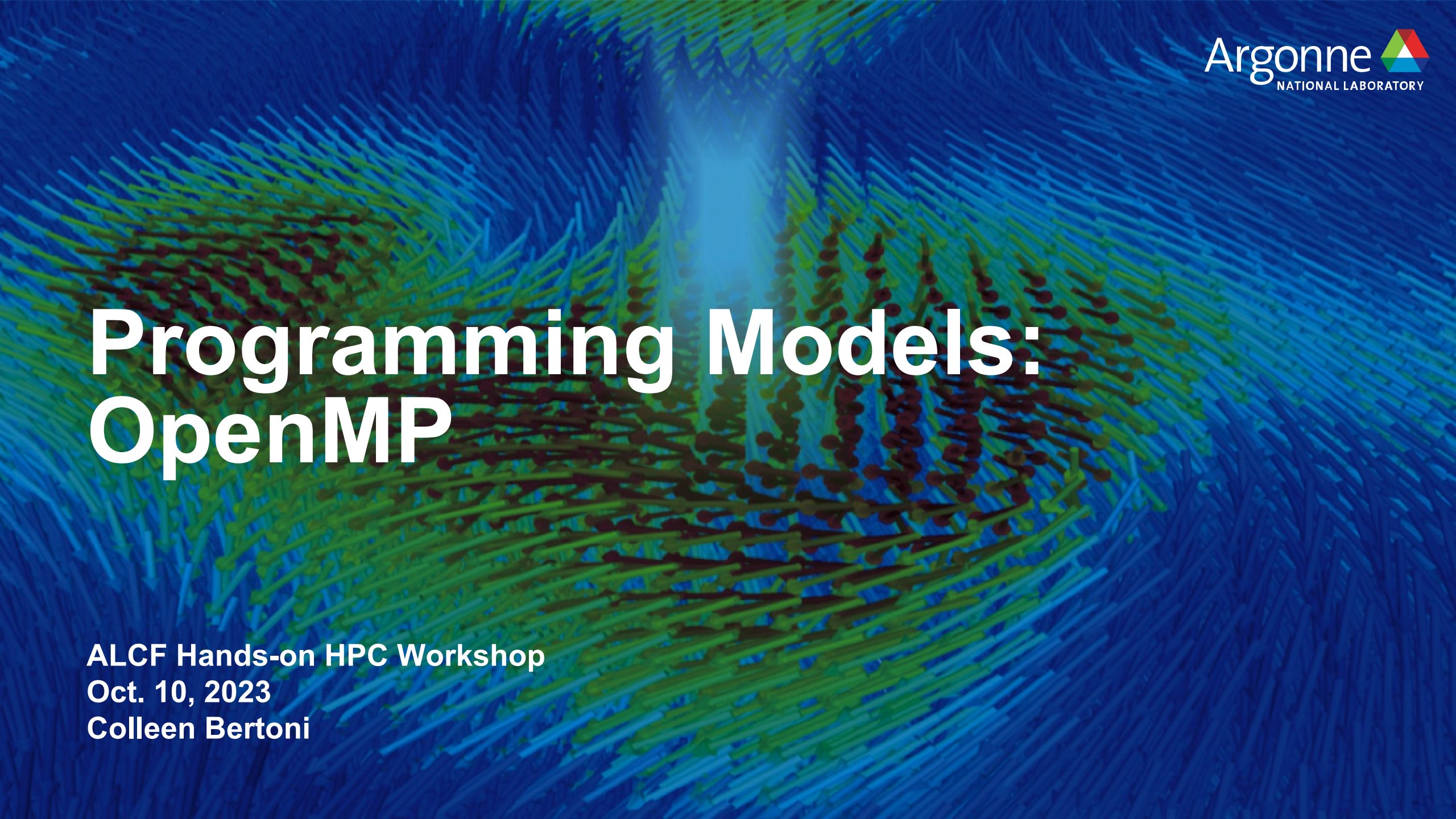

October 10-12, 2023
ALCF Hands-on HPC Workshop



Programming Models: OpenMP

ALCF Hands-on HPC Workshop
Oct. 10, 2023
Colleen Bertoni

Why OpenMP?

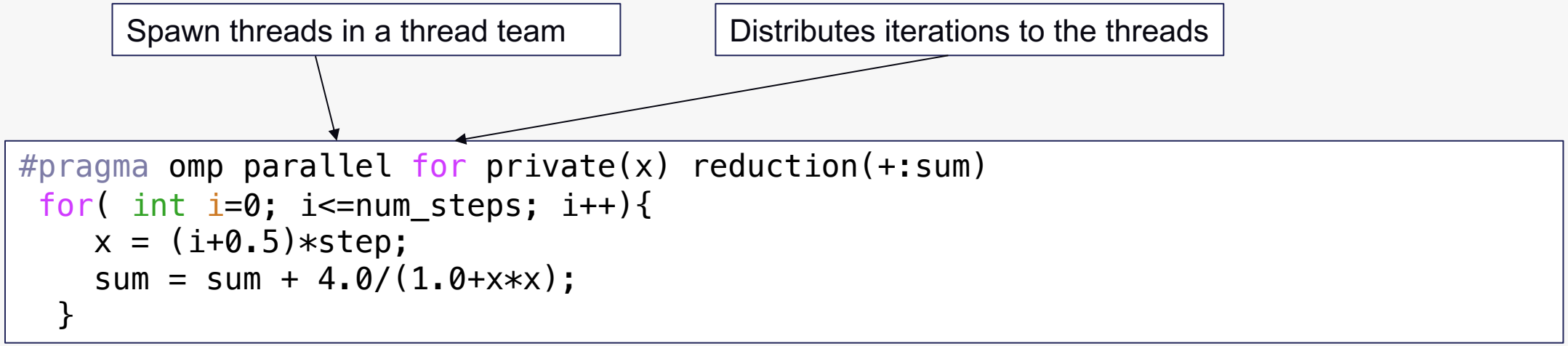
- Open standard for parallel programming with support across vendors
  - API and environment variables
  - Specification document and examples: http://www.openmp.org
  - Broad and expressive
    - OpenMP runs on CPU threads, GPUs, SIMD units
  - C/C++ and Fortran
  - Supported by Intel, HPE, AMD, GNU, LLVM compilers and others
  - OpenMP offload is supported on Aurora, Frontier, Perlmutter
    - Portable across large DOE systems
- For Polaris: Why instead of CUDA?
  - Easy to get started and trivial to parallelize loops
  - The reduction clause simplifies data reduction

OpenMP Compiler Support for GPUs Across Hardware and Vendors

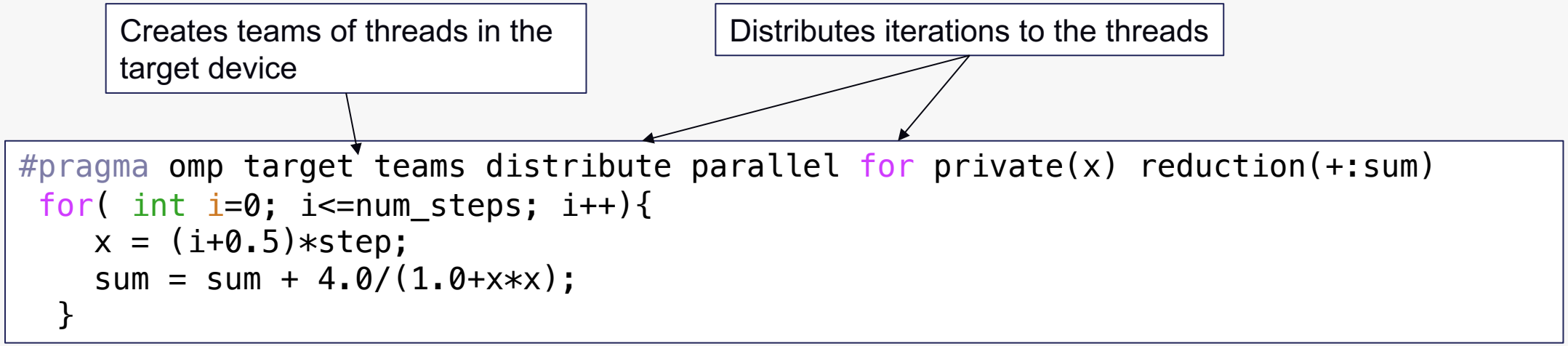
GPU	Vendor	Compiler	flags
Nvidia	LLVM	clang++	-fopenmp -fopenmp-targets=nvptx64-nvidia-cuda
	HPE	CC/ftn	-fopenmp -fopenmp-targets=nvptx64/-h omp
	Nvidia	nvc++/ nvfortran	-mp=gpu -gpu=cc80
	IBM	xlC_r/xlf90_r	-qsmp=omp -qoffload
	GNU	g++/gfortran	-fopenmp -foffload=-lm
Intel	Intel	icpx/ifx	-fiopenmp -fopenmp-targets=spir64
AMD	AMD	clang++/flang	-fopenmp -fopenmp-targets=amdgcn-amd-amdhsa -Xopenmp-target=amdgcn-amd-amdhsa
	GNU	g++/gfortran	-fopenmp -foffload=-lm
	HPE	CC/ftn	-fopenmp/-homp

Generally about CPU and GPU compilers:
https://www.openmp.org/resources/openmp-compilers-tools/

CPU OpenMP parallelism



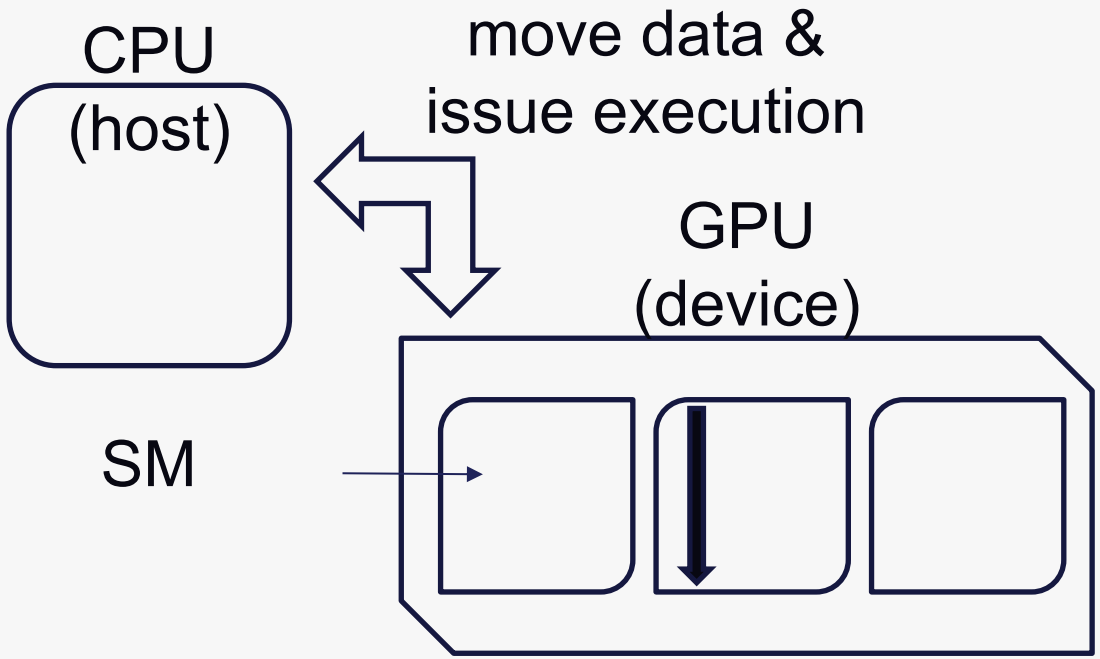
GPU OpenMP parallelism



OpenMP Offload: Steps

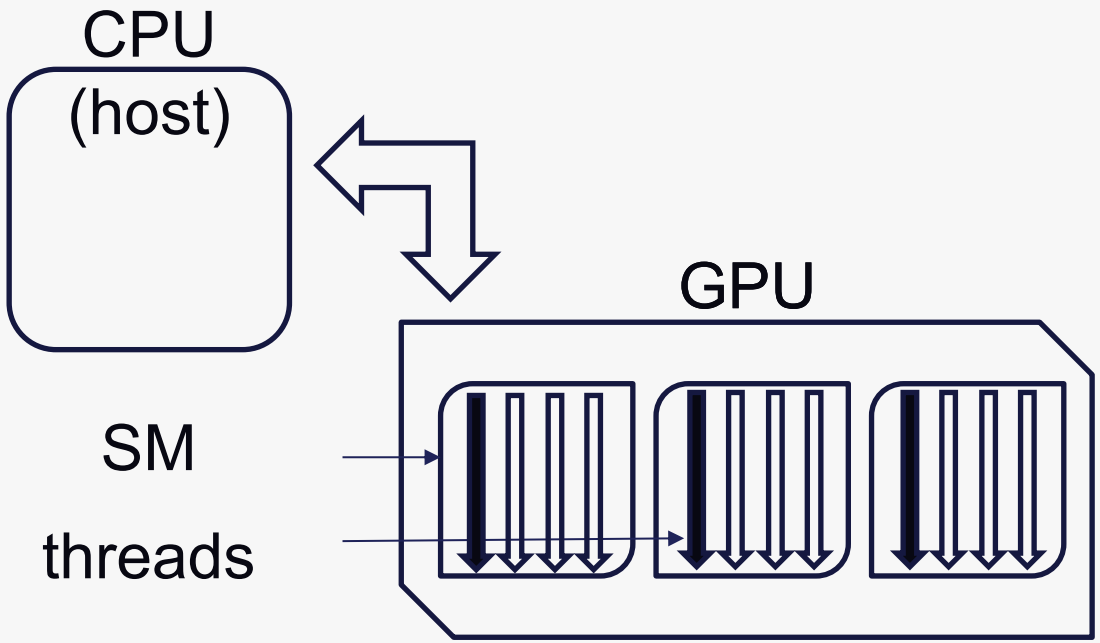
- Basic offloading mechanisms
  - Offloading code to the device
  - Expressing parallelism
  - Mapping data

How to use OpenMP – Execution Mapping



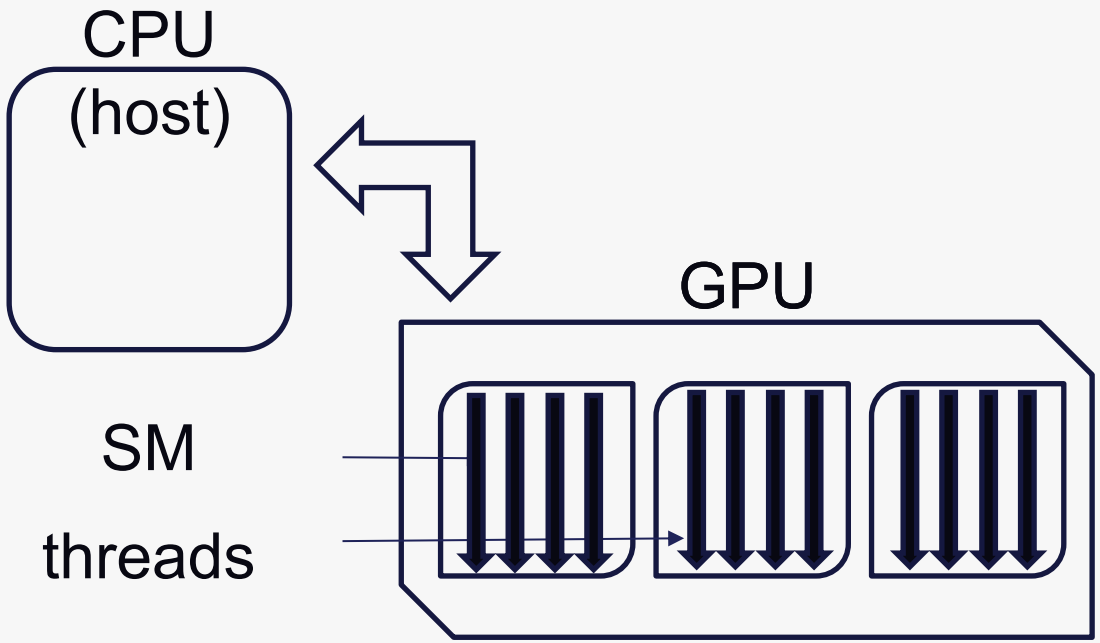
- The **target** construct offloads the enclosed code to the accelerator: single thread on a device (GPU)

How to use OpenMP – Execution Mapping



- The **target** construct offloads the enclosed code to the accelerator: single thread on a device (GPU)
- The **teams** construct creates a league of teams: one thread each, concurrent execution (on SMs)

How to use OpenMP – Execution Mapping



- The **target** construct offloads the enclosed code to the accelerator: single thread on a device (GPU)
- The **teams** construct creates a league of teams: one thread each, concurrent execution (on SMs)
- The **parallel** construct creates threads in each team: parallel execution (by hardware threads)

How to use OpenMP – Execution Mapping

```
#pragma omp target
#pragma omp teams distribute
for (int i=0; i<N; ++i) {
#pragma omp parallel for
  for (int j=0; j<N; ++j) {
    x[j+N*i] *= 2.0;
  }
}
```

- The **target** construct offloads the enclosed code to the accelerator
- The **teams** construct creates a league of teams
- The **distribute** construct distributes the outer loop iterations between the league of teams
- The **parallel for** combined construct creates a thread team for each team and distributes the inner loop iterations to threads

How to use OpenMP – Data Mapping

```
#pragma omp target map(tofrom:x[0:M])
#pragma omp teams distribute
for (int i=0; i<N; ++i) {
#pragma omp parallel for
  for (int j=0; j<N; ++j) {
    x[j+N*i] *= 2.0;
  }
}
```

- The **target** construct offloads the enclosed code to the accelerator
- The **teams** construct creates a league of teams
- The **distribute** construct distributes the outer loop iterations between the league of teams
- The **parallel for** combined construct creates a thread team for each team and distributes the inner loop iterations to threads
- The **map** construct maps data for a single target region

How to use OpenMP – Working with GPU libraries

```
cublasHandle_t handle;
if(cublasCreate(&handle) != CUBLAS_STATUS_SUCCESS){
    exit(EXIT_FAILURE);
}

#pragma omp target enter data \
map(to:aa[0:N*N],bb[0:N*N],cc_gpu[0:N*N])

#pragma omp target data use_device_ptr(aa,bb,cc_gpu)
 {
int cublas_error = cublasDgemm(handle,CUBLAS_OP_N,
CUBLAS_OP_N,size, size, size, &alpha, aa, size, bb,
size, &beta, cc_gpu, size);
}

cudaDeviceSynchronize();
cublasDestroy(handle);
```

- Specific to the vendor
- For Nvidia, you can call the same GPU libraries as in pure CUDA
- You can allocate memory with OpenMP as usual
- The **use_device_ptr** clause tells OpenMP to use the corresponding device address in the data region so it can pass the device pointer to cuBLAS

OpenMP offload compilers and flags on Polaris

module	compiler	flags
PrgEnv-nvhpc	cc/CC/ftn (nvc/nvc++/nvfortran)	-mp=gpu -gpu=cc80
llvm	mpicc/mpicxx (clang/clang++)	-fopenmp -fopenmp-targets=nvptx64-nvidia-cuda
PrgEnv-gnu	cc/CC/ftn (gcc/g++/gfortran)	-fopenmp
PrgEnv-cray	cc/CC/ftn	-fopenmp

- Nvidia compilers are in the default environment on Polaris
- LLVM and Nvidia compilers are recommended
- https://www.alcf.anl.gov/support/user-guides/polaris/programming-models/openmp-polaris/index.html

OpenMP Offload: Hands-on

- 1:30 - 4:00 pm in Room 1404
- Agenda:
- Quickstart/Reminder for OpenMP offload on Polaris
  - Setting the environment
  - Building on Polaris
  - Running on Polaris
- 101 Demo for GPUs
- Multi-GPU runs: Affinity and binding to CPUs and GPUs on Polaris
- Hands-on Example
- Debugging
- Q&A / Open work

Questions?

Backup

OpenMP and the loop directive

- Added in OpenMP 5.0
- Similar to “distribute” and “for”, it workshares loop iterations
- It also asserts that loop iterations can be run in any order (are independent)
- Can provide a performance advantage (specifically with the Nvidia compiler, which supports it well)

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

      }
```

```
#pragma omp target teams loop
     for (size_t j=0; j<num; j++) {
        a[j] = a[j]+scalar*b[j];

      }
```

How to use OpenMP – Execution Mapping

- League of teams
  - Runs across SMs, global memory
- One team of threads
  - Runs in one SM, shared memory
- One thread
  - Runs on a cuda core in an SM, local memory and registers

CUDA	DPC++	OpenMP
CUDA thread	Work-item	OpenMP thread
Warp	Sub-group	--
Thread block	Work-group	team