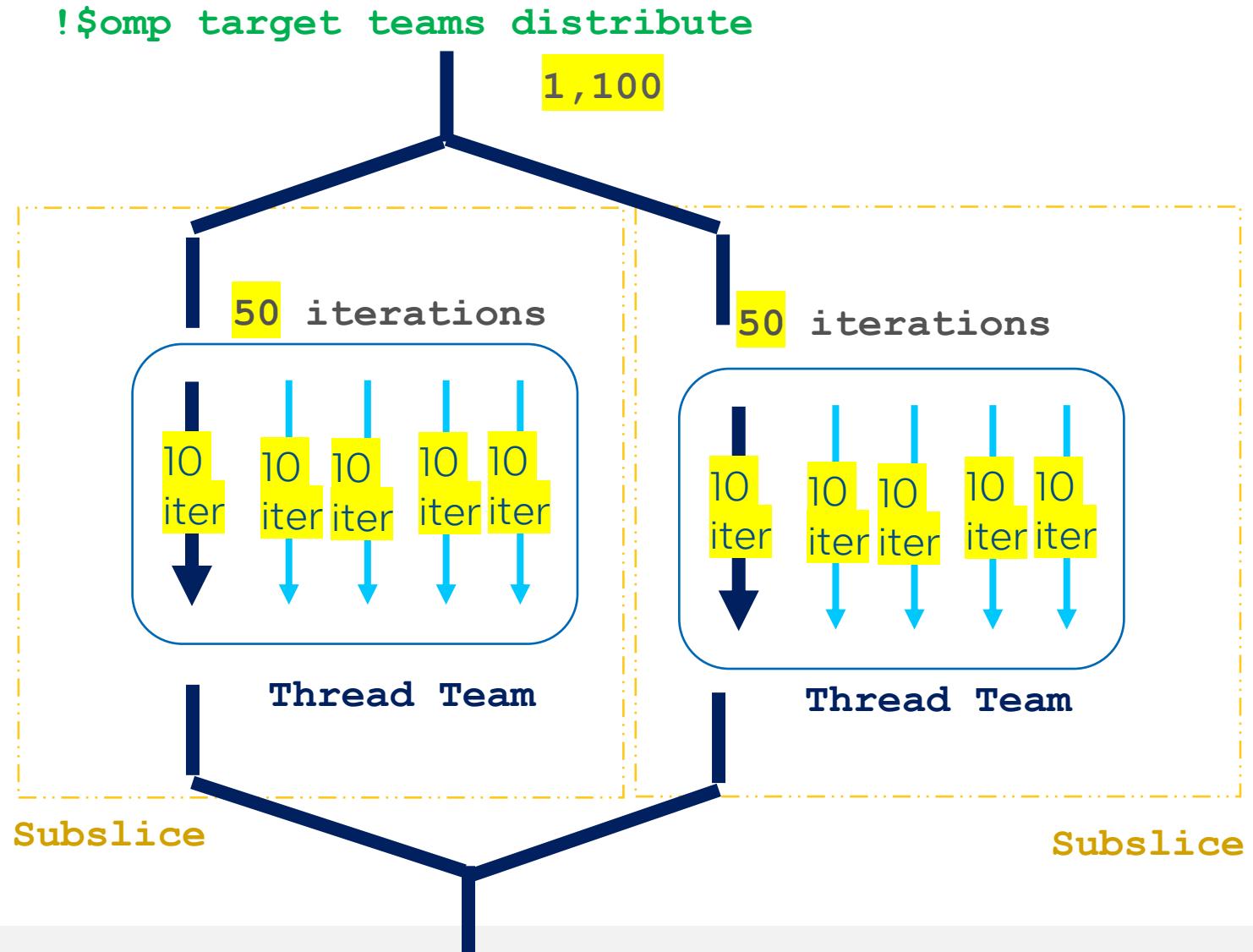
shortcut for specifying a target construct containing a teams distribute construct and no other statements.

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)

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
```

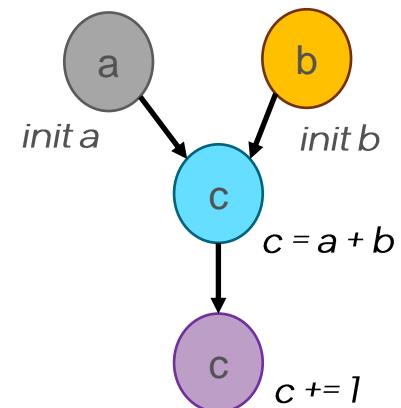
Host code

target [clause]

Offloads a code region to a target device

Synchronous by default

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



Managing Device Data

Fortran

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

Host code

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

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

Device
code

Data environment is destroyed,
data is transferred from device to
host

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

Host code

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
```

Host code

Device code

Host code

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

Demo

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 -fopenmp -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

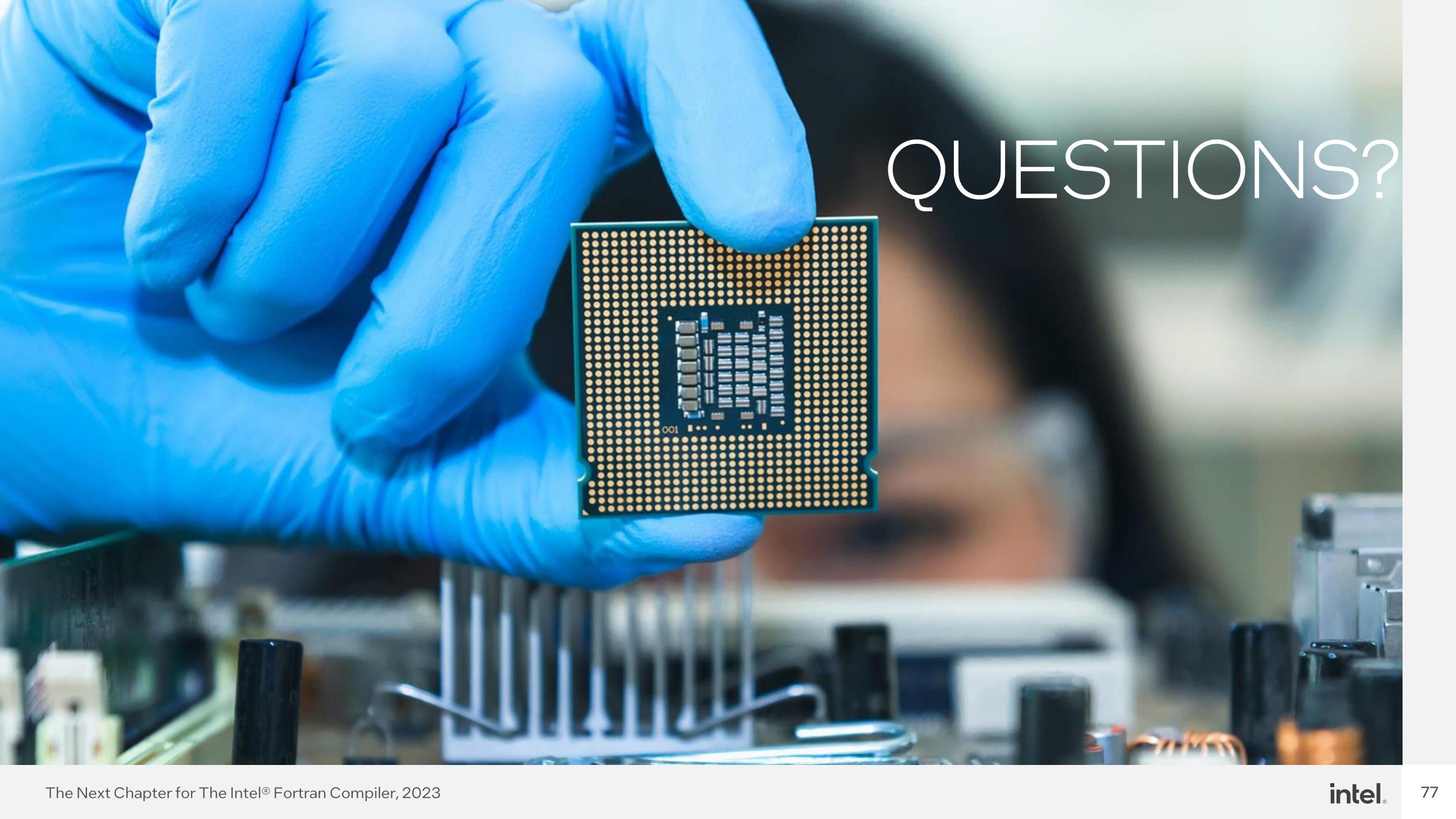
!$omp end target teams distribute parallel do

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: 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](#)



QUESTIONS?

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

```
#pragma omp target X
{
    cgh.parallel_for(A,
[=] (cl::sycl::id<1> I) {
        ...
    });
}
```

```
#pragma omp target X
{
    SyclKernelFunction();
}
```

```
cgh.parallel_for(A,
[=] (cl::sycl::id<1> I) {
    OpenMPTargetFunction();
});
```

```
cgh.parallel_for(A, X
[=] (cl::sycl::id<1> I) {
    #pragma omp target
    {
        ...
    }
});
```

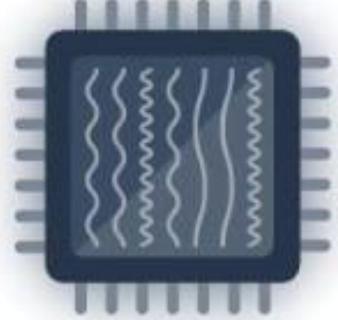
```
#pragma omp target map(to: A) X
{
    ...
}
cgh.parallel_for(A,
[=] (cl::sycl::id<1> I) {
});
```

- Restrictions:

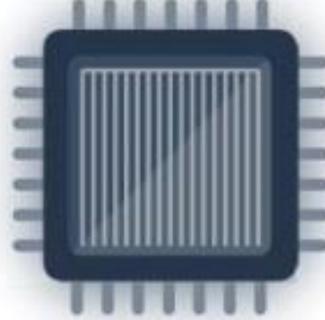
- ❖ OpenMP directives cannot be used inside DPC++/SYCL GPU kernels
- ❖ DPC++/SYCL code cannot be used inside the OpenMP target regions.
 - ✓ ! it is possible to use SYCL constructs within the OpenMP code that runs on the host CPU.
- ❖ OpenMP and DPC++/SYCL device parts of the program cannot have cross 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

Unified Shared Memory

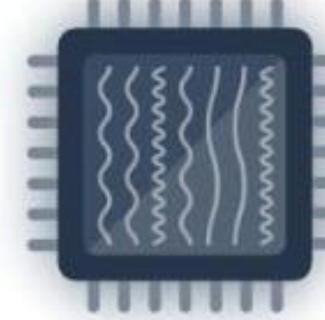
CPU



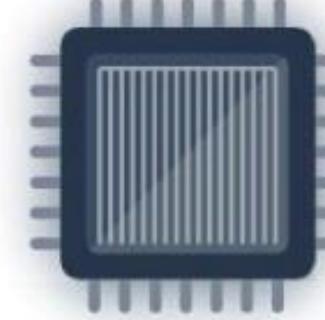
GPU



CPU



GPU



HOST MEMORY

GPU MEMORY

UNIFIED SHARED MEMORY

Managed memory allocators:

- `omp_target_alloc_host(...)`
- `omp_target_alloc_device(...)`
- `omp_target_alloc_shared(...)`

```
#pragma omp requires unified_shared_memory
float *a = (float *)omp_target_alloc(N, deviceID);
float *b = (float *)omp_target_alloc(N, deviceID);
float *c = (float *)omp_target_alloc(N, deviceID);
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

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

Host code

Clause	On which directive	Type of list item	Description
is_device_ptr	target, dispatch	C/C++: Pointer, array, or reference Fortran: C_PTR	Indicates that list item is a device pointer (has valid device address).
use_device_ptr	target data	C/C++: Pointer, array, or reference Fortran: C_PTR	Indicates that list item is a pointer to an object that has corresponding storage on device or is accessible on device.
has_device_addr	target	Any type (may be array section)	Indicates that list item has a valid device address.
use_device_addr	target data	Any type (may be array section)	Indicates that list item has corresponding storage on device or is accessible on the device.

is_device_ptr

Argument is device pointer

```
#pragma omp requires unified_shared_memory
float *a = (float *)omp_target_alloc_device(N *
                                             sizeof(int, deviceID));
float *b = (float *)omp_target_alloc(N, deviceID);
float *c = (float *)omp_target_alloc(N, deviceID);
for (int i=0; i<N; i++){
    a[i] = 1;
    b[i] = 1;
}
```

Host code

Use with:

- target
- Dispatch

```
#pragma omp target is_device_ptr(a) map(from: a[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

use_device_ptr

```
#pragma omp requires unified_shared_memory
float *a = (float *)omp_target_alloc_device(N *
                                             sizeof(int), deviceID);  
  
for (int i=0; i<N; i++) {  
    a[i] = 1;  
    b[i] = 1;  
}
```

Host code

```
#pragma omp target data use_device_ptr(a)  
{  
    #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

Argument is a pointer to an object that has corresponding storage on the device or is accessible on the device.

Use with:

- Target data

has_device_ptr

```
#pragma omp requires unified_shared_memory
float *a = (float *)omp_target_alloc_device(N *
                                             sizeof(int, deviceID));
for (int i=0; i<N; i++) {
    a[i] = 1;
    b[i] = 1;
}

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

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

The diagram illustrates the flow of data between host and device code. It consists of three rounded rectangular boxes. The top box, outlined in blue, contains the first snippet of code. The middle box, outlined in green, contains the second snippet. The bottom box, outlined in blue, contains the third snippet. Arrows point from the 'Host code' sections of the top and bottom boxes down towards the 'Device code' section of the middle box, indicating that the host code transfers data to the device code.

Argument already has valid device addresses, and therefore may be directly accessed from the device.

Use with:

- Target data

has_device_addr

```
#pragma omp requires unified_shared_memory
float *a = (float *)omp_target_alloc_device(N *
                                             sizeof(int, deviceID));
```

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

Host code

```
#pragma omp target data map(alloc:a)
#pragma omp target data use_device_addr(a)
#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

indicates that each list item already has corresponding storage on the device or is accessible on the device

Use with:

- Target data

C++ Code Sample

```
#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 -fopenmp -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

```
#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: 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);  
}
```



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



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.

© 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.