Transition to the Intel® Fortran Compiler

Q2 2023
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
Celebrate IFX Feature Completion!

Our Goal: IFX feature parity with IFORT with 2023.0.0

ACCOMPLISHED!

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

AND MORE!

Acceleration with Intel GPUs
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

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

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](http://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](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 (Oটopiа) 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

Non-auto Parallel (est.)
(Higher is Better)

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


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.
How IFX Relates to IFORT

Intel Fortran Front-End

Mostly shared code between IFX and IFORT

Lexical Analysis

Syntax Analysis

Semantic Analysis

Intermediate Code Gen

Code Optimization

Code Generation

IFX

IFORT

LLVM

ILO

The Next Chapter for The Intel® Fortran Compiler, 2023
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.

<table>
<thead>
<tr>
<th>Option</th>
<th>ifx Linux and ifx Windows</th>
</tr>
</thead>
<tbody>
<tr>
<td>Disable optimization</td>
<td><code>-O0</code> /<code>O0</code></td>
</tr>
<tr>
<td>Check for compile-time warnings</td>
<td><code>-warn all</code> /<code>warn:all</code></td>
</tr>
<tr>
<td>Runtime checks</td>
<td><code>-check all</code> /<code>check:all</code></td>
</tr>
<tr>
<td>Print stack traceback on crash</td>
<td><code>-g -traceback</code> /<code>debug:full /traceback</code></td>
</tr>
<tr>
<td>Create symbols for debugging</td>
<td><code>-g</code> /<code>debug:full</code></td>
</tr>
<tr>
<td>Turn off default OpenMP SIMD or !dir$ simd</td>
<td><code>-qno-openmp-simd</code> /<code>no-simd</code></td>
</tr>
<tr>
<td></td>
<td><code>/Qopenmp-simd-</code> /<code>simd-</code></td>
</tr>
<tr>
<td>Obey Fortran semantics for expressions</td>
<td><code>-standard-semantics</code> /<code>standard-semantics</code></td>
</tr>
<tr>
<td>Use precise floating-point settings</td>
<td><code>-fp-model=precise</code> /<code>fp:precise</code></td>
</tr>
</tbody>
</table>
## Common Optimization Compiler Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Linux* ifx (ifort)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Disable optimization</td>
<td>-O0</td>
</tr>
<tr>
<td>Optimize for speed (no code size increase)</td>
<td>-O1</td>
</tr>
<tr>
<td>Optimize for speed (default)</td>
<td>-O2</td>
</tr>
<tr>
<td>High-level loop optimization</td>
<td>-O3</td>
</tr>
<tr>
<td>Create symbols for debugging</td>
<td>-g</td>
</tr>
<tr>
<td>Multi-file inter-procedural optimization</td>
<td>-ipo translates to -flto=full</td>
</tr>
<tr>
<td>Profile guided optimization (multi-step build)</td>
<td>-fprofile-generate (-prof-gen)</td>
</tr>
<tr>
<td></td>
<td>-fprofile-use (-prof-use)</td>
</tr>
<tr>
<td>Optimize for speed across the entire program (&quot;prototype switch&quot;)</td>
<td>-fast  is same as &quot;-ipo -O3 -static -fp-model fast&quot; (-ipo -O3 -no-prec-div -static -fp-model fast=2 -xHost)</td>
</tr>
<tr>
<td>Recognize OpenMP directives</td>
<td>-qopenmp or -fiopenmp (-qopenmp)</td>
</tr>
</tbody>
</table>
## IFX Performance Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description/Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>-Ofast</td>
<td>Sets compiler options: -O3  -no-prec-div  -fp-model fast=2</td>
</tr>
<tr>
<td>-flto[=[ full</td>
<td>thin ]]</td>
</tr>
<tr>
<td>-align arrayNbyte</td>
<td>N byte data alignment, use 32 on AVX2 or 64 on AVX-512</td>
</tr>
<tr>
<td>-ffast-math</td>
<td>Compatibility ICX macro option, same as IFX -fp-model=fast=2</td>
</tr>
<tr>
<td>-funroll-loops</td>
<td>Compatibility ICX option Unroll loops, same as IFX -unroll</td>
</tr>
<tr>
<td>-nostandard-realloc-lhs</td>
<td>Determines whether the compiler uses the current Fortran Standard rules or the old Fortran 2003 rules when interpreting assignment statements.</td>
</tr>
</tbody>
</table>

**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](https://software.intel.com/content/www/us/en/develop/simplified-fortran-manuals.html) 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.
More Optimization Report

- `--qopt-report` 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

<table>
<thead>
<tr>
<th>ifort</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>-ip</td>
<td>Only between modules of one source file</td>
</tr>
<tr>
<td>-ipo</td>
<td>Modules of multiple files/whole application</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ifx</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>-ipo (mapped to -flto)</td>
<td>Link Time Optimization in IFX</td>
</tr>
</tbody>
</table>

## Without IPO

- Compile & Optimize → file1.f90
- Compile & Optimize → file2.f90
- Compile & Optimize → file3.f90
- Compile & Optimize → file4.f90

Add -ipo to both compiling and linking steps

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

## With IPO

- Compile & Optimize

- file1.f90
- file2.f90
- file3.f90
- file4.f90
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
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

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

OpenMP C/C++/Fortran

OpenMP CPU RT

OpenMP offload runtime - libomptarget

OCL Plugin

CPU OpenCL RT

GPU OpenCL

TBB

CPU

LO ZE Plugin

Level Zero ZE

GPU KMD Driver

Intel GPU

GPU OpenCL RT

CPU OpenCL RT

CPU
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)

```fortran
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:

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

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

```fortran
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

SourceFile.cpp → Compiler driver → LLVM -IR Opt's → Byte code gen → llvm-link → llvm-spirv → SPIR-V → Target specific LLVM compiler → Requires JIT at runtime (default) → Target binary → Offload-wrapper → Wrapper object file

SourceFile.f90 → Host obj gen → HW-specific Code-gen → .obj → .lib → Host Linker → .exe

-ffopenmp-targets=spir64
-ffopenmp-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

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

Passing options to device compiler

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

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

Kernel 1                  : __omp_offloading_3b_dd7d2220_MAIN___l36

<table>
<thead>
<tr>
<th>Name</th>
<th>Host Time (msec)</th>
<th>Device Time (msec)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Total</td>
<td>Average</td>
</tr>
<tr>
<td>Compiling</td>
<td>2586.27</td>
<td>2586.27</td>
</tr>
<tr>
<td>DataAlloc</td>
<td>5.93</td>
<td>0.42</td>
</tr>
<tr>
<td>DataRead (Device to Host)</td>
<td>11.70</td>
<td>11.70</td>
</tr>
<tr>
<td>DataWrite (Host to Device)</td>
<td>29.86</td>
<td>3.32</td>
</tr>
<tr>
<td>Kernel 1</td>
<td>7660.45</td>
<td>7660.45</td>
</tr>
<tr>
<td>Linking</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>OffloadEntriesInit</td>
<td>807.01</td>
<td>807.01</td>
</tr>
</tbody>
</table>
```

- Other LLVM OpenMP Runtime ENV vars are accepted.

  - [https://openmp.llvm.org//design/Runtimes.html](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.
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

IFX OpenMP Features and Support

IFX Fortran Language & OpenMP Features Support
Kept up to date with tips and techniques to help you move from ifort to ifx

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

Parallelizing heterogeneous applications with Intel® OpenMP and OpenMP offloading

Advanced Topics
Alina Shadrina
alina.shadrina@intel.com

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

```
ifx -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"`
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
OpenMP Offload Constructs

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

<table>
<thead>
<tr>
<th>C++</th>
<th>Fortran</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>#pragma omp target [clause[[],clause]...] structured-block</code></td>
<td><code>!$omp target [clause[[],clause]...] structured-block !$omp end target</code></td>
</tr>
<tr>
<td><code>#pragma omp target data [clause[[],clause]...] structured-block</code></td>
<td><code>!$omp target [clause[[],clause]...] structured-block !$omp end target data</code></td>
</tr>
<tr>
<td><code>#pragma omp teams [clause[[],clause]...] structured-block</code></td>
<td><code>!$omp teams [clause[[],clause]...] structured-block</code></td>
</tr>
<tr>
<td><code>#pragma omp distribute [clause[[],clause]...] structured-block</code></td>
<td><code>!$omp distribute [clause[[],clause]...] structured-block</code></td>
</tr>
</tbody>
</table>
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 message
  - **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

The `target` construct:

- 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**.

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

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

```fortran
do k=1,100
   write (*,*) c(k)
end do
```
Target Device Construct

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

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

2.12.5 `target Construct`

**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

**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

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

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

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

**Host code**

**Device code**

*runs on tiles 2, 3, 4, and 5*
OpenMP* Device Parallelism

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

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

**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.
**OpenMP* Device Parallelism**

### Host code

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

### Device code

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

### Host code

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

---

**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

<table>
<thead>
<tr>
<th>OpenMP</th>
<th>GPU Hardware</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIMD</td>
<td>SIMD Lane (Channel)</td>
</tr>
<tr>
<td>Thread</td>
<td>SIMD Thread mapped to an EU</td>
</tr>
<tr>
<td>Team</td>
<td>Group of threads mapped to a Subslice</td>
</tr>
<tr>
<td>League</td>
<td>Multiple Teams mapped to a GPU</td>
</tr>
</tbody>
</table>

```
!$omp target teams
```

```
$omp parallel do
```

Thread Team

Subslice

```
$omp parallel do
```

Subslice

The Next Chapter for The Intel® Fortran Compiler, 2023

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

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

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

**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

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

Diagram explanation:
- **Teams Distribute Construct**
- **Parallel Do Loop**: `do k=1,100`
  - `c(k) = a(k) + b(k)`
- **Thread Team**
  - 50 iterations
  - Subslice: 10 iter
- **Subslice**
  - 10 iter

The diagram illustrates the parallel execution of the do loop, with the loop iterations distributed among multiple threads and subslices.
Calling functions inside Target region

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

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

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

**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

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)

Device code

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

Host code

2.17.11 depend Clause
15.6 nowait Clause
Managing Device Data
Fortran
```fortran
integer :: a(100), b(100), c(100)
do k=1,100
    a(k) = 1
    b(k) = 1
end do

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

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

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

Data environment is destroyed, data is transferred from device to host.
2.19.7 Data-Mapping Attribute Rules, Clauses, and Directives

**target map (map_type)**

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

- **map_type**: `to`, `from`, `tofrom`, `alloc`, `release`, `delete`
- **modifier**: `always`, `close`, `<mapper identifier>`
Dynamically Allocated Data

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

Host code

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

Device code

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

```
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

2.19.7 Data-Mapping Attribute Rules, Clauses, and Directives
Minimize Copy Overhead

• What if we need \( a \) and \( b \) in multiple target regions?
• Data movement overhead
• Solution:
  • target enter data
  • target update

```fortran
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

Device code

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
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])

2.12.3 target enter data Construct

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

```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 ompFort.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'!
...
```
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=0x00007fc6f0441b0,
  TgtPtrBegin=0x000000000168b000, Size=400000, DynRefCount=1, HoldRefCount=0, Name=unknown
Libomptarget device 0 info: Copying data from host to device, HstPtr=0x00007fc6f0441b0,
  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

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
Managed memory allocators:

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

Host code:
```c
#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;
}
```

Device code:
```c
#pragma omp target ...
{
    for (int i=0; i<N; i++){
        c[i] = a[i] + b[i];
    }
}
```

Host code:
```c
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;
}
<table>
<thead>
<tr>
<th>Clause</th>
<th>On which directive</th>
<th>Type of list item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>is_device_ptr</td>
<td>target, dispatch</td>
<td>C/C++: Pointer, array, or reference Fortran: C_PTR</td>
<td>Indicates that list item is a device pointer (has valid device address).</td>
</tr>
<tr>
<td>use_device_ptr</td>
<td>target data</td>
<td>C/C++: Pointer, array, or reference Fortran: C_PTR</td>
<td>Indicates that list item is a pointer to an object that has corresponding storage on device or is accessible on device.</td>
</tr>
<tr>
<td>has_device_addr</td>
<td>target</td>
<td>Any type (may be array section)</td>
<td>Indicates that list item has a valid device address.</td>
</tr>
<tr>
<td>use_device_addr</td>
<td>target data</td>
<td>Any type (may be array section)</td>
<td>Indicates that list item has corresponding storage on device or is accessible on the device.</td>
</tr>
</tbody>
</table>
is_device_ptr

Argument is device pointer

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

```fortran
#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];
    }
}
```

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

```fortran
Host code
```

```fortran
Device code
```

```fortran
Host code
```

Use with:
- target
- Dispatch

Argument is device pointer

- Requires unified shared memory
- Requires device_ptr

Use cases:
- Symbolic pinning
- Bind a, b to device
- Call device intrinsics

Example:
- Allocate device arrays
- Compute on device
- Copy results to host

Notes:
- Requires explicit device pointer
- No data transfer with device_ptr
- Use with explicit device memory
- No unified shared memory

Device code

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

```fortran
Host code
```
use_device_ptr

```fortran
#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 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];
    }
}
```

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

**Use with:**
- Target data

Host code

Device code

Host code
has_device_ptr

```cpp
#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**

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

**Device code**

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

**Host code**

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

Use with:
- Target data

---

The Next Chapter for The Intel® Fortran Compiler, 2023
has_device_addr

```c
#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(aloc: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;
}
```

### Use with:
- Target data

indicates that each list item already has corresponding storage on the device or is accessible on the device.
```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!
...
#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: alloc(unknown)[32]
Libomptarget device 0 info: Creating new map entry with HstPtrBegin=0x0000007ffe70620c00, TgtPtrBegin=0x00000000023c5000, Size=400, DynRefCount=1, HoldRefCount=0, Name=unknown
Libomptarget device 0 info: Copying data from host to device, HstPtr=0x0000007ffe70620c00, TgtPtr=0x00000000023c5000, Size=400, Name=unknown
Libomptarget device 0 info: Creating new map entry with HstPtrBegin=0x0000007ffe70620a70, TgtPtrBegin=0x00000000023c5200, Size=400, Name=unknown
Libomptarget device 0 info: Creating new map entry with HstPtrBegin=0x0000007ffe70620c00, TgtPtrBegin=0x00000000023c5000, Size=400, Name=unknown
Libomptarget device 0 info: Creating new map entry with HstPtrBegin=0x0000007ffe70620a70, TgtPtrBegin=0x00000000023c5200, Size=400, Name=unknown
Libomptarget device 0 info: Creating new map entry with HstPtrBegin=0x0000007ffe70620c00, TgtPtrBegin=0x00000000023c5000, Size=400, Name=unknown
Libomptarget device 0 info: Creating new map entry with HstPtrBegin=0x0000007ffe70620a70, TgtPtrBegin=0x00000000023c5200, Size=400, Name=unknown
Mixing openMP* and SYCL Code Sample

```c
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.