OpenMP Offload in oneAPI HPC Toolkit

oneAPI – 가속 컴퓨팅을 개발하기 위한 스마트한 방식

2022. 06. 17.

MOASYS

oneAPI Smart Development Series (2021)

1. Introduction to Intel oneAPI for HPC and AI-DL
   - https://www.allshowtv.com/detail.html?idx=474

2. Benchmarking the Performance of oneAPI on Heterogeneous Computing Platforms
   - https://www.allshowtv.com/detail.html?idx=660

3. Optimization and GPU Offloading Workflow with Intel oneAPI
   - https://www.allshowtv.com/detail.html?idx=826

4. Leveraging Intel® oneDNN for AI Workload
   - https://www.allshowtv.com/detail.html?idx=909

oneAPI Smart Development Series (2022)

1. FPGA Development Flow with Intel® oneAPI Base Toolkit
   - https://www.allshowtv.com/detail.html?idx=995

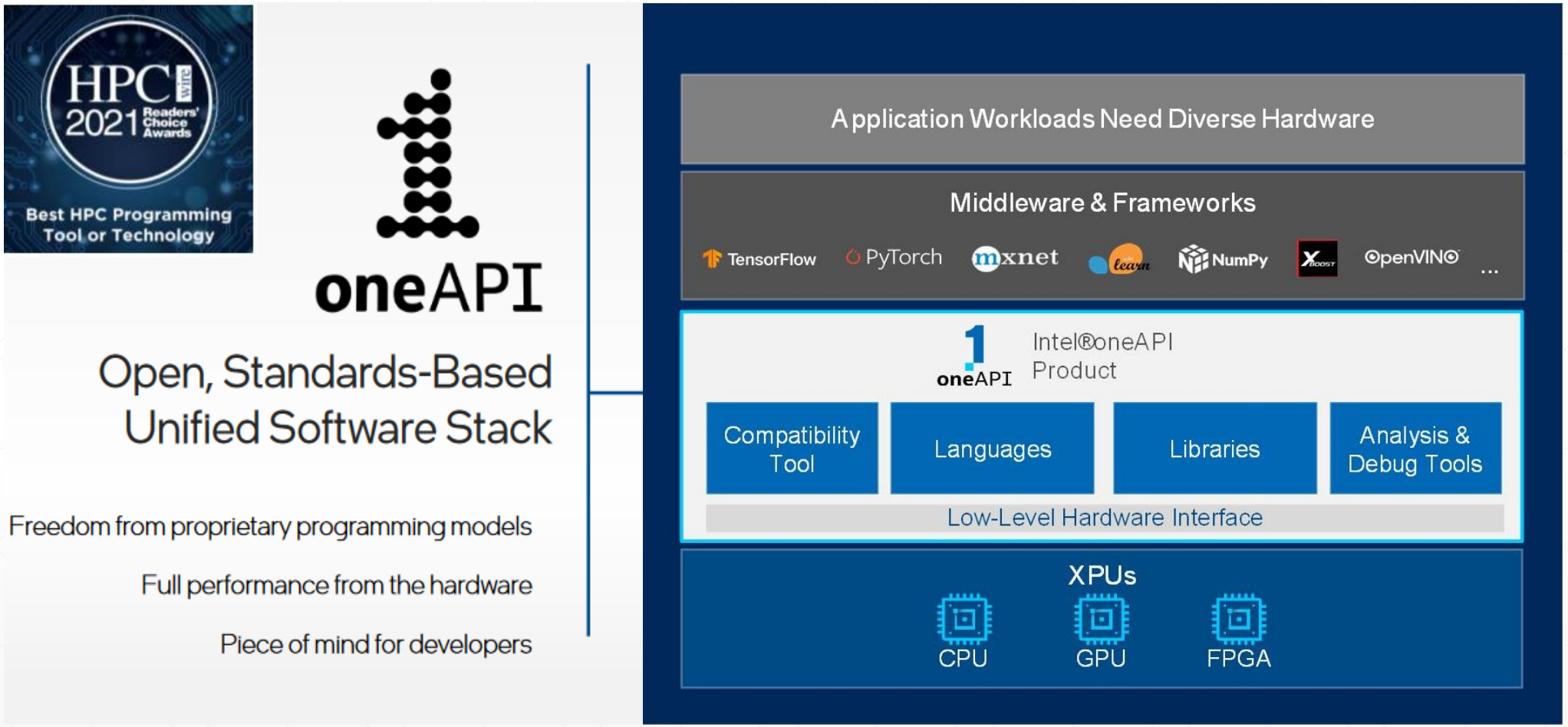
2. OpenMP Offload with Intel® oneAPI HPC Toolkit
   - https://www.allshowtv.com/detail.html?idx=1041

3. DPC++ Optimization Techniques for Accelerators
   - TBA

4. Introduction to Intel® oneAPI Rendering Toolkit
   - TBA

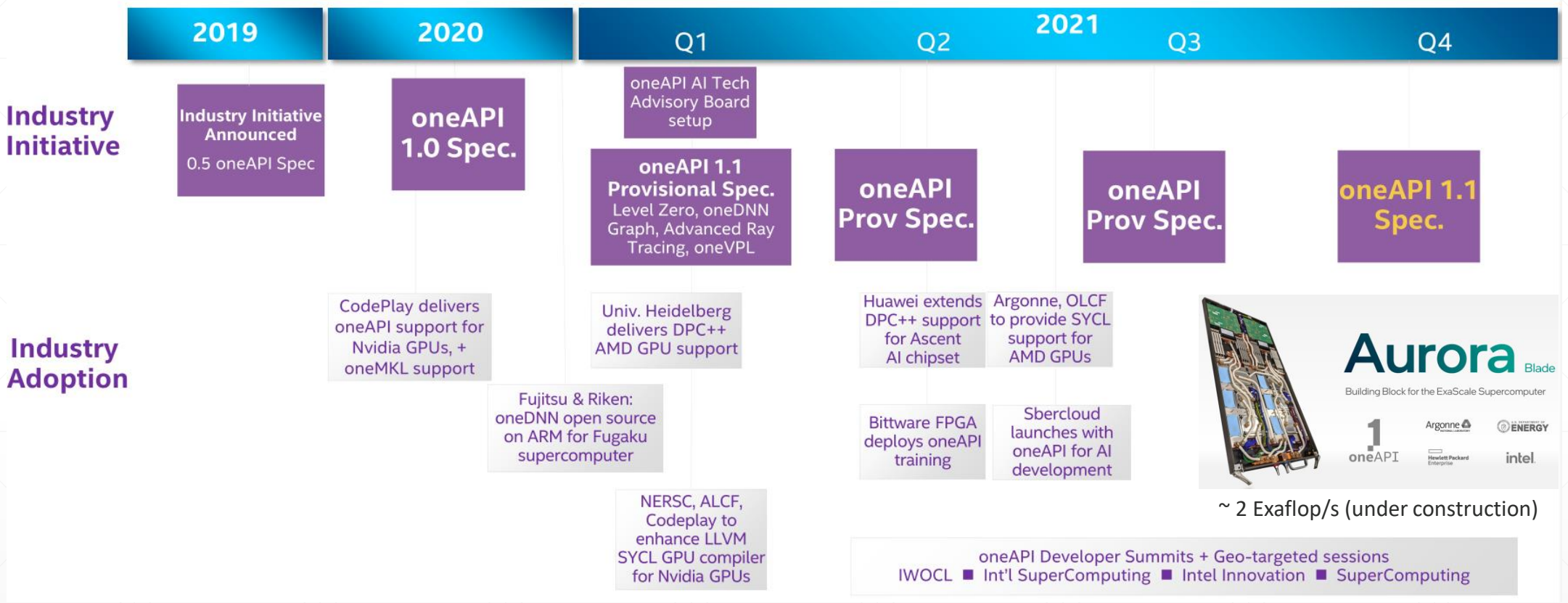
Contents

Overview of oneAPI for Heterogenous Computing



- Support diverse accelerator devices (XPU) such as CPU, GPU and FPGA
- Continuously evolving specifications for high performance computing and machine learning

oneAPI Industry Initiative Progress

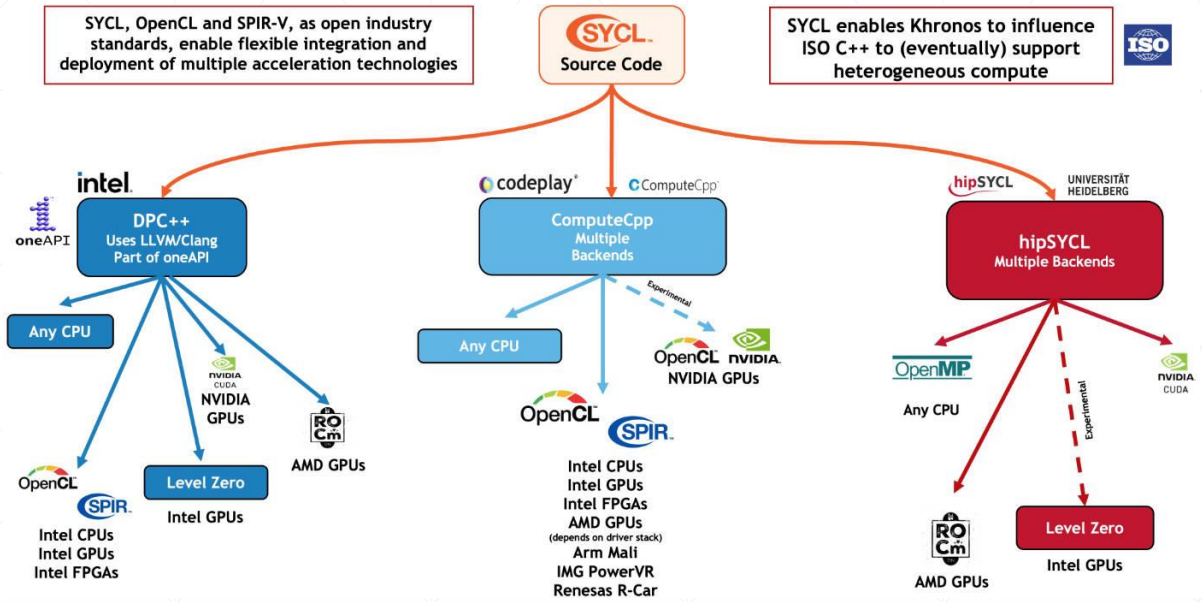


- Cross-Vendor implementations:
  - ARM CPU: Fujitsu & Riken (Japan)
  - NVIDIA GPU: CodePlay (USA), NERSC (USA), Argonne National Lab (USA)
  - AMD GPU: Heidelberg Computing Center (Germany), Argonne National Lab (USA), Oak Ridge National Lab(USA)

Top500: June 2022

Rank	System	Cores	Rmax (PFlop/s)	Rpeak (PFlop/s)	Power (kW)
1 AMD GPU	**Frontier** - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, HPE DOE/SC/Oak Ridge National Laboratory United States	8,730,112	1,102.00	1,685.65	21,100
2 ARM	**Supercomputer Fugaku** - Supercomputer Fugaku, A64FX 48C 2.2GHz, Tofu interconnect D, Fujitsu RIKEN Center for Computational Science Japan	7,630,848	442.01	537.21	29,899
3 AMD GPU	**LUMI** - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, HPE EuroHPC/CSC Finland	1,110,144	151.90	214.35	2,942

https://www.top500.org/lists/top500/2022/06/



https://www.khronos.org/sycl/

- Intel DPC++ Compiler: oneAPI Base Toolkit
  - OpenCL backend: optimized for Intel CPUs, GPUs (Gen9, 11, Xe) and FPGA (Stratix, Aria)
  - Level Zero backend: low-level offloading API currently supporting only Intel GPUs
- Intel LLVM Compiler: open source project
  - CUDA backend: experimental support for NVIDIA GPUs
  - HIP backend: experimental support for AMD GPUs via ROCm 4.x

Intel Fortran Compilers: Classic vs. Modern

- ifort: Intel® Fortran Compiler Classic
  - Intel optimizer/code generation
  - Support only CPUs
  - Support full Fortran standards

- ifx: Intel® Fortran Compiler
  - LLVM optimizer/code generation with Intel enhancements
  - Support CPUs and OpenMP Offload to Intel GPUs
  - Support major features of OpenMP 5.0 and 5.1
  - Support major of Fortran standards:
    - Fortran 1995: fully implemented
    - Fortran 2003: fully implemented
    - Fortran 2008: fully implemented excepts coarrays
    - Fortran 2018: continuous implementation
  - Full Fortran support planned by the end of 2022

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

Porting Guide

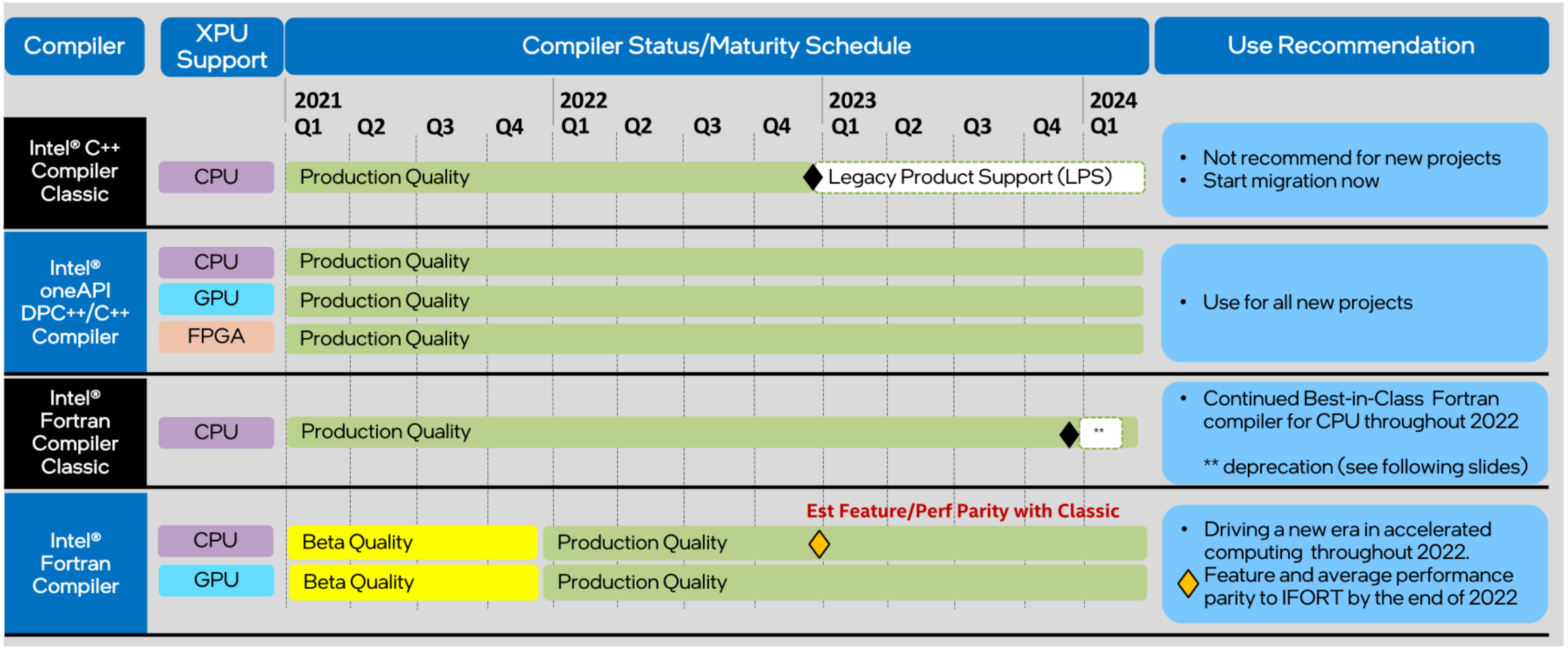
Compiler options	ifort	ifx
Disable optimization	-O0	-O0
Optimization for speed (no code size increase)	-O1	-O1
Optimization for speed	-O2	-O2 **-xSAPPHIRERAPIDS**
High-level loop optimization	-O3	-O3 **-xSAPPHIRERAPIDS**
AVX512 vector width	-qopt-zmm-usage=high	**-mprefer-vector-width=512**
Floating point optimization	-fp model fast=2	-fp model fast=2 **-assume nan_compares**
Micoarchitecture optimization	-xSAPPHIRERAPIDS	-xSAPPHIRERAPIDS
Interprocedural optimization	-ipo	**-flto**
OpenMP support	-qopenmp	**-fiopenmp**
Just-in-time compilation	N/A	**-fopenmp-targets=spir64**
Ahead-of-time compilation	N/A	**-fopenmp-targets=spir64_gen -Xopenmp-target-backend "-device xehp"**
Optimization report	-qopt-report=5	**-qopt-report=3**
Optimization phase report	-qopt-report-phase=loop	N/A
Optimization report filter	-qopt-report-routine=stencile	N/A

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

Porting Guide

- Archive performance parity between ifort and ifx
  - *-O2*: auto-vectorization, loop unrolling, loop interchange, etc.
    - ifort: inclusion of Intel performance optimizations
    - ifx: inclusion of LLVM optimizations, i.e. NO Intel optimization!
  - *-O3*: more aggressive loop optimization, such as cache blocking, prefetching, loop fusion, etc.
  - *-x<core>*: optimization for specific instruction sets:
    - SIMD extension: *-xCORE-AVX512*
    - Microarchitecture optimization: *-xSAPPHIRERAPIDS* (in oneAPI 2022.1)
    - Enable Intel optimizations on top of LLVM for ifx at -O2/-O3 levels
  - *-ax<code>*: fat binary with optimization for multiple instructions sets:
    - Not yet supported by ifx
- Not all ifort's options are ported to ifx
  - List options supported and unsupported by ifx: *-qnextgen-diag*
- CMake tool chain:
  - v3.21.3 for oneAPI compilers support

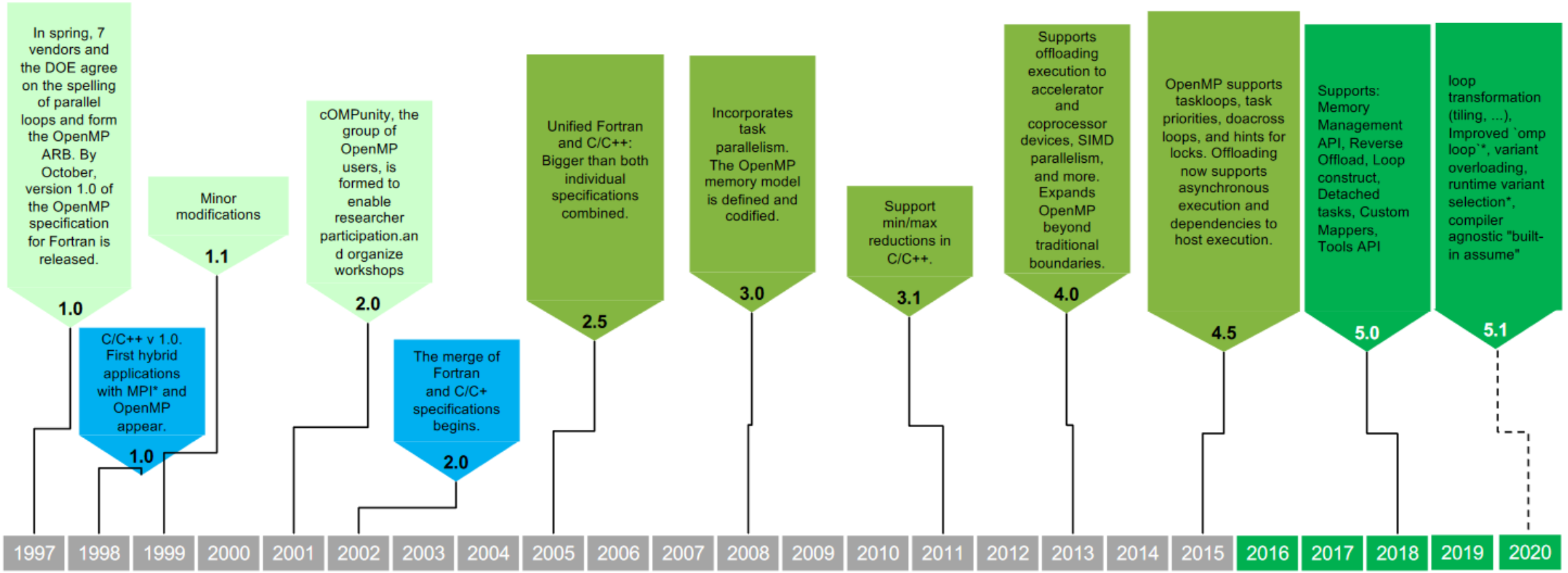
Intel Compilers Roadmap



https://www.ixpug.org/resources/intel-fortran-compilers-a-tradition-of-trusted-application-performance

- **Production quality**: ifx/icx/dpcpp++ have passed performance and validation tests, and are ready to use

Continuous Evolution of OpenMP



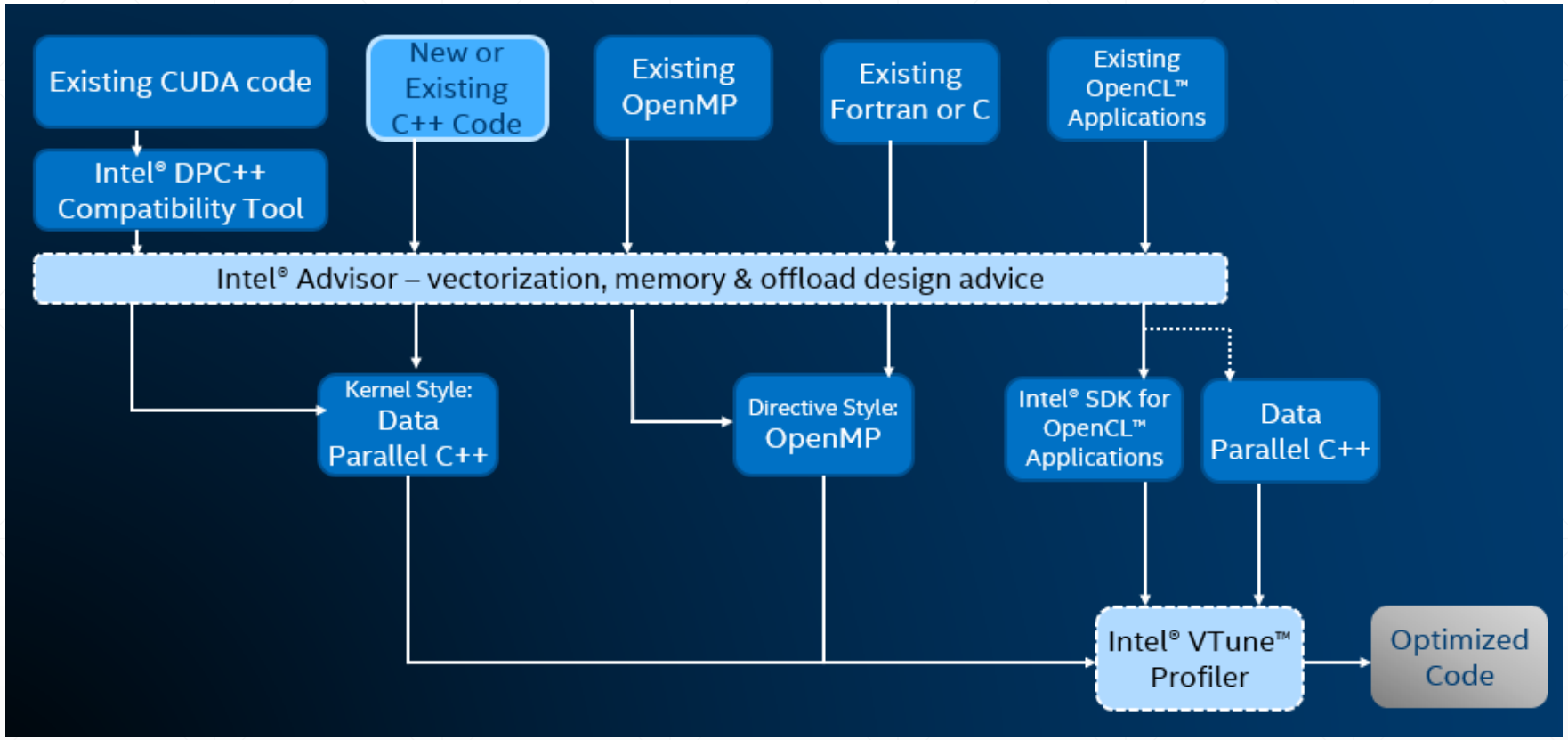
- OpenMP (Open Multi-Processing):
  - Open-source, portable and directive-based API for shared-memory parallelism in C/C++/Fortran
  - OpenMP 4.0: support for heterogenous offloading
  - OpenMP 4.5: support for asynchronous execution and dependencies to host execution

Perspectives on Heterogenous Programming Models

	CUDA	HIP	OpenACC	OpenMP	DPC++
Languages	C/C++/Fortran	C/C++/Fortran	C/C++/Fortran	C/C++/Fortran	C++
Abstraction	Low	Low	High	High	Medium
Coding	-	-	Directive-based	Directive-based	C++ lambda
Parallelism	SIMT	SIMT	Fork-join SIMD	Fork-join SIMD	OpenCL
Offload	GPU (NVIDIA)	GPU (NVIDIA/AMD)	GPU (NVIDIA)	CPU/GPU (NVIDIA/AMD/Intel)	CPU/GPU/FPGA (NVIDIA/AMD/Intel)
Compiler	Proprietary	LLVM	PGI/CCE/GCC	PGI/CCE/GCC/LLVM/XL	LLVM
License	Proprietary	Open-source	Open-source	Open-source	Open-source

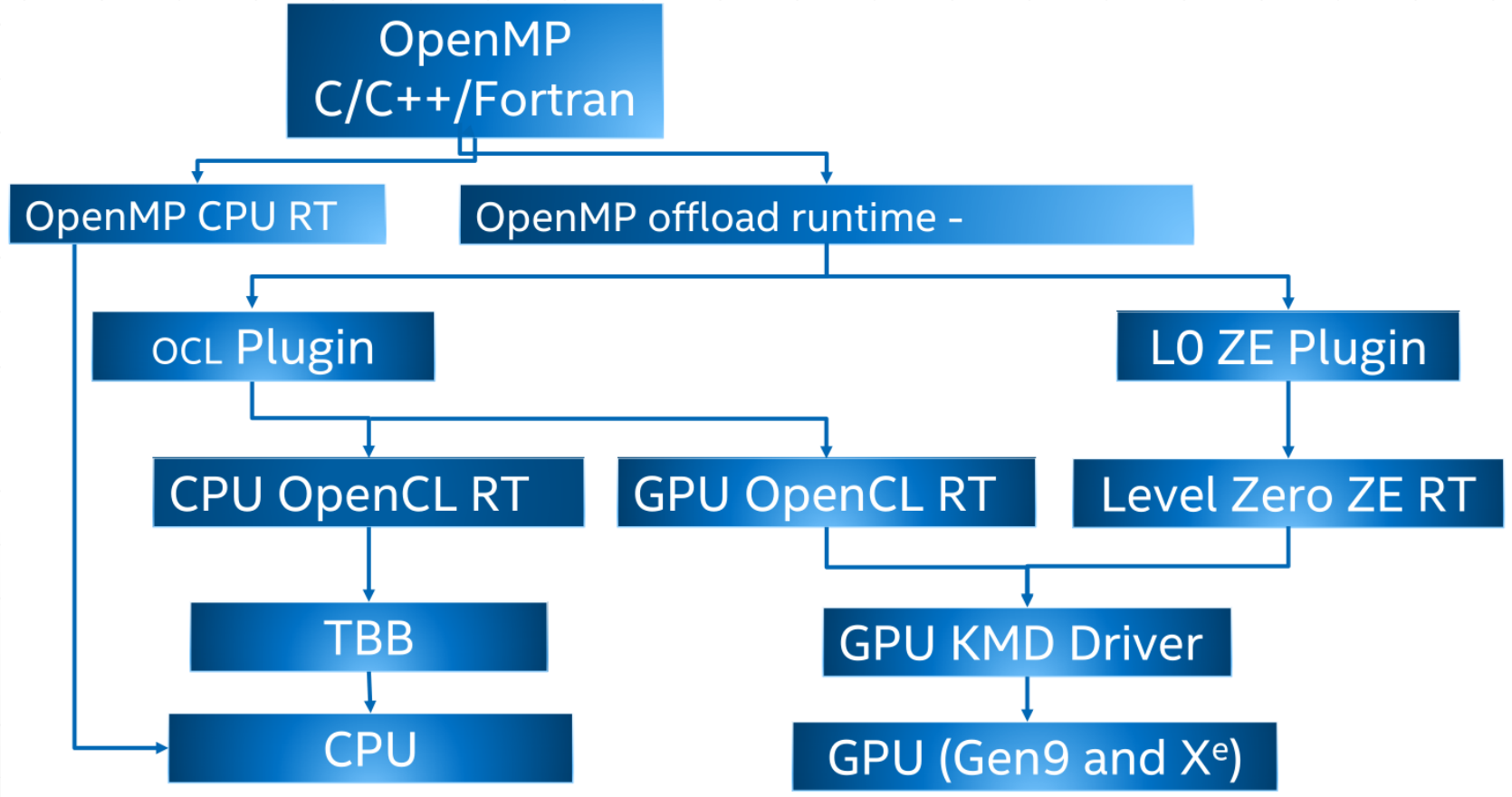
- Important features of OpenMP for GPU offloading:
  - Target directives
  - Management of data transfer
  - Constructs and teams for GPU parallelism
  - Unified Shared Memory (USM)

OpenMP Offload Workflow with Intel® oneAPI



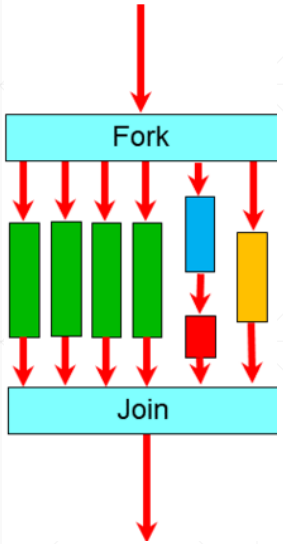
- OpenMP offloading style is suitable for:
  - Migration of existing OpenMP codes
  - Migration of existing Fortran or C codes
  - Intel® Offload Advisor can be used to identify suitable candidates for GPU offloading

OpenMP Offload Architecture in Intel® oneAPI



- Dependencies for OpenMP offload:
  - Intel® CPU/GPU OpenCL runtime
  - Intel® Thread Building Blocks (TBB) for CPU target (provided by oneAPI Basic Toolkit)

OpenMP Offload: target




```
subroutine saxpy(a, x, y, n)
    use iso_fortran_env
    integer                             :: n,i
    real(kind=real32)                   :: a
    real(kind=real32), dimension(n) :: x
    real(kind=real32), dimension(n) :: y
    !$omp parallel do
    do i=1, n
        y(i) = a * x(i) + y(i)
    end do
    !$omp end parallel do
end subroutine
```

```
subroutine saxpy_offload(a, x, y, n)
    use iso_fortran_env
    integer                             :: n,i
    real(kind=real32)                   :: a
    real(kind=real32), dimension(n) :: x
    real(kind=real32), dimension(n) :: y
    !$omp target
    do i=1, n
        y(i) = a * x(i) + y(i)
    end do
    !$omp end target
end subroutine
```

host

device

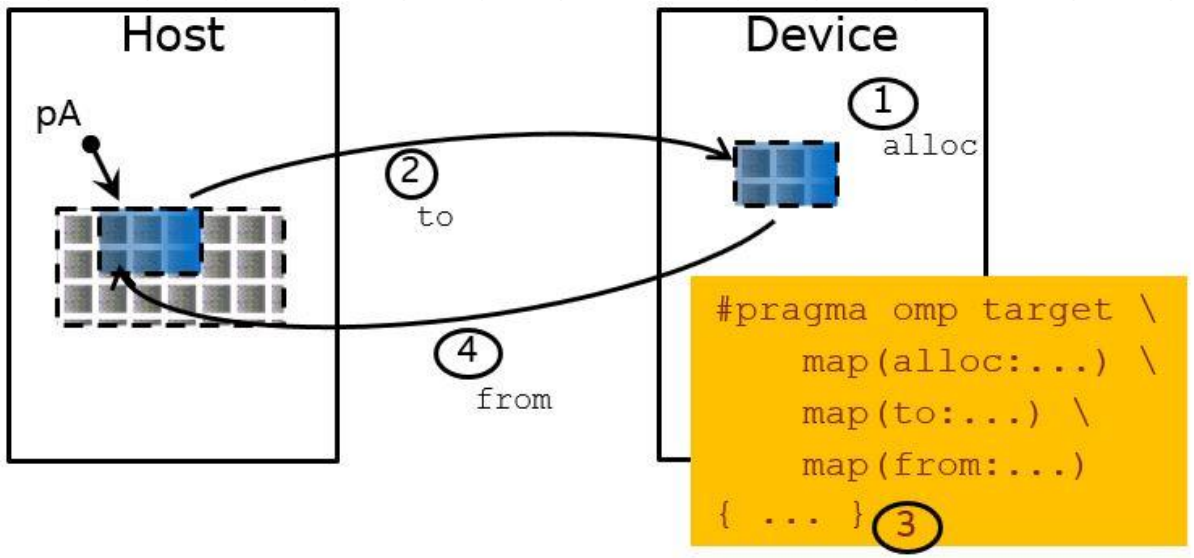
- target: offload kernel to device
- C/C++:
  - *#pragma omp target [clause]*
- Fortran:
  - *!$omp target [clause]*
    !$omp end target
- OpenMP clause:
  - *device(scalar-integer-expression)*
  - *if(scalar-expr)*
  - *map([{alloc | to | from | tofrom}:] list)*

OpenMP Offload: Implicit vs. Explicit Data Movement

```
subroutine saxpy_offload(a, x, y, n)
    use iso_fortran_env
    integer                           :: n,i
    real(kind=real32)                 :: a
    real(kind=real32), dimension(n) :: x
    real(kind=real32), dimension(n) :: y
    !$omp target
    do i=1, n
        y(i) = a * x(i) + y(i)
    end do
    !$omp end target
end subroutine
```

host

device



- OpenMP execution model:
  - Distinct memory spaces for host and devices
- OpenMP implicit data movement
  - Scalar (a): treated as first private
  - Static arrays (x, y): host → device on target entry, and device → host on target exit
  - Dynamics arrays (x,y): same as static case, but dimension must be specified

OpenMP Offload: Implicit vs. Explicit Data Movement

```
subroutine saxpy_offload(a, x, y, n)
    use iso_fortran_env
    integer                          :: n,i
    real(kind=real32)                :: a
    real(kind=real32), dimension(n) :: x
    real(kind=real32), dimension(n) :: y
    !$omp target map(tofrom: y) map(to:x)
    do i=1, n
        y(i) = a * x(i) + y(i)
    end do
    !$omp end target
end subroutine
```

host

device

```
subroutine saxpy_offload(a, x, y, n)
    use iso_fortran_env
    integer                          :: n,i
    real(kind=real32)                :: a
    real(kind=real32), dimension(:) :: x
    real(kind=real32), dimension(:) :: y
    !$omp target map(tofrom: y(1:n)) map(to:x(1:n))
    do i=1, n
        y(i) = a * x(i) + y(i)
    end do
    !$omp end target
end subroutine
```

host

device

- map: explicit control movement of data
- Available map type:
  - *alloc*: allocated but not initialized
  - *to*: host → device copy on target entry
  - *from*: device → host copy on target exit
  - *tofrom*: both *to* and *from* (default)
- Pointers (C/C++) and dynamically allocated arrays (Fortran): dimensions are required

OpenMP Offload: Optimized Data Movement

```
subroutine saxpy_offload(a, x, y, n)
    use iso_fortran_env
    integer :: n, i
    real(kind=real32) :: a
    real(kind=real32), dimension(n) :: x
    real(kind=real32), dimension(n) :: y
    !$omp target map(from:x) map(fromto:y)
    do i=1, n
        y(i) = a * x(i) + y(i)
    end do
    !$omp end target

    call init (y, n, 1.0)

    !$omp target map(from:x) map(fromto:y)
    do i=1, n
        y(i) = a * x(i) * y(i)
    end do
    !$omp end target
end subroutine
```

Allocation of data on host

On target region entry
- allocate(x,y) / x: host → device / y: host → device

On target region exit
- y: device → host / deallocate(x,y)

Reinitialize y on host

On target region entry:
- allocate(x,y) / x: host → device / y: host → device

On target region exit:
- y: device → host / deallocate(x,y)

- Optimization of data movement across multiple target regions
  - x is copied back and forth between host and device

OpenMP Offload: target data

```
subroutine saxpy_offload(a, x, y, n)
    use iso_fortran_env
    integer :: n, i
    real(kind=real32) :: a
    real(kind=real32), dimension(n) :: x
    real(kind=real32), dimension(n) :: y
    !$omp target data map(to:x)
        !$omp target map (to:y)
        do i=1, n
            y(i) = a * x(i) + y(i)
        end do
        !$omp end target

        call init(y, n, 1.0)

        !$omp target map(to:y)
        do i=1, n
            y(i) = a * x(i) * y(i)
        end do
        !$omp end target
    !$omp end target data
end subroutine
```

host

device

host

device

Allocation of data on host

On target region entry

- allocate(x,y) / x: host → device / y: host → device

On target region exit

- y: device → host / deallocate(y)

Reinitialize y on host

On target region entry

- allocate(y) / y: host → device

On target region exit

- y: device → host / deallocate(x,y)

- x is allocated and copied once in target data region

OpenMP Offload: target enter/target exit/target update

```
subroutine saxpy_offload(a, x, y, n)
    use iso_fortran_env
    integer :: n, i
    real(kind=real32) :: a
    real(kind=real32), dimension(n) :: x
    real(kind=real32), dimension(n) :: y
    !$omp target enter data map(to:x) map(to:y)
        !$omp target
        do i=1, n
            y(i) = a * x(i) + y(i)
        end do
        !$omp end target

        call init (y, n, 1.0)
        !$omp target update to(y)
        !$omp target
        do i=1, n
            y(i) = a * x(i) * y(i)
        end do
        !$omp end target
    !$omp target exit data map(from:y)
end subroutine
```

host

device

host

device

Allocation of data on host

On target region entry

- allocate(x,y) / x: host → device / y: host → device

Reinitialize y on host

Force update

- y: host → device

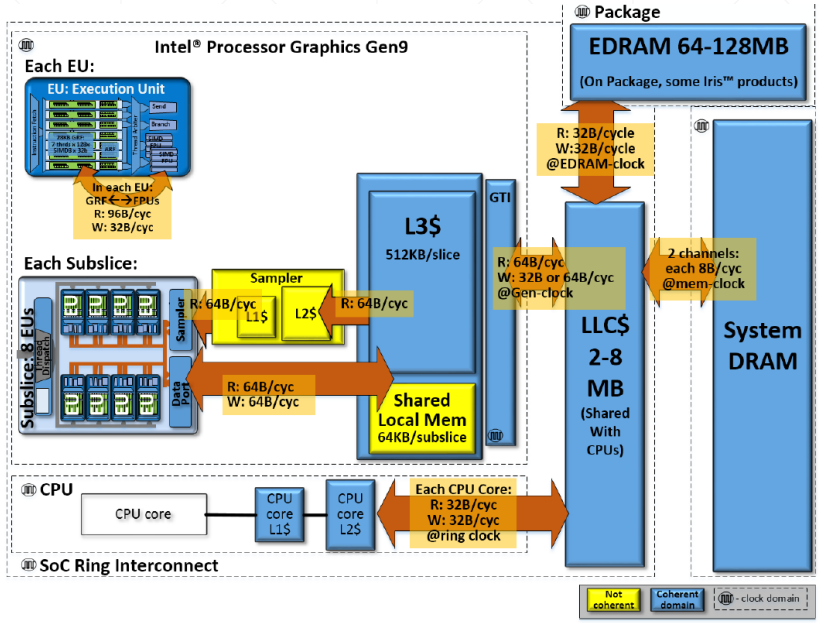
On target region exit

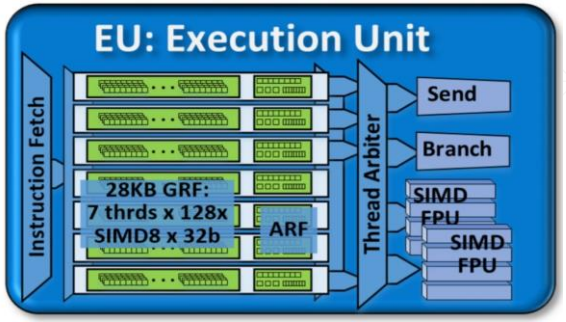
- y: device → host / deallocate(x,y)

- x is allocated and copied once in target data region

OpenMP offload: Device Parallelism

```
subroutine saxpy_offload(a, x, y, n)
    use iso_fortran_env
    integer :: n, i
    real(kind=real32) :: a
    real(kind=real32), dimension(n) :: x
    real(kind=real32), dimension(n) :: y
    !$omp target map(tofrom: y) map(to:x)
    !$omp parallel do simd
    do i=1, n
        y(i) = a * x(i) + y(i)
    end do
    !$omp end target
end subroutine
```





- 1 x slice
- 3 x subslices
- 8 x EUs per subslice
- 7 x threads per EU
- 128 x SIMD8 lanes per thread

- Separation between OpenMP offload and parallelism
  - *target* creates a single device thread
- Explicit parallel region required
  - *parallel do simd* maps OpenMP threads to a single GPU subslice
    - No synchronization among subslice
    - No memory fence between subslice L1 caches

OpenMP offload: team

```
subroutine saxpy_offload(a, x, y, n)
    use iso_fortran_env
    integer :: n, i
    real(kind=real32) :: a
    real(kind=real32), dimension(n) :: x
    real(kind=real32), dimension(n) :: y
    !$omp target map(tofrom: y) map(to:x)
    !$omp parallel do simd
    do i=1, n
        y(i) = a * x(i) + y(i)
    end do
    !$omp end target
end subroutine
```

host

device

```
subroutine saxpy_offload(a, x, y, n)
    use iso_fortran_env
    integer :: n, i
    real(kind=real32) :: a
    real(kind=real32), dimension(n) :: x
    real(kind=real32), dimension(n) :: y
    !$omp target map(tofrom: y) map(to:x)
    !$omp teams distribute parallel do simd
    do i=1, n
        y(i) = a * x(i) + y(i)
    end do
    !$omp end target
end subroutine
```

host

device

- team: creation of master threads mapped to subslices
- C/C++:
  - *#pragma omp team [clause]*
- Fortran:
  - *!$omp team [clause]*
- OpenMP clause:
  - *num_teams(integer-expression), thread_limit(integer-espression)*
  - *default(shared | first private | private none)*
  - *private(list), firstprivate(list), shared(list), reduction(operation:list)*

OpenMP offload: distribute

```
subroutine saxpy_offload(a, x, y, n)
    use iso_fortran_env
    integer :: n, i
    real(kind=real32) :: a
    real(kind=real32), dimension(n) :: x
    real(kind=real32), dimension(n) :: y
    !$omp target map(tofrom: y) map(to:x)
    !$omp parallel do simd
    do i=1, n
        y(i) = a * x(i) + y(i)
    end do
    !$omp end target
end subroutine
```

host

device

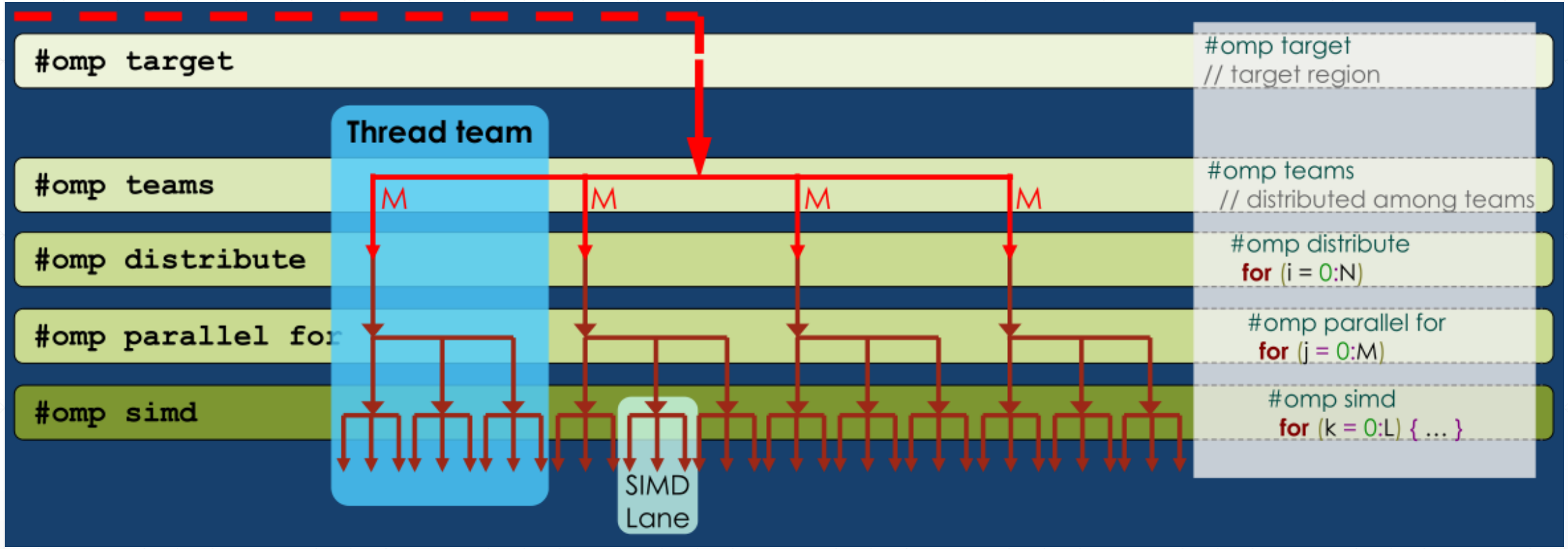
```
subroutine saxpy_offload(a, x, y, n)
    use iso_fortran_env
    integer :: n, i
    real(kind=real32) :: a
    real(kind=real32), dimension(n) :: x
    real(kind=real32), dimension(n) :: y
    !$omp target map(tofrom: y) map(to:x)
    !$omp teams distribute parallel do simd
    do i=1, n
        y(i) = a * x(i) + y(i)
    end do
    !$omp end target
end subroutine
```

host

device

- distribute: distribution of loop iterations across different teams
- C/C++:
  - *#pragma omp distribute [clause]*
- Fortran:
  - *!$omp distribute [clause]*
- OpenMP clause:
  - *collapse(n)*
  - *dist_schedule(static[,chunk_size])*
  - *firstprivate(list), lastprivate(list), private(list)*

OpenMP offload: Device Parallelism



- target: create a single device thread
- teams: create a group of master threads, mapped to iGPU subslice
- distribute: assign chunks of loop iterations to teams
- parallel for simd: run each chunk of iterations on EU threads and SIMD lanes

OpenMP offload: Host Device Concurrency

```
subroutine saxpy_offload(a, x, y, n)
    use iso_fortran_env
    integer :: n, i
    real(kind=real32) :: a
    real(kind=real32), dimension(n) :: x
    real(kind=real32), dimension(n) :: y

    !$omp target map(tofrom: y) map(to:x)
    !$omp parallel do simd
    do i=1, n
        y(i) = a * x(i) + y(i)
    end do
    !$omp end target


    call host_work()


print *, y(0)
end subroutine
```

host / device / host

```
subroutine saxpy_offload(a, x, y, n)
    use iso_fortran_env
    integer :: n, i
    real(kind=real32) :: a
    real(kind=real32), dimension(n) :: x
    real(kind=real32), dimension(n) :: y

    !$omp target map(tofrom: y) map(to:x) nowait
    !$omp parallel do simd
    do i=1, n
        y(i) = a * x(i) + y(i)
    end do
    !$omp end target

    !$omp task
    call host_work()

    !$omp taskwait
    print *, y(0)
end subroutine
```

host / device / host

- nowait: enable asynchronous offloading
- taskwait: synchronize target region back to host device

DPC++ and OpenMP Interoperability

```
#include <CL/sycl.hpp>
#include <array>
#include <iostream>

float computePi(int 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, int N) {
    range<1> R(N);
    buffer<int,1> X(A, R);
    queue().submit([&](handler &cgh) {
        auto Y = X.template get_access<access::mode::write>(cgh);
        cgh.parallel_for<class Iota>(R, [=](id<1> idx) {
            Y[idx] = idx;
        });
    });
}
```

```
float computePi(int);
void iota(float*, int);

int main() {
    std::array<int, 1024> V;
    float Pi;

     #pragma omp parallel sections
     {
         #pragma omp section
         iota(V.data(), V.size());

         #pragma omp section
         Pi = computePi(8192);
      }

      return 0;
}
```

```
$ icpx \
      -fiopenmp \
      -fopenmp-targets=spir64 \
      -fsycl \
      -O2 \
      mixed.cpp
```

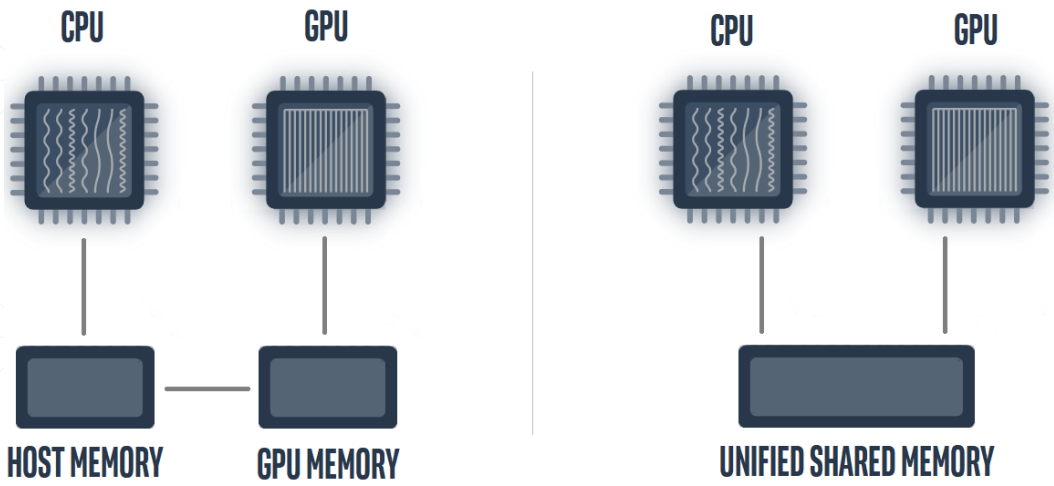
Essential OpenMP Environment Variables

- Select target device
  - OMP_TARGET_OFFLOAD = mandatory | disable | default
    - mandatory: run target region on GPU
    - disabled: run target regions on CPU
    - default: run target region on GPU if available, else false back to CPU
- Select OpenMP backend
  - LIBOMPTARGET_PLUGIN = opencl | level0
- Performance profiling on GPU
  - LIBOMPTARGET_PLUGIN_PROFILE = T | F

```
================================================================================
LIBOMPTARGET_PLUGIN_PROFILE(LEVEL0) for OMP DEVICE(0) Intel(R) UHD Graphics 630 [0x3e92], Thread 0
--------------------------------------------------------------------------------
-- Kernel 0                  : __omp_offloading_3a_dca6bdd1_MAIN___l30
--------------------------------------------------------------------------------
-- Name                      :     Host Time (msec)   Device Time (msec)
-- Compiling                 :            2632.679            2632.679
-- DataAlloc                 :              11.350              11.350
-- DataRead (Device to Host) :               2.633               2.633
-- DataWrite (Host to Device):               8.398               8.398
-- Kernel 0                  :            5279.800            5258.379
-- OffloadEntriesInit        :               2.731               2.731
-- Total                     :            7937.591            7916.170
================================================================================
```

Unified Share Memory (USM)

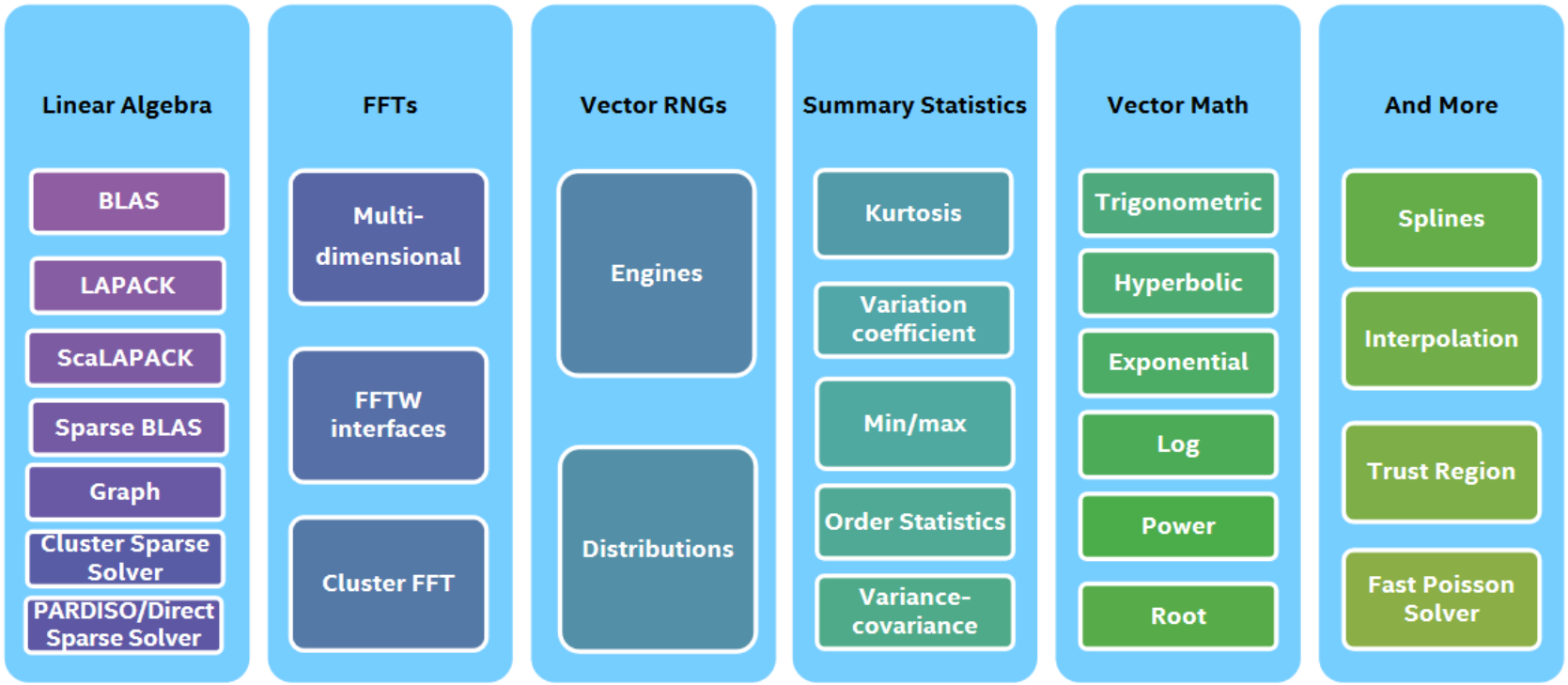
```
program main
    use omp_lib
    integer, parameter                :: n = 1024
    real(kind=real32), allocatable :: x(:), y(:)
    !$omp allocate allocator(omp_target_shared_mem_alloc)
    allocate(x(n))

    allocate(y(n))
    …
end program

subroutine saxpy_offload(a, x, y, n)
    use iso_fortran_env
    integer :: n, i
    real(kind=real32) :: a
    real(kind=real32) :: x(:)
    real(kind=real32) :: y(:)
    !$omp target map(tofrom: y(1:n)) has_device_addr(x)
    !$omp parallel do simd
    do i=1, n
        y(i) = a * x(i) + y(i)
    end do
    !$omp end target
end subroutine
```



Type	Location	Accessible From	allocate directive
Host	Host	Host or Device	!$omp allocate allocator(omp_target_host_mem_alloc)
Device	Device	Device	!$omp allocate allocator(omp_target_device_mem_alloc)
Shared	Host or Device	Host or Device	!$omp allocate allocator(omp_target_shared_mem_alloc)

- has_device_addr(list)
  - Items has a valid memory address on device
  - Can be any types, including arrays

oneMKL: OpenMP Offload to GPU



- OpenMP offload supported:
  - BLAS Level 1, 2, 3: synchronous and asynchronous executions
  - LAPACK: synchronous and asynchronous executions
  - FFTs: synchronous and asynchronous executions in 1D, 2D and 3D (FFTW3/DFTI interface)
  - Random number generator (RNG)

oneMKL: GPU Offload

```
include "common_blas.f90"
program dgemm
    use onemkl_blas_omp_offload_lp64
    use common_blas

    integer                          :: m = 1024, n = 1024, k = 256
    double precision                 :: alpha = 1.0, beta = 1.0
    double precision, allocatable    :: a(:,:), b(:,:), c_gpu(:,:), c_cpu(:,:)

    ! Array initialization
    call dinit_matrix('N', m, k, m, a)
    ...

    ! Call DGEMM on CPU for reference
    call dgemm('N', 'N', m, n, k, alpha, a, m, b, k, beta, c_cpu, m)

    ! Map matrices to device memory
    !$omp target data map(a,b,c_gpu)
        ! Call DGEMM on GPU
        !$omp target variant dispatch use_device_addr(a,b,c_gpu)
        call dgemm('N', 'N', m, n, k, alpha, a, m, b, k, beta, c_gpu, m)
        !$omp end target variant dispatch
    !$omp end target data

    ! free memory
    deallocate(a)
    ...
end program
```

host

device

host

Allocation of data on host

Initialization of matrices on host

Perform matmul on host for reference

On target region entry

- copy (a,b,c_gpu) to device memory
- ***use_device_addr()***:
  - Arrays are accessible on device

On target region exit:

- deallocate (a,b,c_gpu) on device

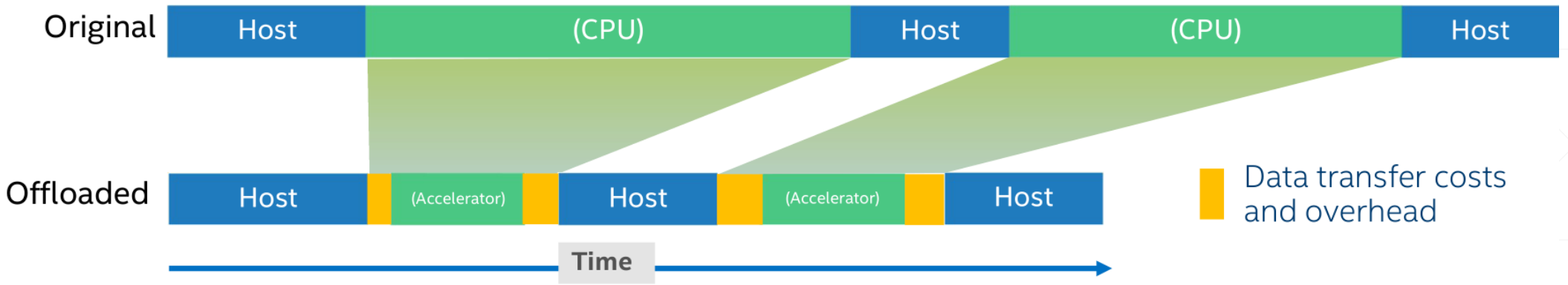
Deallocation of data on host

OpenMP Example: miniWeather

```
!$omp target teams distribute parallel do simd collapse(2) private(stencil,vals,d3_vals) depend(inout:asyncid) nowait
  do k = 1 , nz
    do i = 1 , nx+1
      !Use fourth-order interpolation from four cell averages to compute the value at the interface in question
      do ll = 1 , NUM_VARS
        do s = 1 , sten_size
          stencil(s) = state(i-hs-1+s,k,ll)
        enddo
        !Fourth-order-accurate interpolation of the state
        vals(ll) = -stencil(1)/12 + 7*stencil(2)/12 + 7*stencil(3)/12 - stencil(4)/12
        !First-order-accurate interpolation of the third spatial derivative of the state (for artificial viscosity)
        d3_vals(ll) = -stencil(1) + 3*stencil(2) - 3*stencil(3) + stencil(4)
      enddo

      !Compute density, u-wind, w-wind, potential temperature, and pressure (r,u,w,t,p respectively)
      r = vals(ID_DENS) + hy_dens_cell(k)
      u = vals(ID_UMOM) / r
      w = vals(ID_WMOM) / r
      t = ( vals(ID_RHOT) + hy_dens_theta_cell(k) ) / r
      p = C0*(r*t)**gamma

      !Compute the flux vector
      flux(i,k,ID_DENS) = r*u     - hv_coef*d3_vals(ID_DENS)
      flux(i,k,ID_UMOM) = r*u*u+p - hv_coef*d3_vals(ID_UMOM)
      flux(i,k,ID_WMOM) = r*u*w   - hv_coef*d3_vals(ID_WMOM)
      flux(i,k,ID_RHOT) = r*u*t   - hv_coef*d3_vals(ID_RHOT)
    enddo
  enddo
```

- Basic dynamics of atmospheric weather:
  - https://github.com/mrnorman/miniWeather
  - Implementation of basic scheme:
    - 4th order finite-volume
    - 3rd order Runge-Kutta integration
    - 4th order hypervicosity
  - MPI/OpenMP/OpenMP offload

Conclusion



- New Intel® Fortran/C/C++ compilers in oneAPI
  - Classic Intel® compilers will enter legacy product support (icc: 2023, ifort: 2024)
  - Migration toward new Intel® compilers are recommended
- OpenMP offload
  - Data management
  - GPU parallelism
  - Unified share memory (USM)
  - oneMKL offload
- Contact: sales@moasys.com
  - GPU, FPGA porting
  - Optimization and parallelization consultant
  - Specialized HPC education