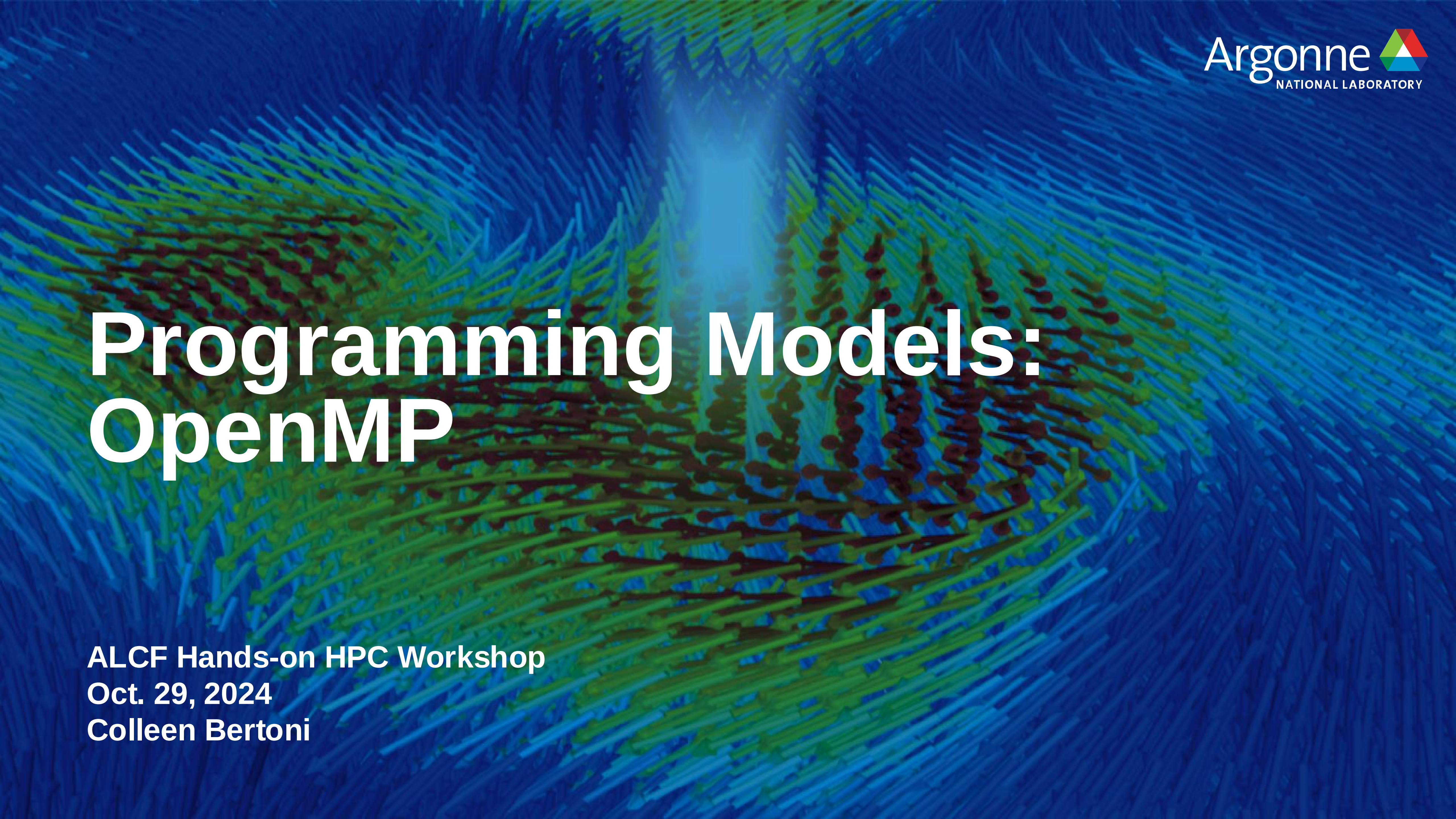
October 29-31, 2024

ALCF Hands-on HPC Workshop



Argonne NATIONAL LABORATORY

Programming Models: OpenMP

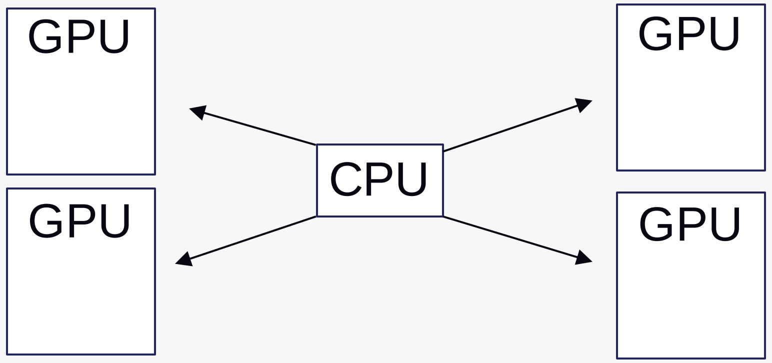
ALCF Hands-on HPC Workshop
Oct. 29, 2024
Colleen Bertoni

Motivation

Maximum speed at which you can compute is bound by

$$\text{(clock rate of cores) * (} \underbrace{\text{number of cores) * (number of operations each core can do per cycle}}_{\boxed{\text{Amount of parallelism you can exploit}}} \text{)}$$

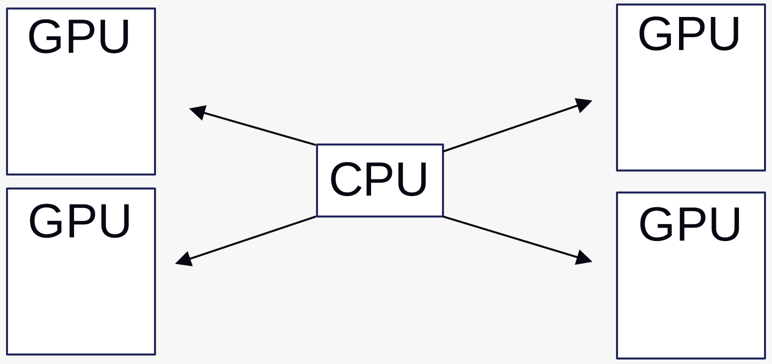
Motivation



Maximum speed at which you can compute is bound by

$$(\text{ clock rate of cores }) * \underbrace{(\text{ number of cores }) * (\text{ number of operations each core can do per cycle })}_{\text{Amount of parallelism you can exploit}}$$

Motivation



Maximum speed at which you can compute is bound by

$$(\text{ clock rate of cores }) * \underbrace{(\text{ number of cores }) * (\text{ number of operations each core can do per cycle })}_{\boxed{\text{Amount of parallelism you can exploit}}}$$

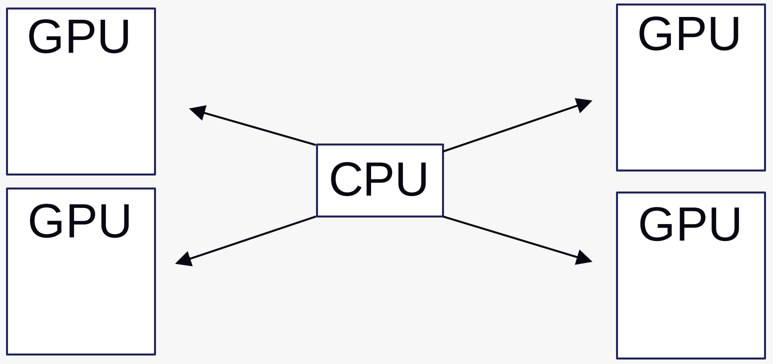
For the GPUs on a Polaris node:

39 Tflops ~= (1.41 *10^9 cycles/second)* (108 SMs/ GPU) * (4 GPUs) * (32 FPU inst/ (SM*cycle))* 2 (FMA factor)

For the CPUs on a Polaris node:

1 Tflops ~= (3.7 *10^9 cycles/second)* (32 cores) * (1 CPUs) * 4 (SIMD width) * 2 (FMA factor)

Motivation



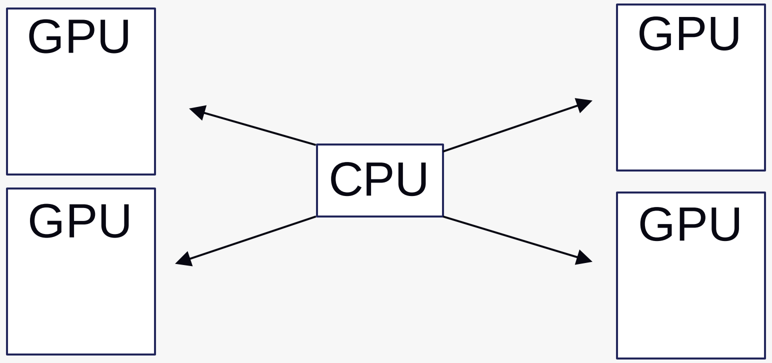
Maximum speed at which you can compute is bound by

$$(\text{clock rate of cores}) * \underbrace{(\text{number of cores}) * (\text{number of operations each core can do per cycle})}_{\text{Amount of parallelism you can exploit}}$$

1. Far more computational power on GPUs than CPUs

For the GPUs on a Polaris node:
39 Tflops ~= (1.41 *10^9 cycles/second)* (108 SMs/ GPU) * (4 GPUs) * (32 FPU inst/ (SM*cycle))* 2 (FMA factor)
For the CPUs on a Polaris node:
1 Tflops ~= (3.7 *10^9 cycles/second)* (32 cores) * (1 CPUs) * 4 (SIMD width) * 2 (FMA factor)

Motivation



Maximum speed at which you can compute is bound by

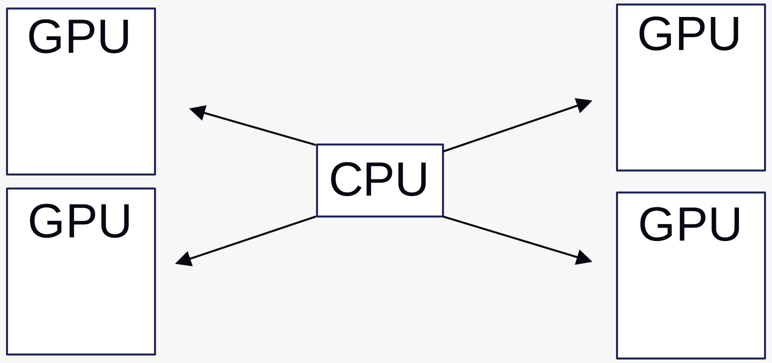
$$(\text{clock rate of cores}) * \underbrace{(\text{number of cores}) * (\text{number of operations each core can do per cycle})}_{\text{Amount of parallelism you can exploit}}$$

1. Far more computational power on GPUs than CPUs

2. A lot of resources, need to be careful to target them properly

For the GPUs on a Polaris node:
39 Tflops ~= (1.41 *10^9 cycles/second)* (108 SMs/ GPU) * (4 GPUs) * (32 FPU inst/ (SM*cycle))* 2 (FMA factor)
For the CPUs on a Polaris node:
1 Tflops ~= (3.7 *10^9 cycles/second)* (32 cores) * (1 CPUs) * 4 (SIMD width) * 2 (FMA factor)

Agenda: how OpenMP can help you…



Maximum speed at which you can compute is bound by

$$(\text{ clock rate of cores }) * \underbrace{(\text{ number of cores }) * (\text{ number of operations each core can do per cycle })}_{\text{Amount of parallelism you can exploit}}$$

1. Effectively use the computational power on GPUs

2. Using affinity settings to target CPUs and GPUs in the manner you want to

For the GPUs on a Polaris node:

39 Tflops ~= (1.41 *10^9 cycles/second)* (108 SMs/ GPU) * (4 GPUs) * (32 FPU inst/ (SM*cycle))* 2 (FMA factor)

For the CPUs on a Polaris node:

1 Tflops ~= (3.7 *10^9 cycles/second)* (32 cores) * (1 CPUs) * 4 (SIMD width) * 2 (FMA factor)

Overview of this talk

- Using OpenMP on Polaris (~30 min)
  - Why OpenMP?
  - Quick Start for Running on Polaris
  - Using GPUs with OpenMP Offload
  - Multi-GPU runs: Affinity and binding to CPUs and GPUs on Polaris
- Demo of OpenMP (~20 min)
  - OpenMP 101 and basics on Polaris

```
$ git clone https://github.com/argonne-lcf/ALCF_Hands_on_HPC_Workshop
$ cd ALCF_Hands_on_HPC_Workshop/programmingModels/OpenMP/demo
```

Using OpenMP on Polaris

- Why OpenMP?
- Quick Start for Running on Polaris
- Using GPUs with OpenMP Offload
- Multi-GPU runs: Affinity and binding to CPUs and GPUs on Polaris

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/

• Quick Start for Running on Polaris

Setting the environment to use OpenMP on Polaris

module	OpenMP CPU support?	OpenMP GPU support?
PrgEnv-nvhpc	yes	yes
llvm	yes	yes
PrgEnv-gnu	yes	no
PrgEnv-cray	yes	yes*

https://docs.alcf.anl.gov/polaris/programming-models/openmp-polaris/

Setting the environment to use OpenMP on Polaris

module	OpenMP CPU support?	OpenMP GPU support?
PrgEnv-nvhpc	**yes**	**yes**
llvm	yes	yes
PrgEnv-gnu	yes	no
PrgEnv-cray	yes	yes*

- PrgEnv-nvhpc
  - Loaded by default

https://docs.alcf.anl.gov/polaris/programming-models/openmp-polaris/

Setting the environment to use OpenMP on Polaris

module	OpenMP CPU support?	OpenMP GPU support?
PrgEnv-nvhpc	yes	yes
llvm	**yes**	**yes**
PrgEnv-gnu	yes	no
PrgEnv-cray	yes	yes*

- module use /soft/modulefiles
- module load mpiwrappers/cray-mpich-llvm
- module load cudatoolkit-standalone

- https://docs.alcf.anl.gov/polaris/compiling-and-linking/llvm-compilers-polaris/

https://docs.alcf.anl.gov/polaris/programming-models/openmp-polaris/

Setting the environment to use OpenMP on Polaris

module	OpenMP CPU support?	OpenMP GPU support?
PrgEnv-nvhpc	yes	yes
llvm	yes	yes
PrgEnv-gnu	**yes**	**no**
PrgEnv-cray	yes	yes*

- module switch PrgEnv-nvhpc PrgEnv-gnu

https://docs.alcf.anl.gov/polaris/programming-models/openmp-polaris/

Setting the environment to use OpenMP on Polaris

module	OpenMP CPU support?	OpenMP GPU support?
PrgEnv-nvhpc	yes	yes
llvm	yes	yes
PrgEnv-gnu	yes	no
PrgEnv-cray	**yes**	**yes***

- module switch PrgEnv-nvhpc PrgEnv-cray
  - (if you want to try offloading, “module load cudatoolkit-standalone”, but it’s not recommended)

https://docs.alcf.anl.gov/polaris/programming-models/openmp-polaris/

Building 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

https://docs.alcf.anl.gov/polaris/programming-models/openmp-polaris/

Running on Polaris

```
$ cat submit.sh
#!/bin/sh
#PBS -l select=1:system=polaris
#PBS -l walltime=0:30:00
#PBS -q HandsOnHPC
#PBS -A alcf_training
#PBS -l filesystems=home:eagle:grand

cd ${PBS_O_WORKDIR}
mpiexec -n 1 ./executable

$ # submit to the queue:
$ qsub -l select=1:system=polaris -l walltime=0:30:00 -l filesystems=home:eagle:grand -q HandsOnHPC -A
alcf_training ./submit.sh
```

https://docs.alcf.anl.gov/polaris/programming-models/openmp-polaris/

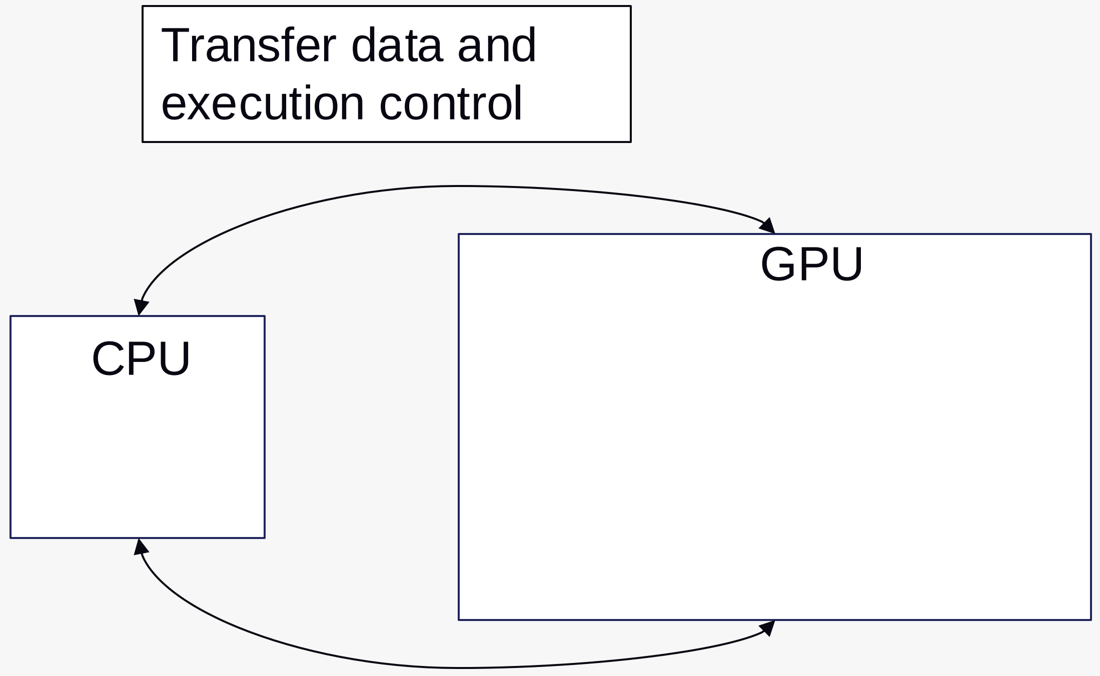
- **Using GPUs with OpenMP Offload**

OpenMP Offload Introduction

The goal is to introduce important basic topics for OpenMP offloading

We will cover three basic offloading topics:

1. Offloading code to the device and getting device info
2. Expressing parallelism
3. Mapping data



Compiler support for offloading

- GCC
- LLVM
- IBM XL
- HPE
- NVIDIA
- Intel
- AMD

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

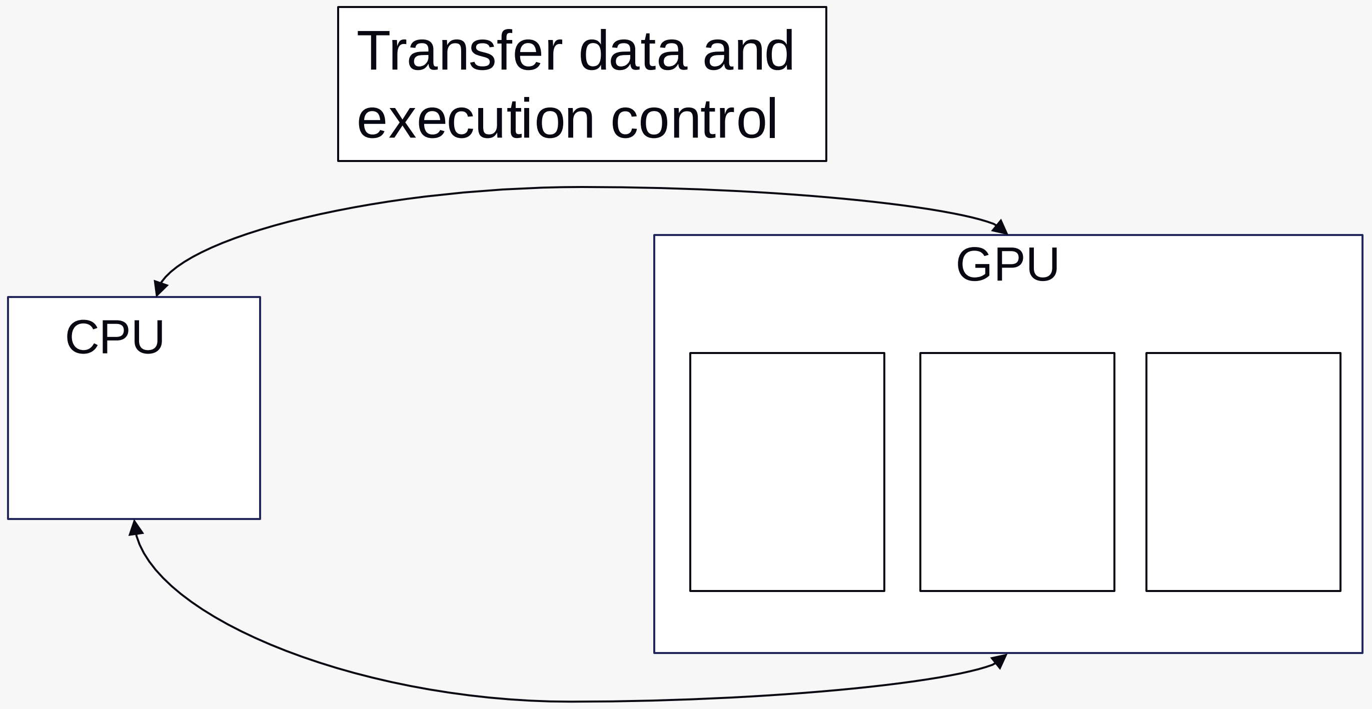
Creates teams of threads in the target device (CUDA: grid of thread blocks)

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);
  }
```

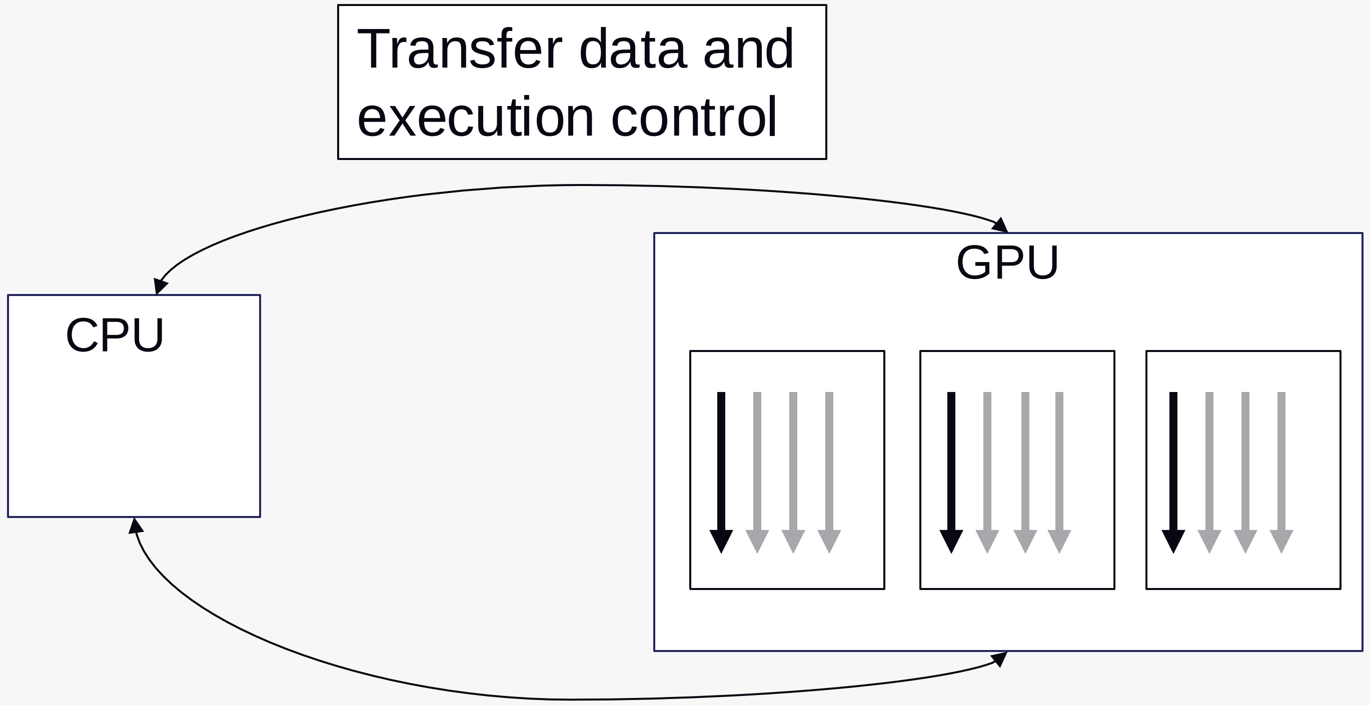
OpenMP Offload Introduction

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



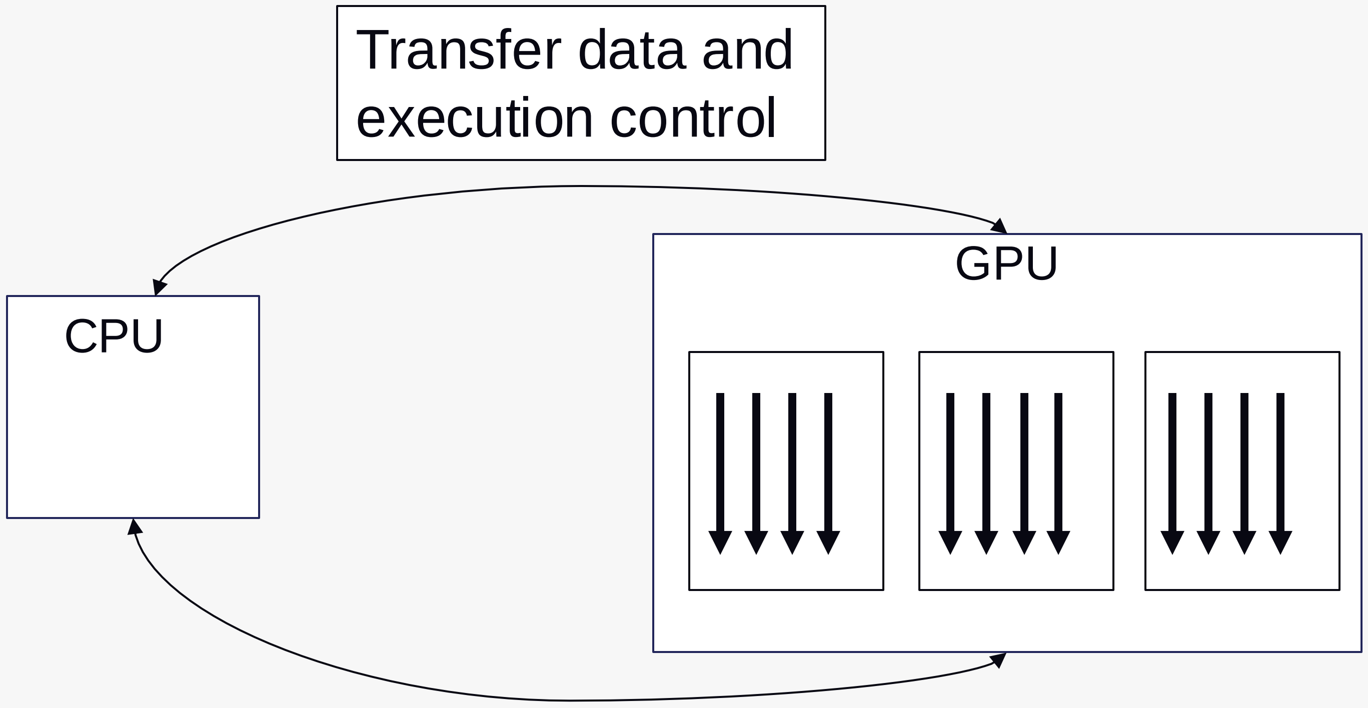
OpenMP Offload Introduction

- **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 (CUDA: thread block)



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 (CUDA: thread block)
- **Parallel construct**: creates multiple threads in the teams, each which can run concurrently (CUDA: thread)



GPU OpenMP parallelism

Distributes iterations to the threads, where each thread uses SIMD parallelism

```
...

#pragma omp target teams distribute parallel for simd map(v1[0:N],p[0:N])
 for (i=0; i<N; i++)
 {
   p[i] = v1[i];
 }

...
```

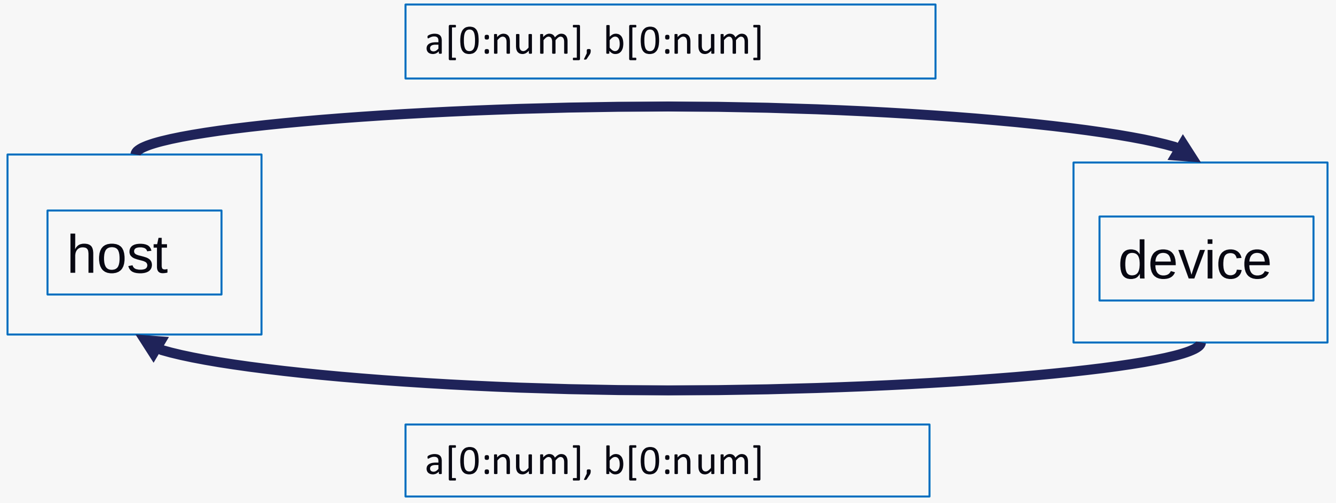
Controlling data transfer

Creates teams of threads in the target device

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



...

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

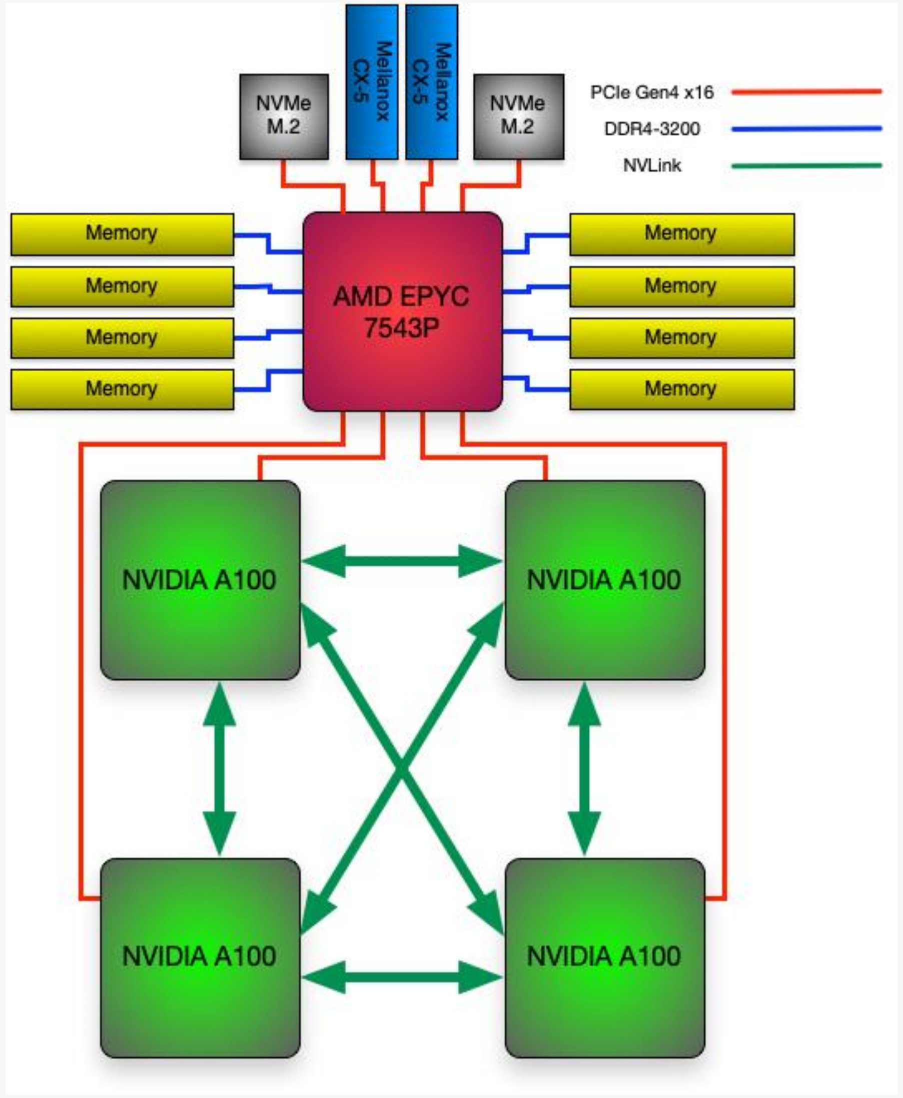
Multi-GPU runs: Affinity and binding to CPUs and GPUs

ALCF Hands-on HPC Workshop
Oct. 29, 2024
Colleen Bertoni

Motivation: Managing Computational Resources

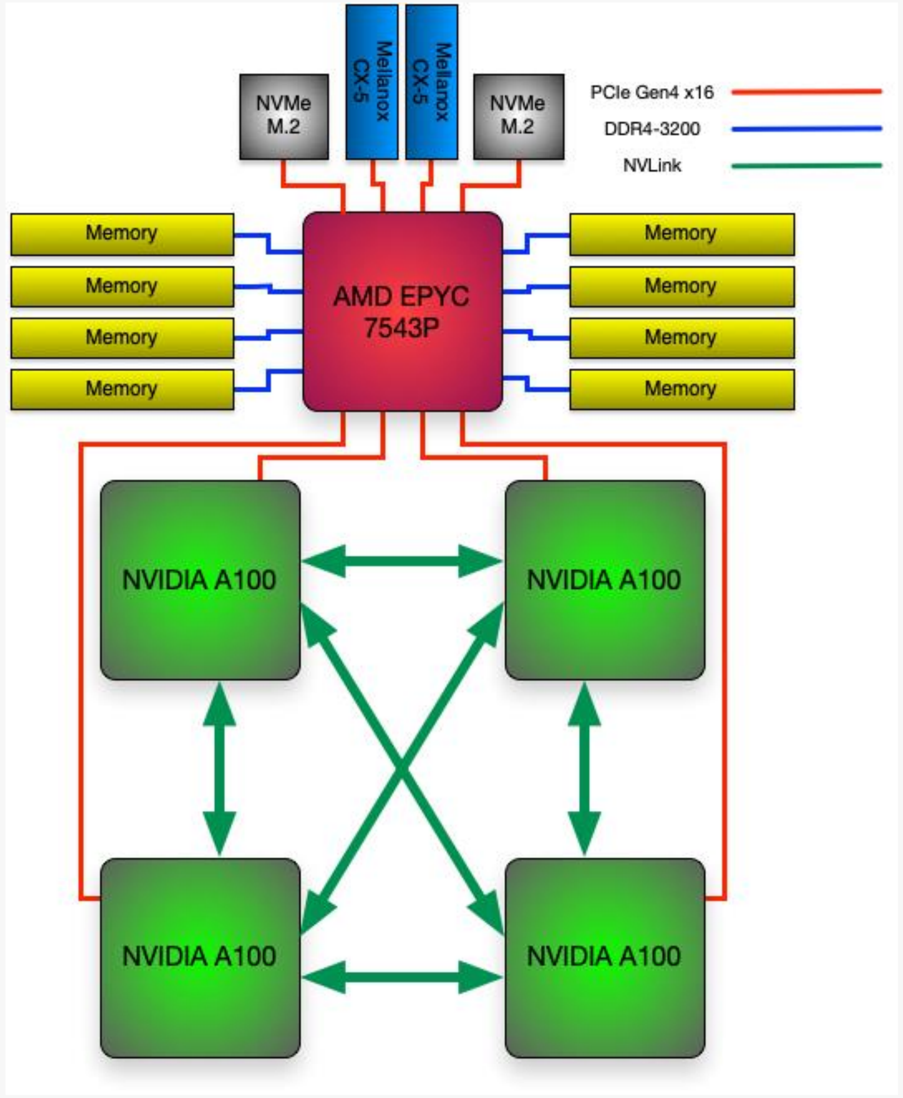
- Each Polaris node has
  - 1 CPU (AMD EPYC) with 32 cores (64 HW threads)
  - 4 A100 GPUs

- How to efficiently assign work (MPI ranks and OpenMP threads) to the GPUs and CPU cores?



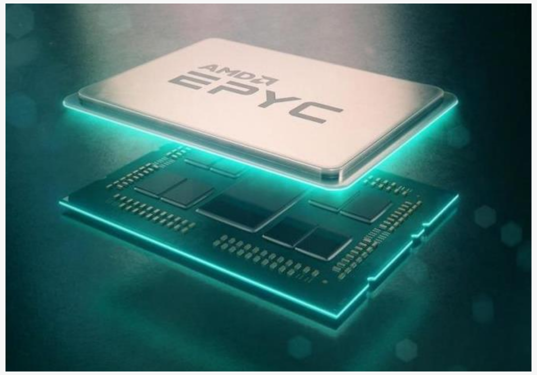
Affinity: Keeping Work Close to Data

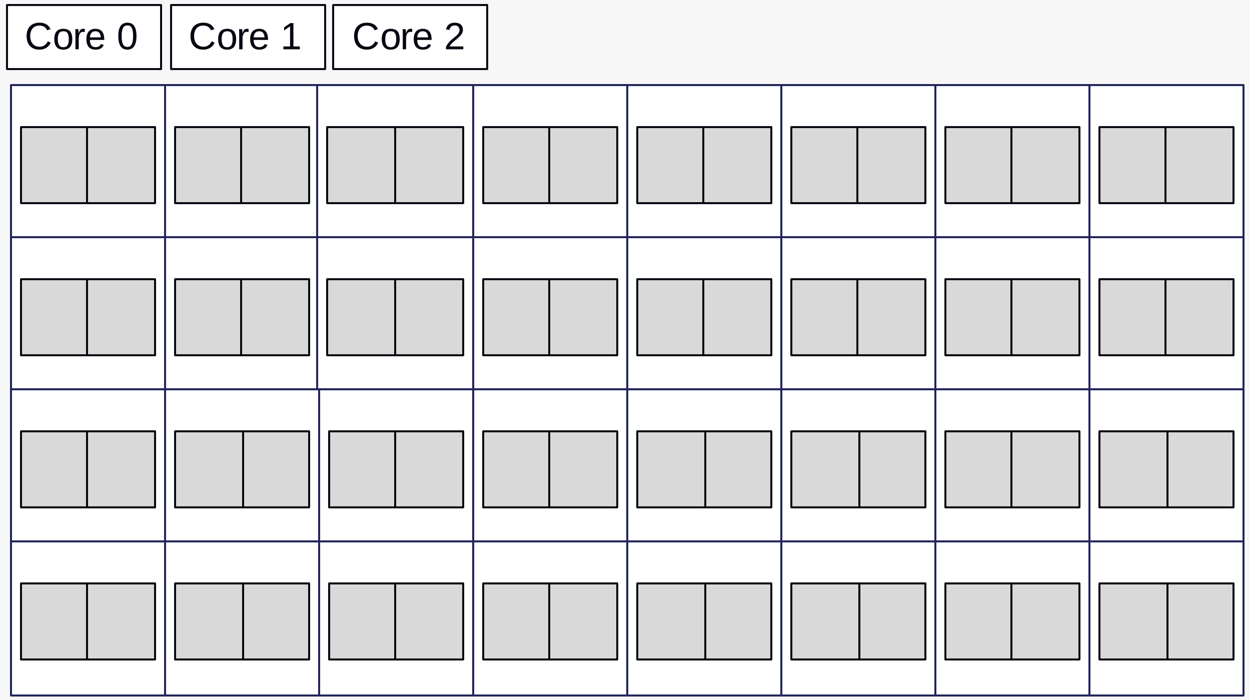
- **Affinity** is the binding or mapping of processes or threads to a specific HW resource, like a physical core or GPU
- Why is this important?
  - It prevents threads from moving between physical cores, which can cause lower performance
    - Ex: If a thread allocated data in the L1 cache of one physical core and then moved to a new core, data will have to move into a new L1 cache
- **Goal**
  - To keep the threads executing close to the data they access frequently



CPU Affinity

MPI ranks and OpenMP threads



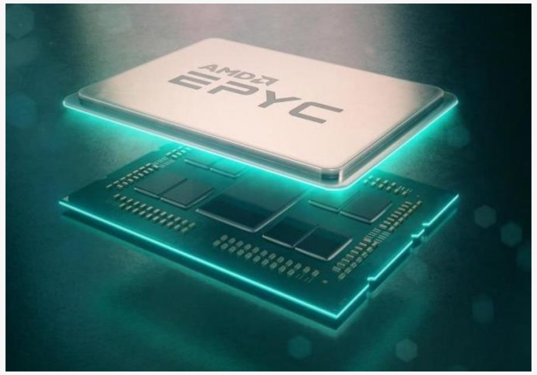


- Each AMD EYPC has 32 Zen3 cores and 64 HW threads (2 per physical core)
- When a parallel job is begun, the job must have some way of mapping specific ranks and threads to each of the 64 HW threads.

CPU Affinity

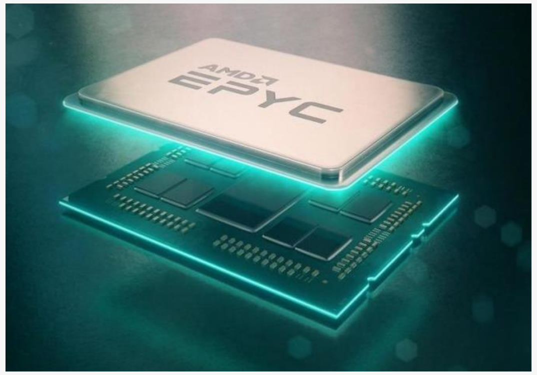
MPI ranks and OpenMP threads

Core 0	Core 1	Core 2					
0 32	1 33	2 34	3 35	4 36	5 37	6 38	7 39
8 40	9 41	10 42	11 43	12 44	13 45	14 46	15 47
16 48	17 49	18 50	19 51	20 52	21 53	22 54	23 55
24 56	25 57	26 58	27 59	28 60	29 61	30 62	31 63

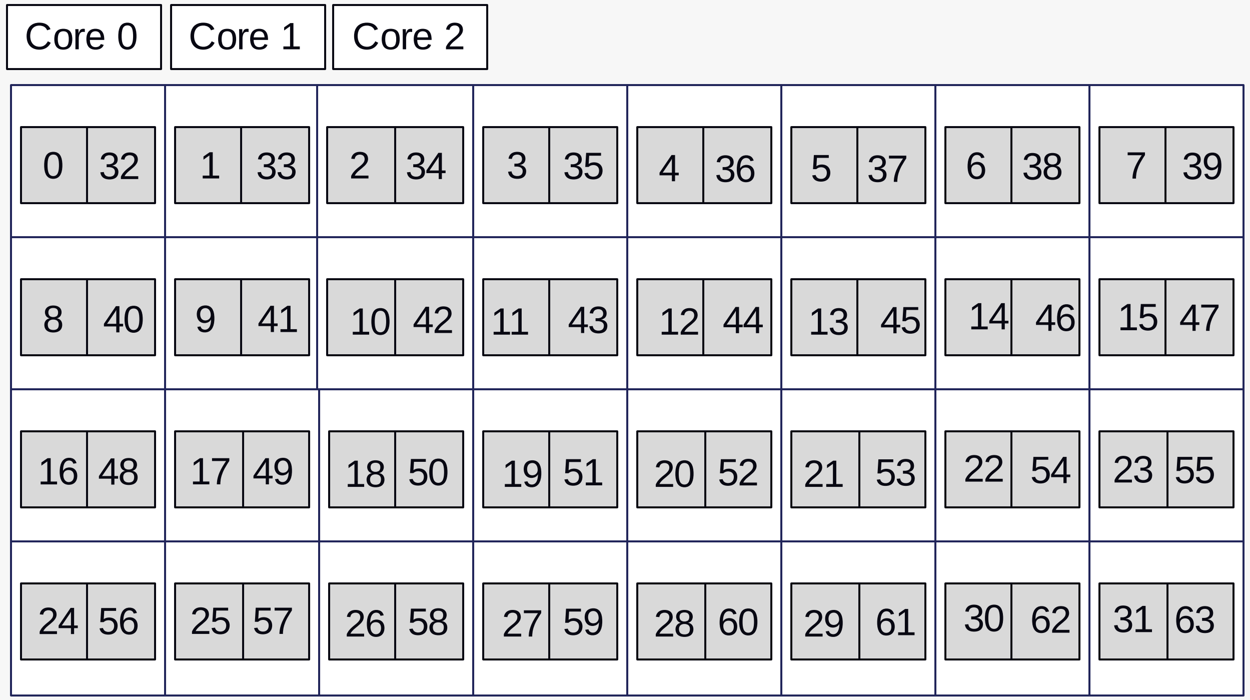


- Each AMD EYPC has 32 Zen3 cores and 64 HW threads (2 per physical core)
- When a parallel job is begun, the job must have some way of mapping specific ranks and threads to each of the 64 HW threads.
- Mapping is typically done by an affinity mask, whic assigns HW threads to each MPI rank or thread to use

CPU Affinity

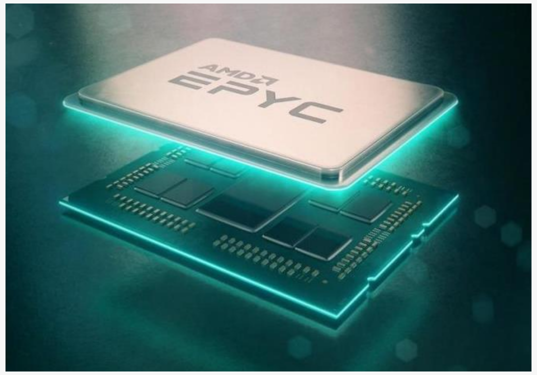


MPI ranks and OpenMP threads

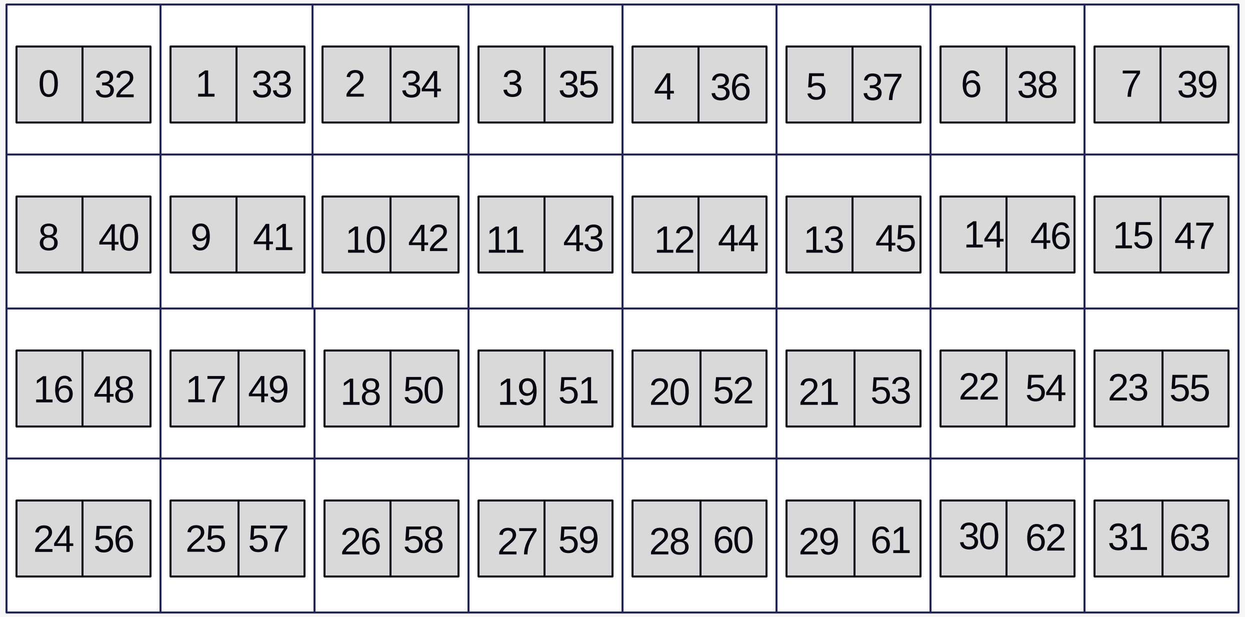


- Using the --cpu-bind and --depth commands to mpiexec and OpenMP environment variables, we can assign MPI ranks and OpenMP threads to specific HW threads

CPU Affinity: Example

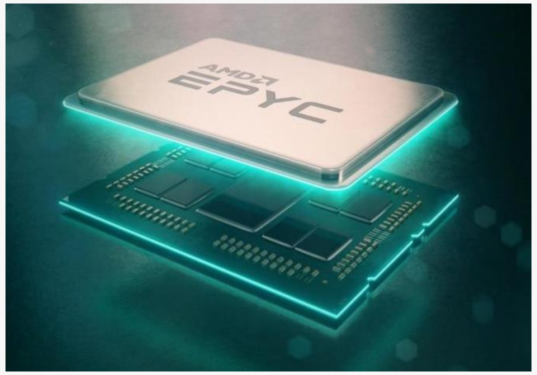


```
export OMP_NUM_THREADS=8
export OMP_PLACES=threads
mpiexec -n 8 --ppn 4 --depth=$OMP_NUM_THREADS \
                  --cpu-bind depth ./hello_affinity
```

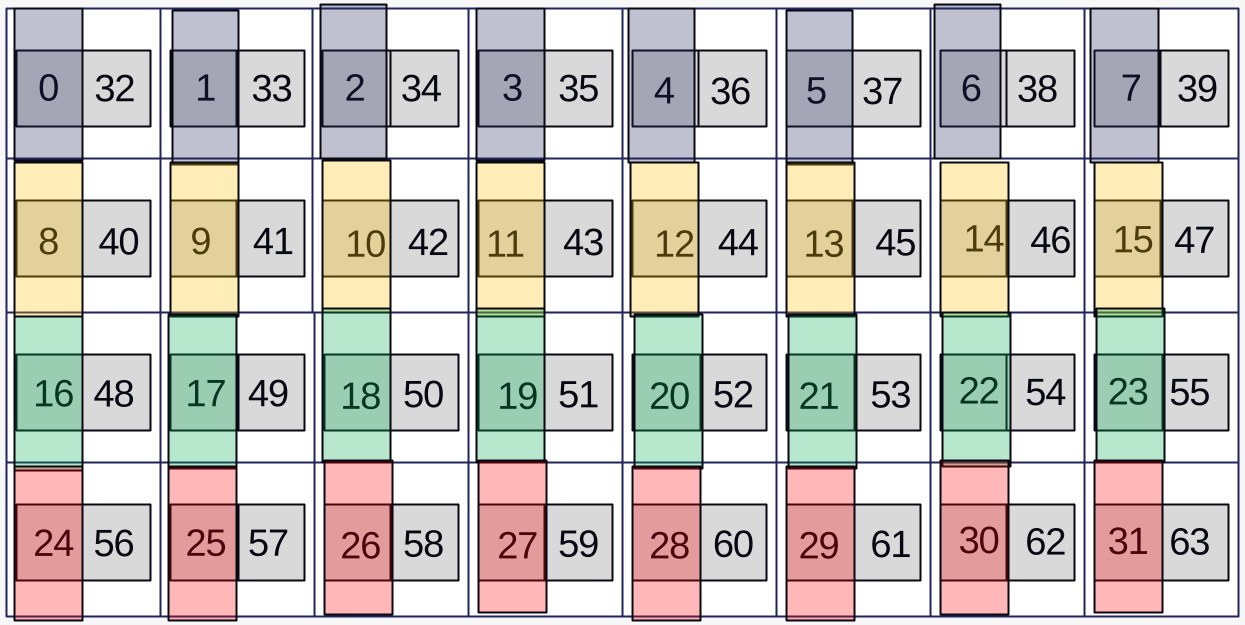


- Runs on 2 nodes with 4 ranks per node and 8 OpenMP threads per rank
- The depth clause spaces out the rank ranks by $OMP_NUM_THREADS
- The “--cpu-bind depth ” binds ranks in a compact round-robin manner
- OMP_PLACES=threads defines that threads can only execute on threads

CPU Affinity: Example



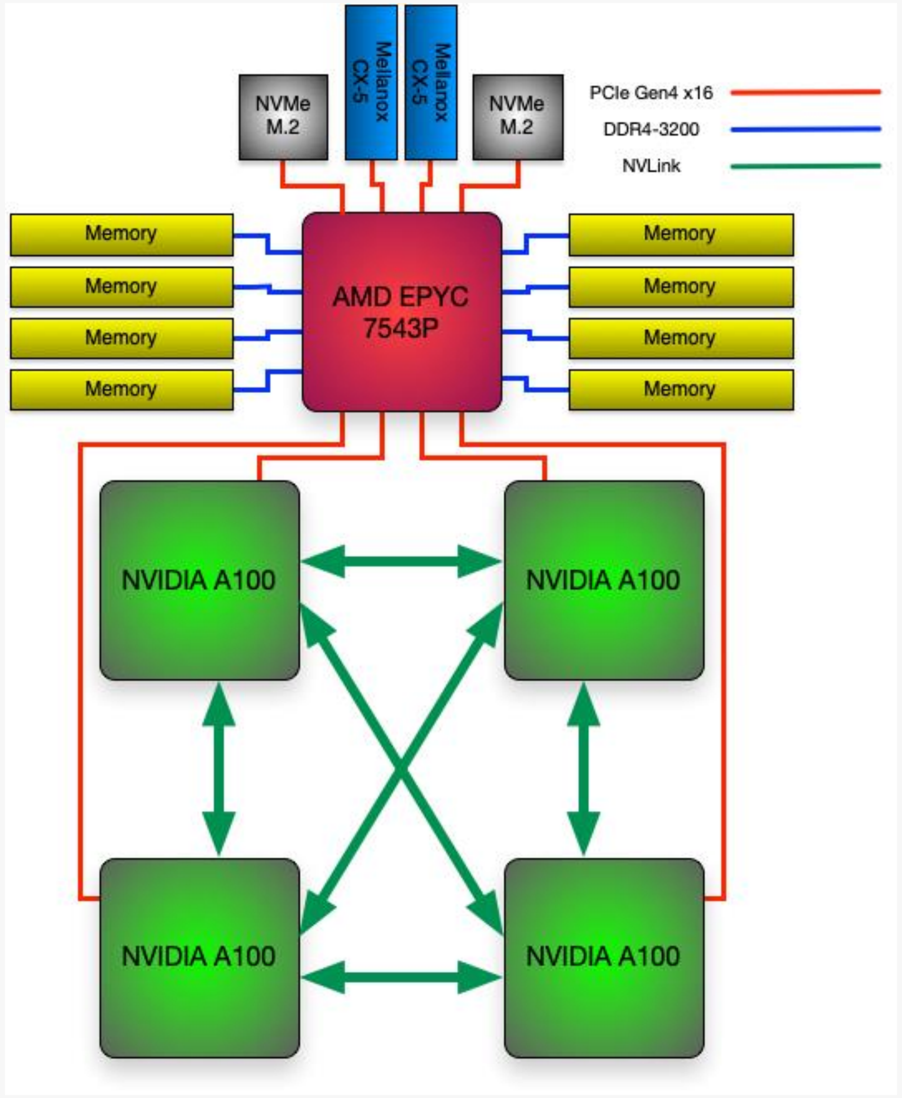
```
export OMP_NUM_THREADS=8
export OMP_PLACES=threads
mpiexec -n 8 --ppn 4 --depth=$OMP_NUM_THREADS \
                    --cpu-bind depth ./hello_affinity
```



- MPI rank 0, thread 0 → node 0, HW thread 0
- MPI rank 0, thread 1 → node 0, HW thread 1
- …
- MPI rank 1, thread 0 → node 0, HW thread 8
- MPI rank 1, thread 1 → node 0, HW thread 9
- …
- MPI rank 2, thread 0 → node 0, HW thread 16
- …
- MPI rank 3, thread 0 → node 0, HW thread 24
- …
- MPI rank 7, thread 0 → node 1, HW thread 24

GPU Affinity

- There are 4 A100 GPUs per node

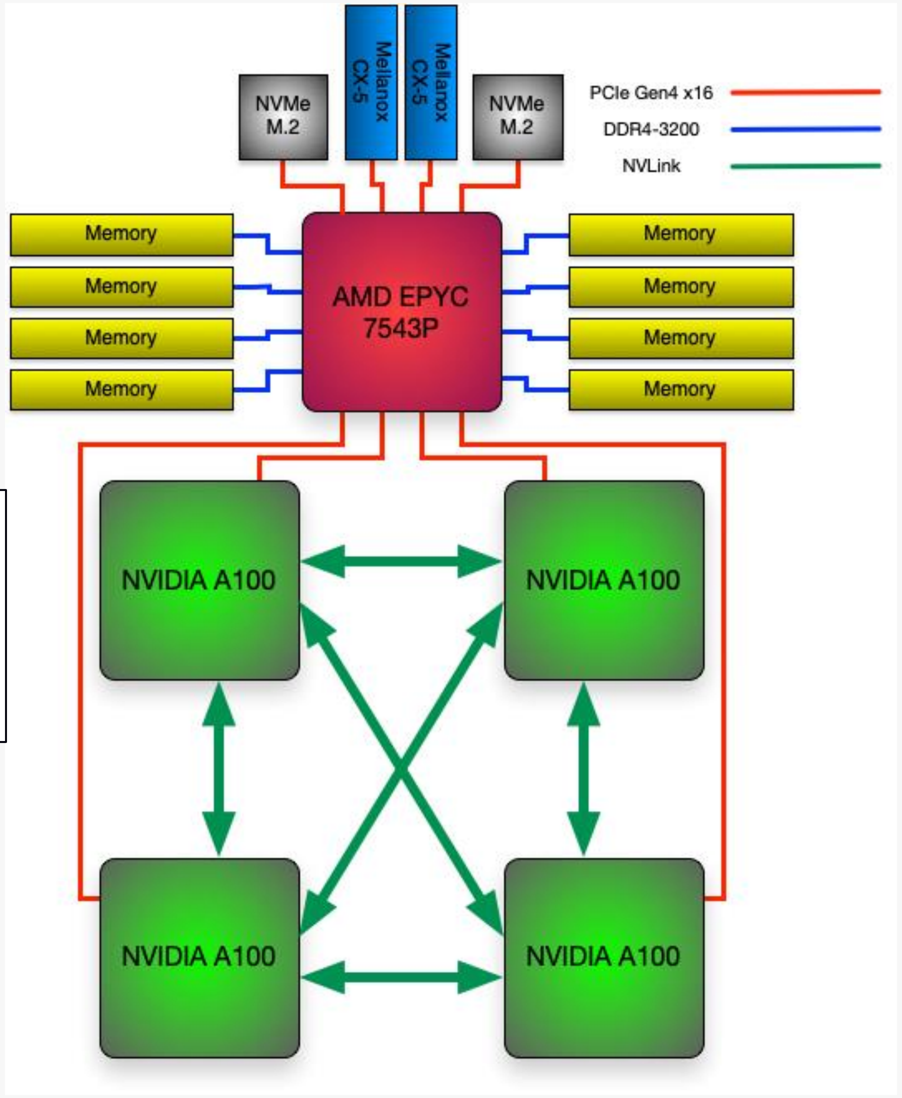


See: https://docs.alcf.anl.gov/running-jobs/example-job-scripts/

GPU Affinity

- There are 4 A100 GPUs per node
- You can manually target each with OpenMP offload via the **device** clause

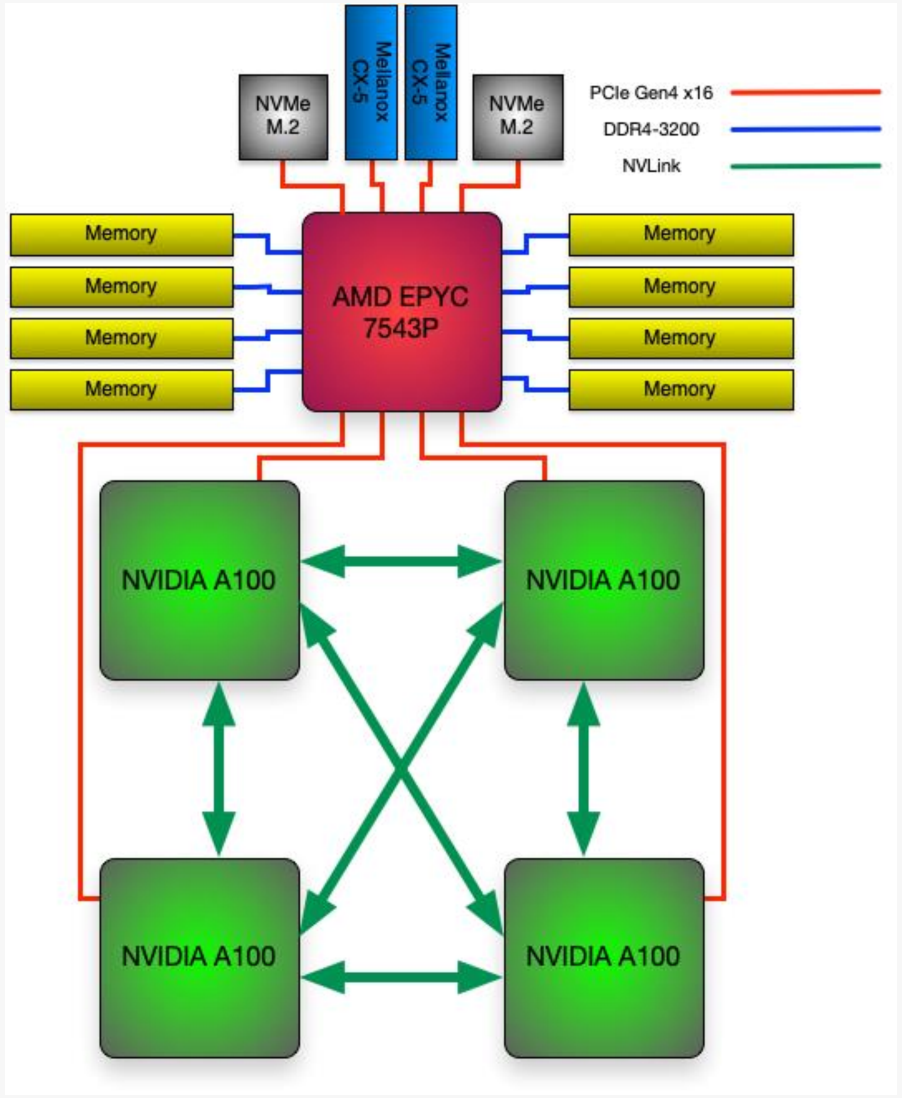
```
const int num_device = omp_get_num_devices();

#pragma omp target device(mpi_rank % num_device)
{}
```



See: https://docs.alcf.anl.gov/running-jobs/example-job-scripts/

GPU Affinity

- There are 4 A100 GPUs per node
- You can manually target each with OpenMP offload via the **device** clause
- Alternatively you can use the set_affinity_gpu_polaris script (https://github.com/argonne-lcf/GettingStarted/blob/master/Examples/Polaris/affinity_gpu/set_affinity_gpu_polaris.sh)
- Sets the environment variable CUDA_VISIBLE_DEVICES to a restricted set of GPUs (e.g. each MPI rank sees only one GPU).



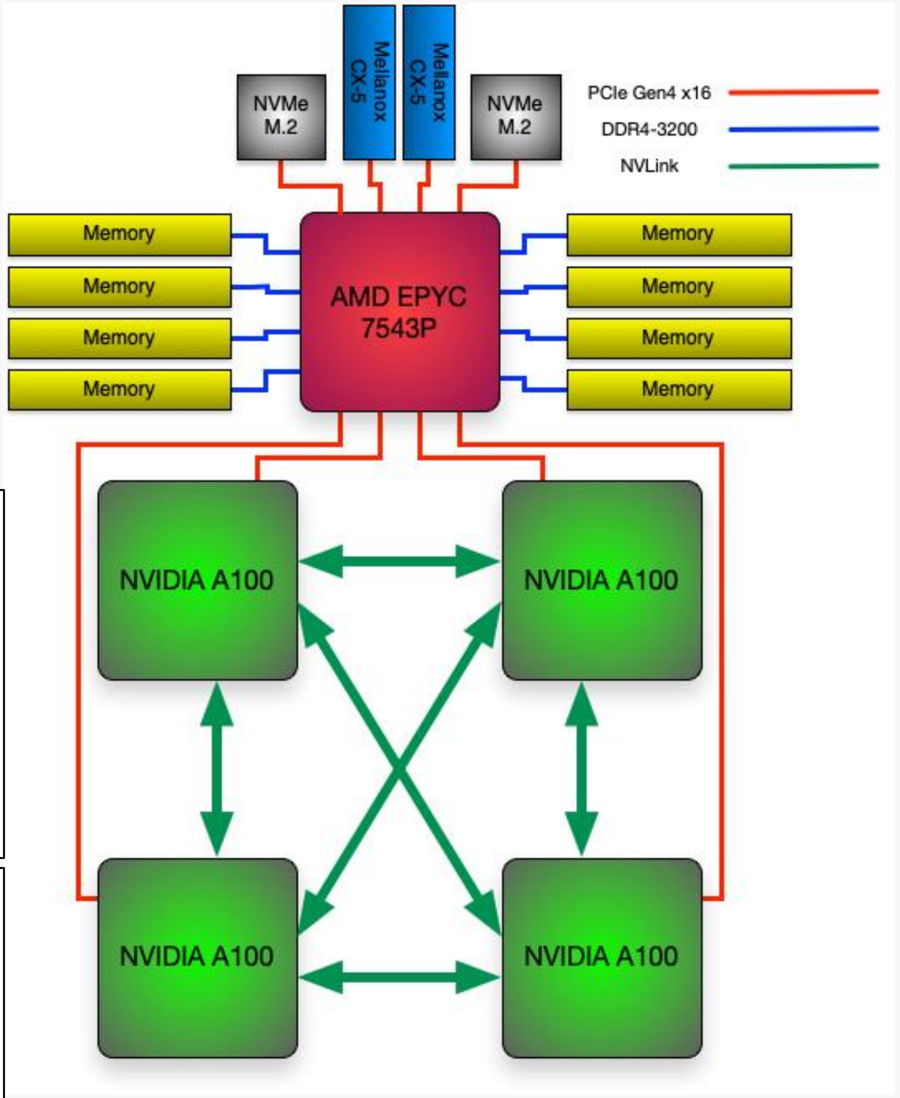
See: https://docs.alcf.anl.gov/running-jobs/example-job-scripts/

GPU Affinity

- There are 4 A100 GPUs per node
- You can manually target each with OpenMP offload via the **device** clause
- Alternatively you can use the set_affinity_gpu_polaris script

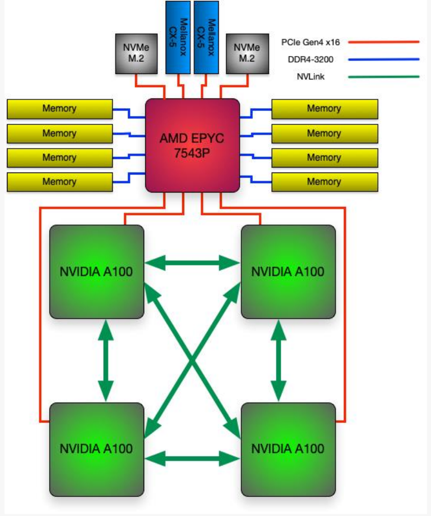
```
const int num_device = omp_get_num_devices();

#pragma omp target device(mpi_rank % num_device)
{}
```

```
// launches with set_affinity_gpu_polaris.sh so
// no need for device clause
#pragma omp target
{}
```



See: https://docs.alcf.anl.gov/running-jobs/example-job-scripts/

GPU Affinity: Example



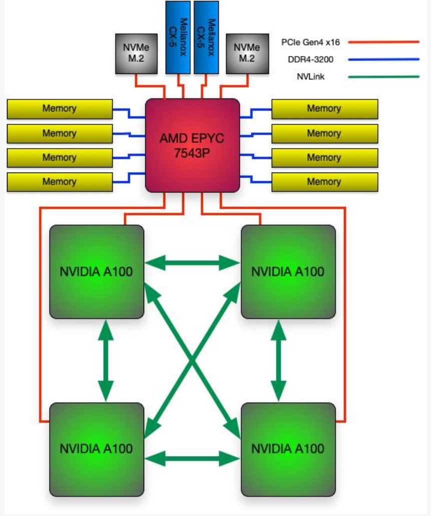
```
export OMP_NUM_THREADS=8
export OMP_PLACES=threads
mpiexec -n 8 --ppn 4 --depth=$OMP_NUM_THREADS \
    --cpu-bind depth ./set_affinity_gpu_polaris.sh ./hello_affinity
```

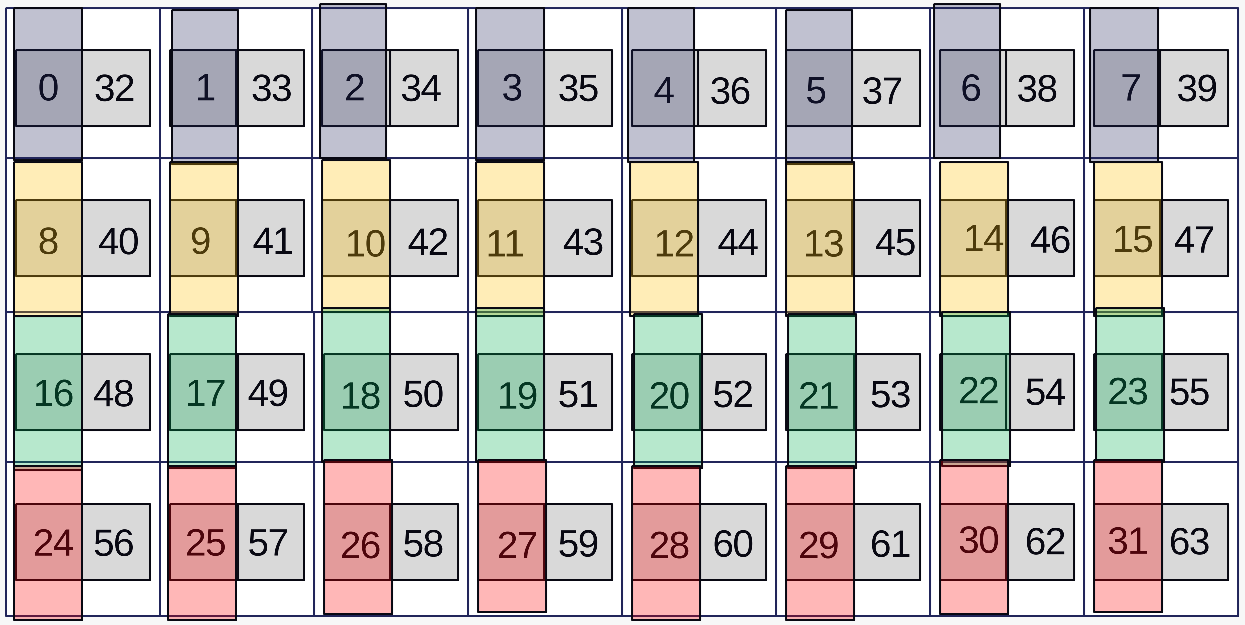
0 32	1 33	2 34	3 35	4 36	5 37	6 38	7 39
8 40	9 41	10 42	11 43	12 44	13 45	14 46	15 47
16 48	17 49	18 50	19 51	20 52	21 53	22 54	23 55
24 56	25 57	26 58	27 59	28 60	29 61	30 62	31 63

- Runs on 2 nodes with 4 ranks per node and 8 OpenMP threads per rank
- set_affinity_gpu_polaris.sh sets CUDA_VISIBLE_DEVICES such that each MPI rank only sees 1 GPU

GPU Affinity: Example

```
export OMP_NUM_THREADS=8
export OMP_PLACES=threads
mpiexec -n 8 --ppn 4 --depth=$OMP_NUM_THREADS \
    --cpu-bind depth ./set_affinity_gpu_polaris.sh ./hello_affinity
```





- MPI rank 0, thread 0-7 → node 0, GPU 3
- MPI rank 1, thread 0-7 → node 0, GPU 2
- MPI rank 2, thread 0-7 → node 0, GPU 1
- MPI rank 3, thread 0-7 → node 0, GPU 0
- …
- MPI rank 7, thread 0-7 → node 1, GPU 0

- **"nvidia-smi topo -m" shows the affinity of the GPUs for the CPUs**

Example from ALCF Github

```
git clone https://github.com/argonne-lcf/GettingStarted.git

cd GettingStarted/Examples/Polaris/affinity_gpu/

make -f Makefile.nvhpc

qsub -l select=1:system=polaris -l walltime=0:30:00 -l filesystems=home:eagle:grand -q HandsOnHPC -A
alcf_training ./submit.sh
```

Overview of this talk

- Using OpenMP on Polaris (~30 min)
  - Why OpenMP?
  - Quick Start for Running on Polaris
  - Using GPUs with OpenMP Offload
  - Multi-GPU runs: Affinity and binding to CPUs and GPUs on Polaris
- Demo of OpenMP (~20 min)
  - OpenMP 101 and basics on Polaris

```
$ git clone https://github.com/argonne-lcf/ALCF_Hands_on_HPC_Workshop
$ qsub -I -l select=1 -l walltime=1:0:0 -l filesystems=home:grand:eagle -q
HandsOnHPC -A alcf_training
$ cd ALCF_Hands_on_HPC_Workshop/programmingModels/OpenMP/demo
```