Compiling and Running for HPC Applications

Colleen Bertoni, Thomas Applencourt, Brian Homerding, Kris Rowe, Abhishek Bagusetty, Nathan Nichols
Argonne Leadership Computing Facility

Agenda

- Quick reminder of Aurora and Programming Models available
- 15 min quickstart of each of the following programming models:
  - OpenMP offload
  - SYCL
  - Kokkos
  - OCCA
- Overview of CPU and GPU binding on Aurora nodes
- Overview of Math and other libraries on Aurora

References

- Argonne documentation
  - https://docs.alcf.anl.gov/aurora/
  - https://docs.alcf.anl.gov/aurora/getting-started-on-aurora/

Aurora

Intel® Data Center GPU Max Series

4th Gen Intel XEON Max Series CPU *with High Bandwidth Memory*

Platform
HPE Cray-Ex

Racks - 166
Nodes - 10,624
CPUs - 21,248
GPUs – 63,744

Interconnect
HPE Slingshot 11
Dragonfly topology with adaptive routing
Cassini NIC, 200 Gb/s (25 GB/s), 8 per node
Network Switch:
25.6 Tb/s per switch (64 200 Gb/s ports)
Links with 25 GB/s per direction

Peak FP64 Performance
≧ 2 exaFLOPS

Memory
10.9PiB of DDR @ 5.95 PB/s
1.36PiB of CPU HBM @ 30.5 PB/s
8.16PiB of GPU HBM @ 208.9 PB/s

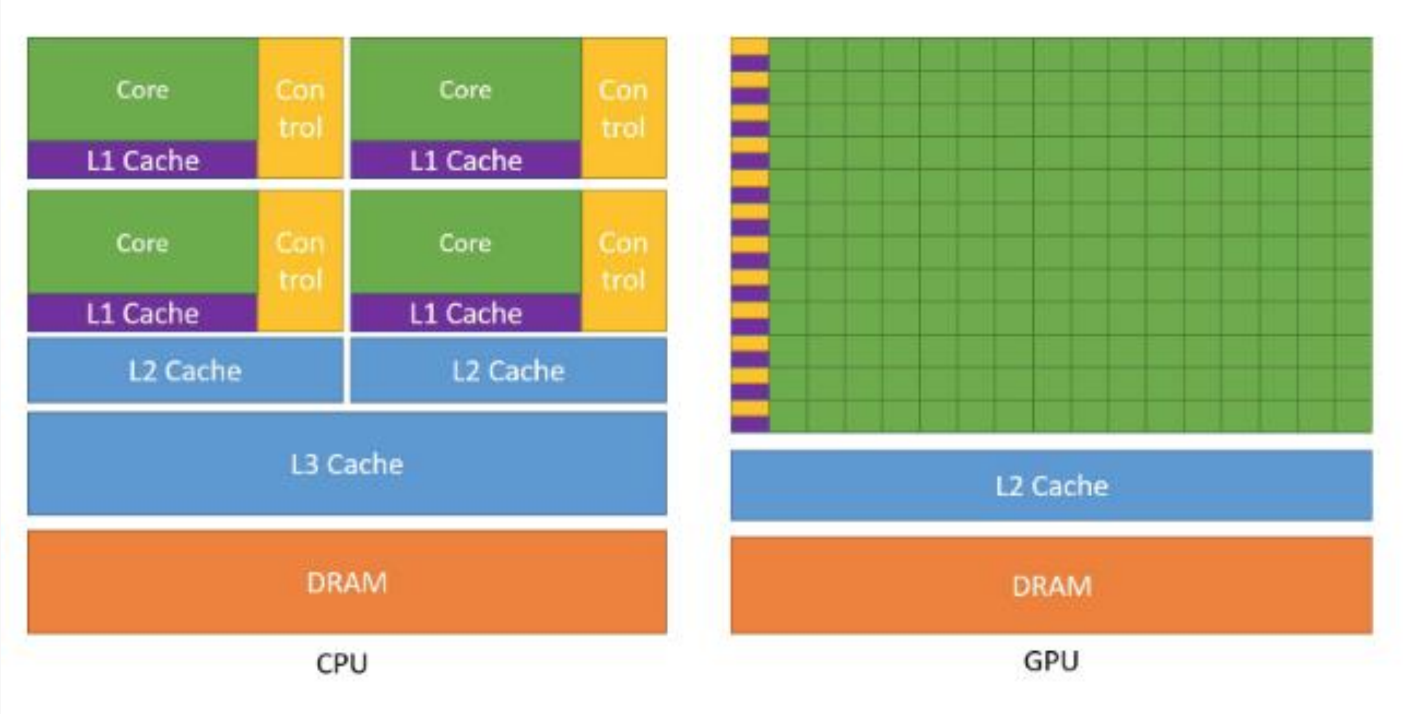
Network
2.12 PB/s Peak Injection BW
0.69 PB/s Peak Bisection BW

Storage
230PB DAOS Capacity
31 TB/s DAOS Bandwidth

Argonne NATIONAL LABORATORY

Reminder about CPU and GPU programming

- CPU
  - Optimized to reduce latency
  - Good for serial work
  - Relatively high clock frequency

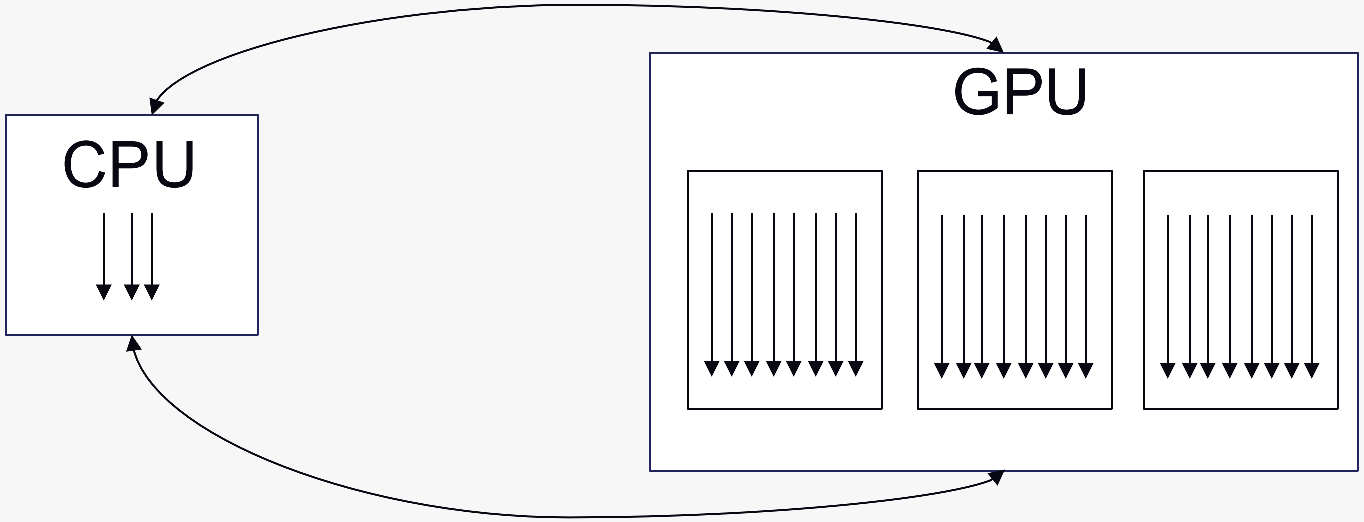


- GPU
  - Optimized for throughput
  - Good for parallel work
  - Relatively low clock frequency

From: https://www.nersc.gov/assets/Uploads/ProgrammingModels.pdf

Reminder about CPU and GPU programming

- CPU+GPU Programming
  - High-level principles
    - Serial work runs on the CPU
    - Parallel work runs on the GPU
    - Minimize transferring data between CPU and GPU



Programming Model Landscape on Aurora

Vendor Supported Programming Models

- OpenMP w/o target
- OpenMP with target
- OpenCL
- SYCL

Programming Models with experimental support

- HIP

Community Provided Programming Models

- Kokkos
- Raja

Vendor Supported AI frameworks

- Python (Numba, NumPy, etc.)
- PyTorch, TensorFlow, JAX, DeepSpeed, Horovod

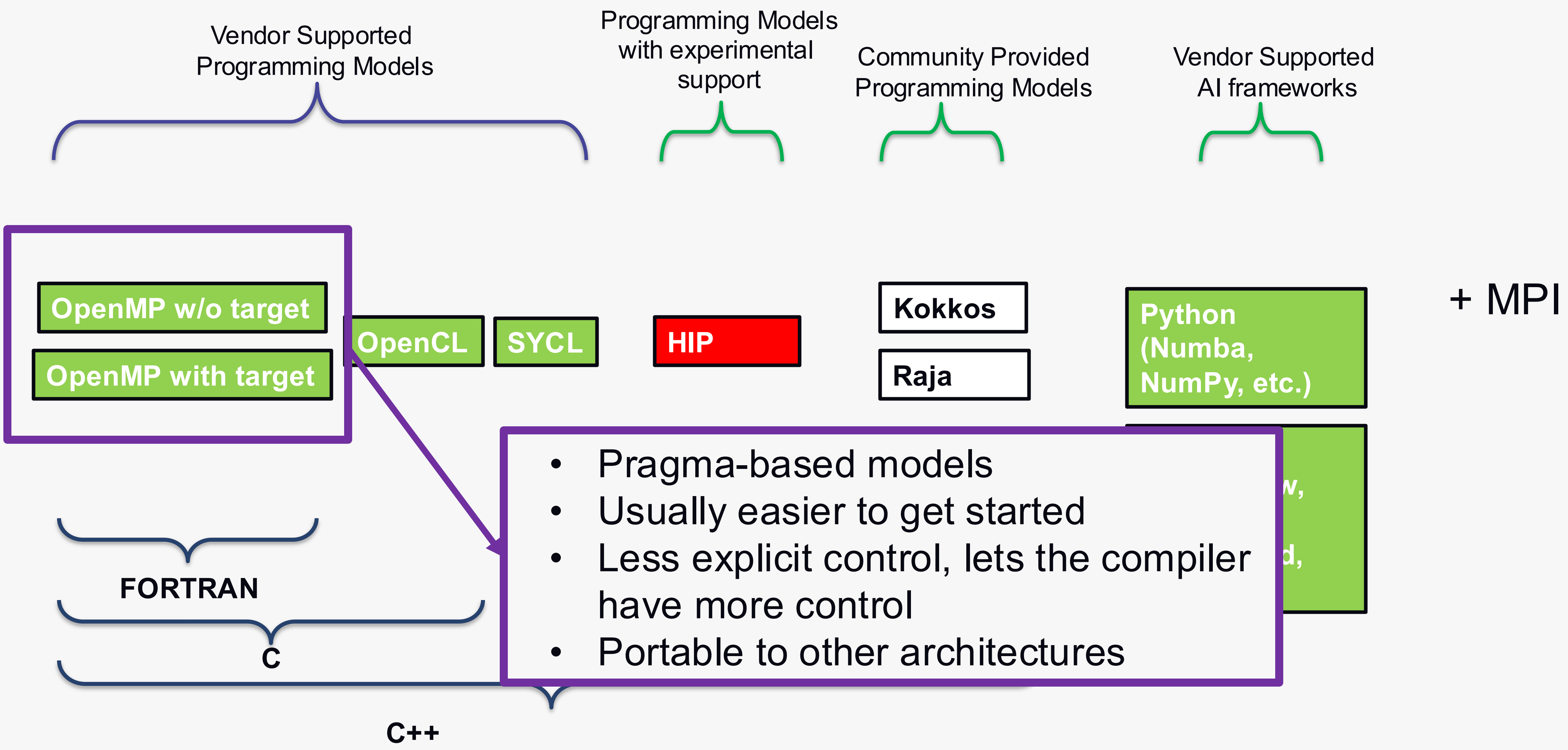
+ MPI

FORTRAN

C

C++

Programming Model Landscape on Aurora



Programming Model Landscape on Aurora

Vendor Supported Programming Models

Programming Models with experimental support

Community Provided Programming Models

Vendor Supported AI frameworks

OpenMP w/o target

OpenMP with target

OpenCL

SYCL

HIP

Kokkos

Raja

Python (Numba, NumPy, etc.)

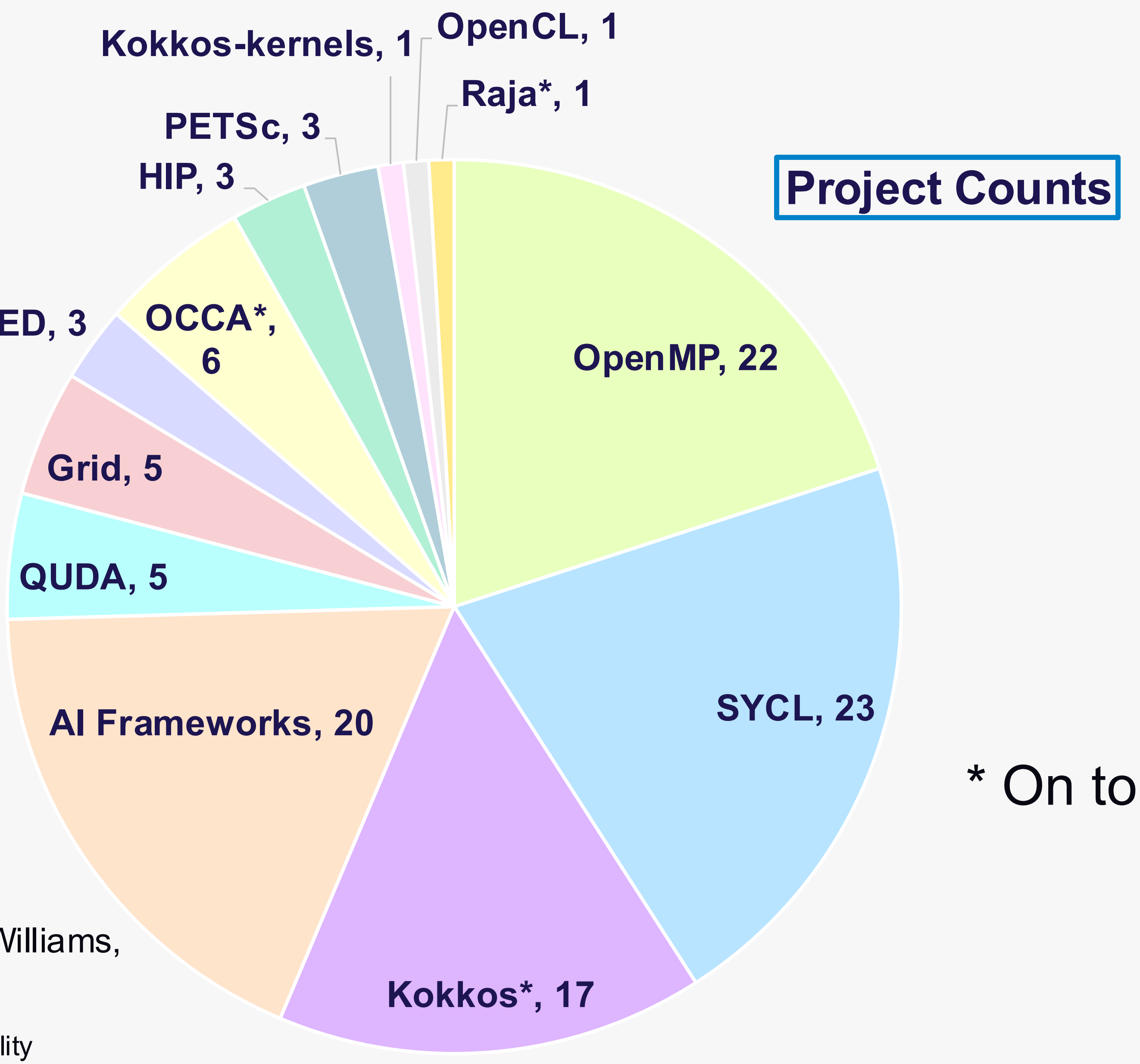
PyTorch, TensorFlow, JAX

+ MPI

- C or C++ frameworks
- Portable to other architectures
- Generally Community-provided instead of vendor-provided

C++

Programming Model Choices by Y1 Aurora Projects + ECP



* On top of SYCL

From: “Targeting Applications to First-Generation Exascale Systems”, Tim Williams, Feb. 2025

OpenMP Offload: Overview and Quickstart

Overview

- Why OpenMP?
  - Open standard for parallel programming with support across vendors
  - OpenMP runs on CPU threads, GPUs, SIMD units
  - C/C++ and Fortran
  - Supported by Intel, Cray, GNU, LLVM compilers and others
  - OpenMP offload will be additionally supported on Aurora, Frontier, Perlmutter
- Four Important high-level features to express parallelism
  - Fork and join thread parallelism
  - SIMD parallelism (added in 4.0)
  - Device Offload parallelism (added in 4.0)
  - Tasking parallelism (added in 3.0)
- Why instead of a C++ framework?
  - Easy to get started and trivial to parallelize loops
  - The reduction clause simplifies data reduction

CPU OpenMP parallelism

Spawn threads in a thread team

Distributes iterations to the threads

```
#pragma omp parallel for private(x) reduction(+:sum)
 for( int i=0; i<=num_steps; i++){
   x = (i+0.5)*step;
   sum = sum + 4.0/(1.0+x*x);
  }
```

GPU OpenMP parallelism

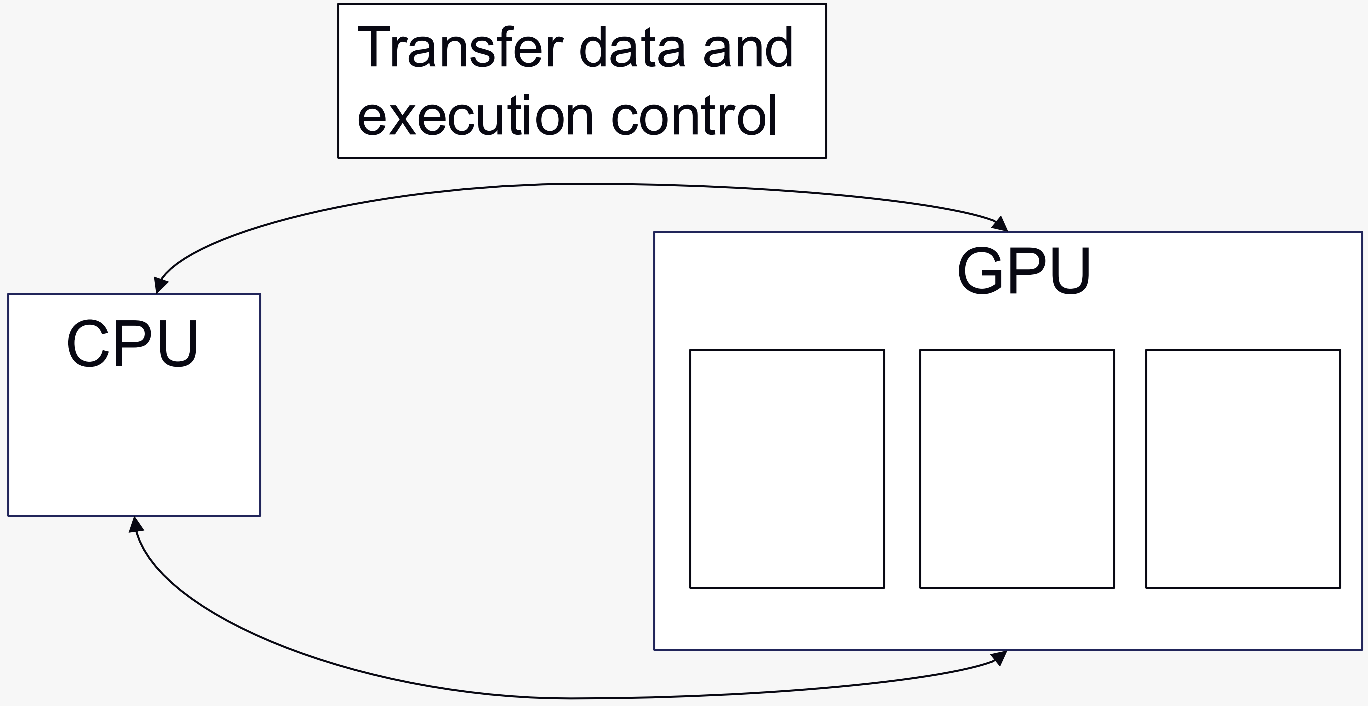
Creates teams of threads in the target device

Distributes iterations to the threads

```
#pragma omp target teams distribute parallel for private(x) reduction(+:sum)
 for( int i=0; i<=num_steps; i++){
   x = (i+0.5)*step;
   sum = sum + 4.0/(1.0+x*x);
 }
```

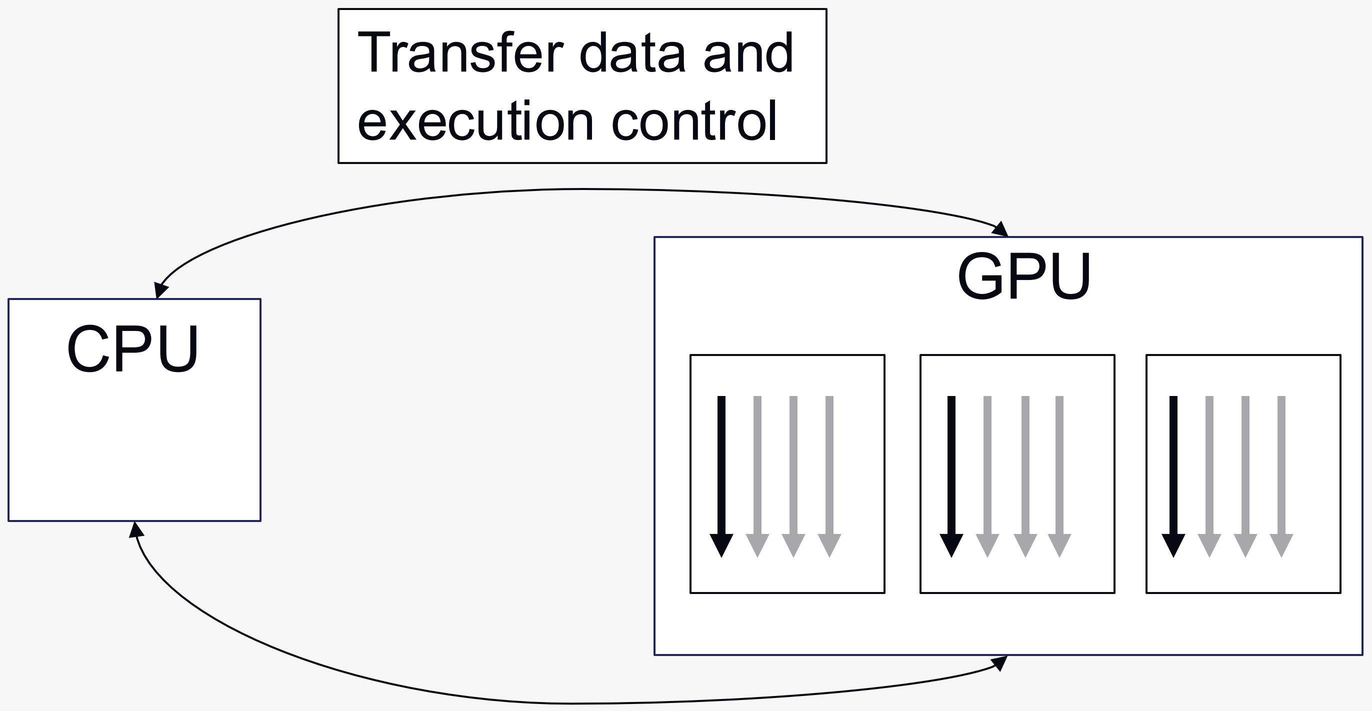
OpenMP Offload Introduction

- **Target construct**: offloads code and data to the device and runs in serial on the device



OpenMP Offload Introduction

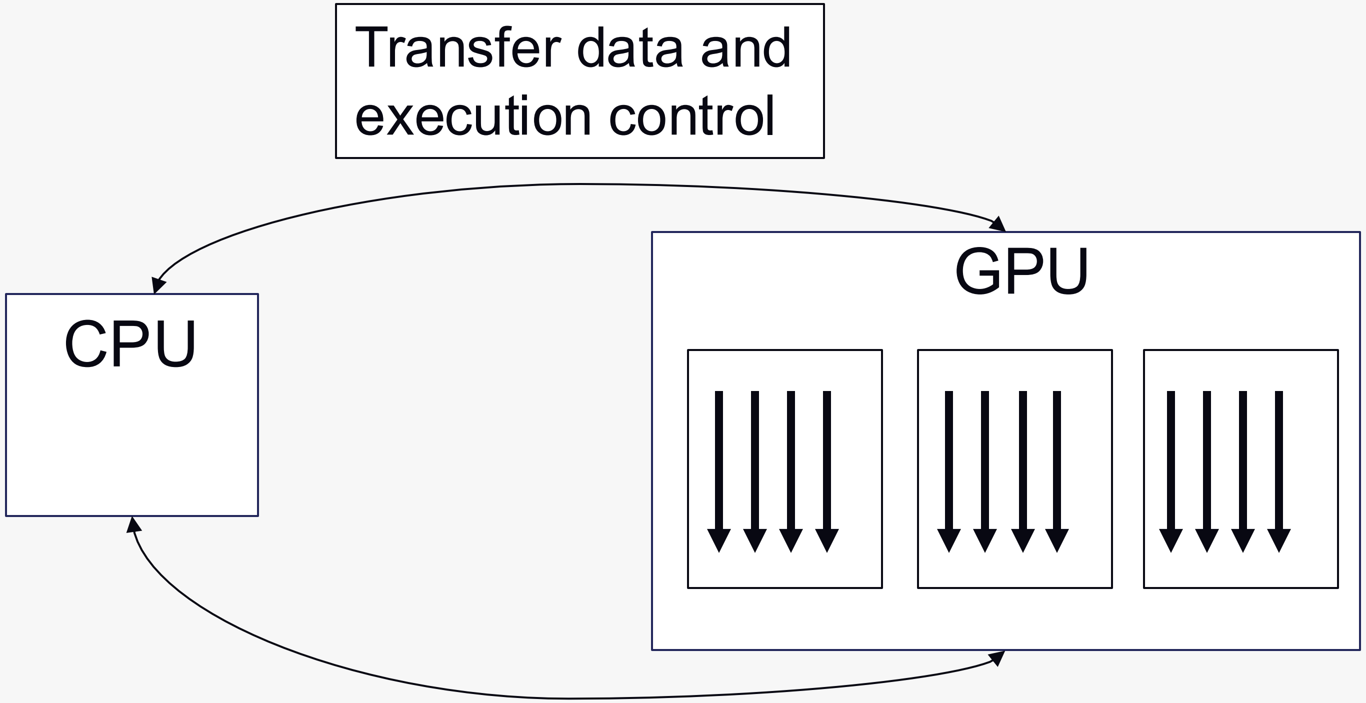
- **Target construct**: offloads code and data to the device and runs in serial on the device
- **Teams construct**: creates a league of teams, each with one thread, which run concurrently on SMs (Nvidia terminology)



Argonne NATIONAL LABORATORY

OpenMP Offload Introduction

- **Target construct**: offloads code and data to the device and runs in serial on the device
- **Teams construct**: creates a league of teams, each with one thread, which run concurrently on SMs (Nvidia terminology)
- **Parallel construct**: creates multiple threads in the teams, each which can run concurrently



GPU OpenMP parallelism

Creates teams of threads in the target device

Distributes iterations to the threads

```
#pragma omp target teams distribute parallel for private(x) reduction(+:sum)
 for( int i=0; i<=num_steps; i++){
   x = (i+0.5)*step;
   sum = sum + 4.0/(1.0+x*x);
  }
```

OpenMP and data transfer

```
...

#pragma omp target teams distribute parallel for map(tofrom:a[0:num], b[0:num])
  for (size_t j=0; j<num; j++) {
    a[j] = a[j]+scalar*b[j];

  }
...
```

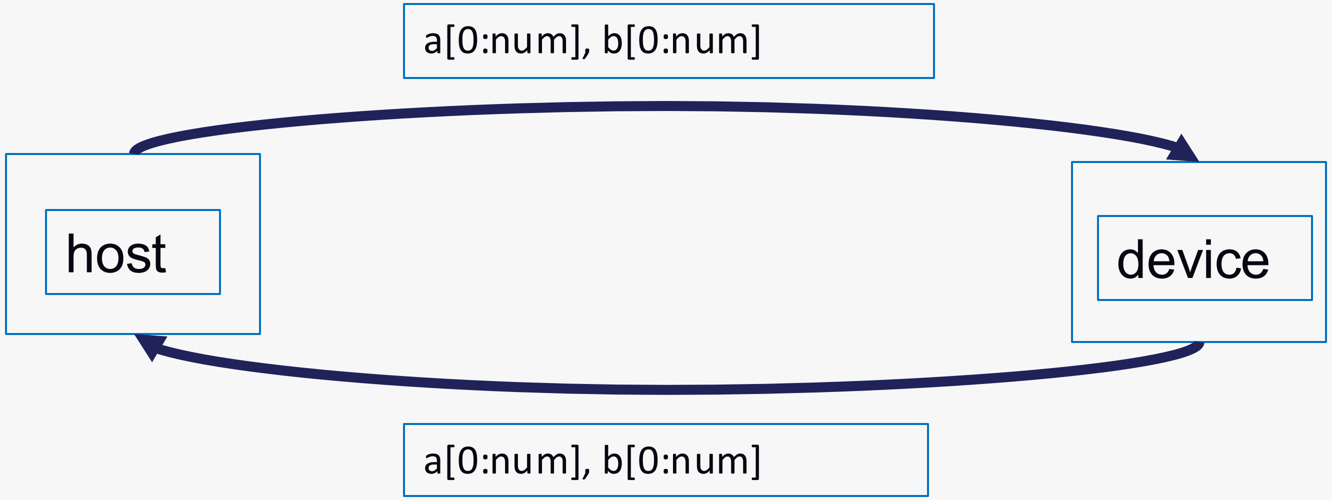
OpenMP and data transfer

```
…

#pragma omp target teams distribute parallel for map(tofrom:a[0:num], b[0:num])
    for (size_t j=0; j<num; j++) {
      a[j] = a[j]+scalar*b[j];

    }
…

…
```



- Maps *a* and *b* to and from the device.
- These are shared and accessible by all of the threads on the GPU.

OpenMP offload compilers and flags on Aurora

Language	MPI Wrapper Compiler (Underlying Compiler)	Flag to Turn on OpenMP Support and Target CPU Threads	Additional Flags to Target GPU Devices
Fortran	mpifort (ifx)	-fiopenmp	-fopenmp-targets=spir64
C	mpicc (icx)	-fiopenmp	-fopenmp-targets=spir64
C++	mpicxx (icpx)	-fiopenmp	-fopenmp-targets=spir64

- Intel OpenMP offload compilers are in the default environment on Aurora
- You can swap "-fopenmp-targets=spir64" for "-fopenmp-targets=spir64_gen -Xopenmp-target-backend "-device pvc" " for AOT compilation
- https://docs.alcf.anl.gov/aurora/programming-models/openmp-aurora/

OpenMP on Aurora: Functionality Benchmarks

- OpenMP vs. Offload: https://github.com/TApplencourt/OvO
- OpenMP Validation and Verification: https://crpl.cis.udel.edu/ompvv/project/
- Some of the tests are for uncommon OpenMP directives, but it gives a general sense that both implementations are generally passing
- (Part of an upcoming submission to IWOMP)

	C/C++		Fortran	
	Intel (2025.0) on Aurora	Nvidia (23.9) on Polaris	Intel (2025.0) on Aurora	Nvidia (23.9) on Polaris
OvO	100% (521/521)	70% (367/521)	100% (389/389)	92% (359/389)
OMPVV-4.5	96% (141/147)	89% (131/147)	95% (99/104)	93% (97/104)
OMPVV-5.0	77% (164/213)	35%(75/213)	66% (85/128)	27% (35/128)
OMPVV-5.1	75% (76/101)	12%(12/101)	60% (17/28)	7% (2/28)
OMPVV-5.2	13% (3/24)	25%(6/24)	80% (4/5)	60% (3/5)
OMPVV-6.0	22% (5/22)	4%(1/22)	0% (0/1)	0% (0/1)

OpenMP vs. OpenACC

- OpenACC is not supported on Intel GPUs
- However, a lot of concepts are shared with OpenMP Offload, so OpenMP Offload can usually be a replacement
- There is an Intel-provided migration tool for OpenACC to OpenMP
  - https://github.com/intel/intel-application-migration-tool-for-openacc-to-openmp

Quickstart

```
$ cp -r /lus/flare/projects/gpu_hack/openmp .

$ cd openmp

$ mpicxx -fiopenmp -fopenmp-targets=spir64 -o c_test hello.cpp

$ mpiexec -n 1 ./c_test
Number of devices: 6
Hello world from accelerator.

$ mpiexec -n 1 gpu_tile_compact.sh ./c_test
Number of devices: 1
Hello world from accelerator.
```