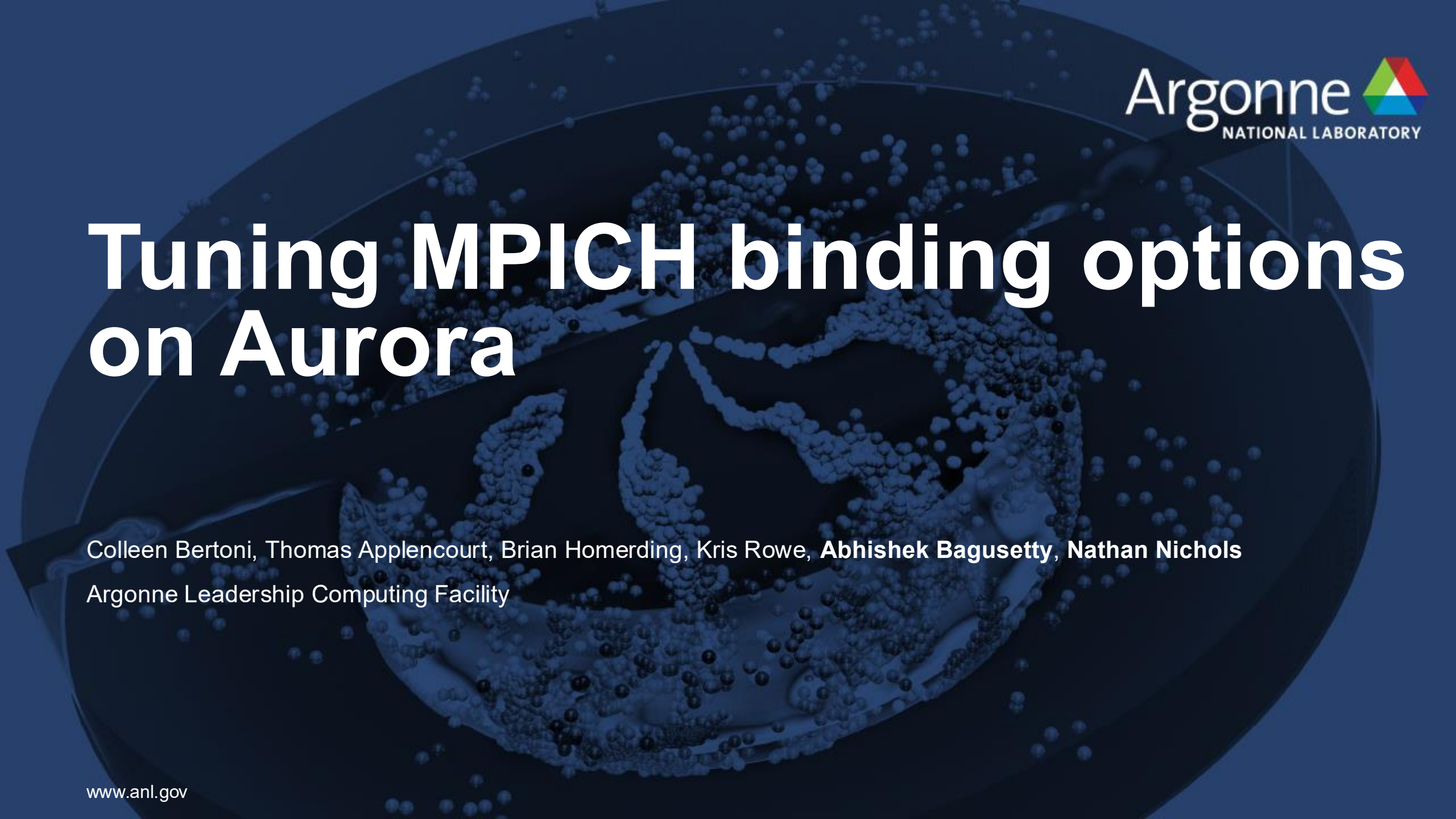


Tuning MPICH binding options on Aurora

Colleen Bertoni, Thomas Applencourt, Brian Homerding, Kris Rowe, **Abhishek Bagusetty**, **Nathan Nichols**

Argonne Leadership Computing Facility

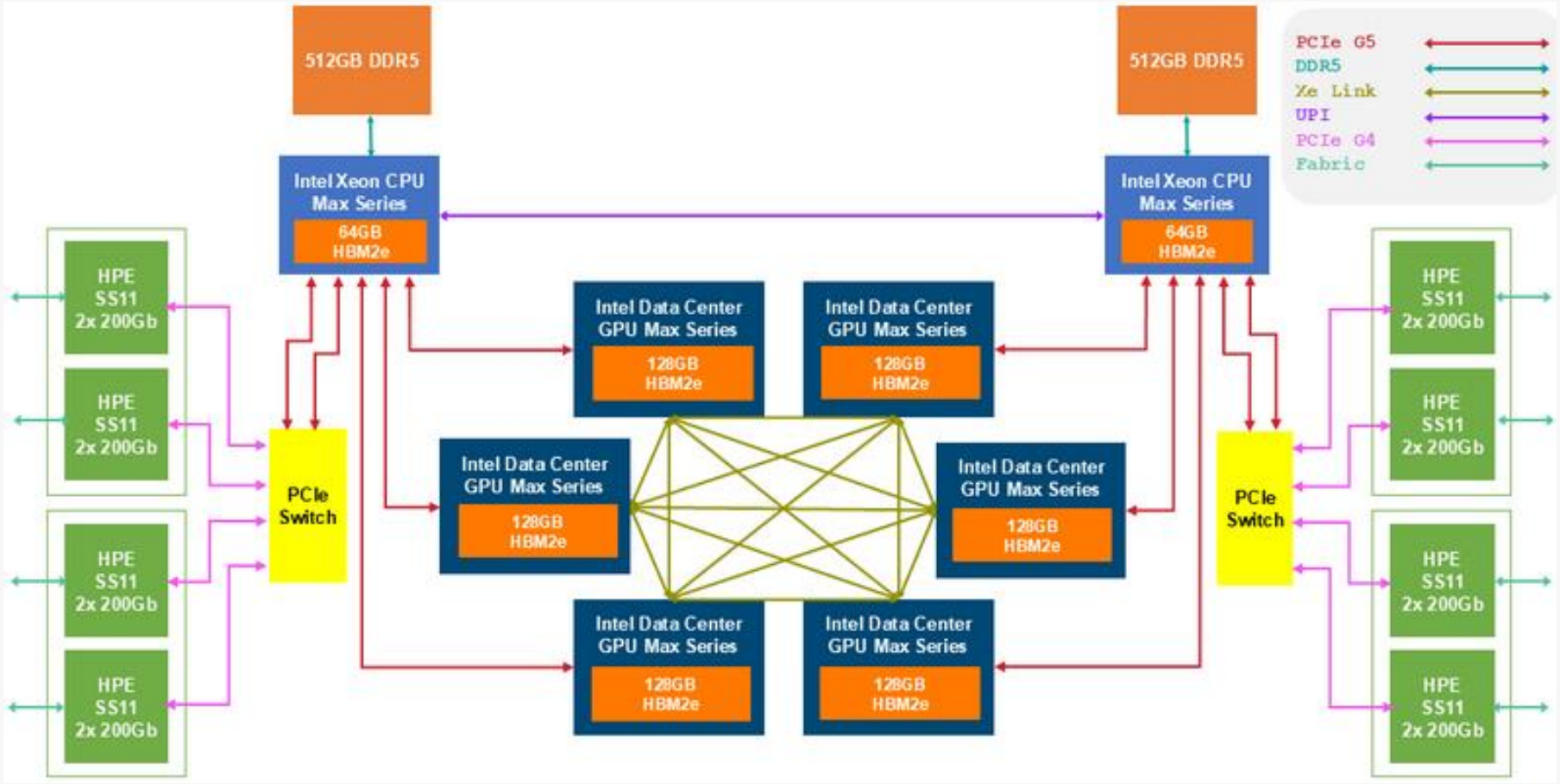
Agenda

- Overview of CPU-core binding
- Overview of GPU/tile binding
- Overview of NUMA Memory binding (DDR & HBM)

Refer to: https://docs.alcf.anl.gov/aurora/running-jobs-aurora/#running-mpiopenmpsycl-applications

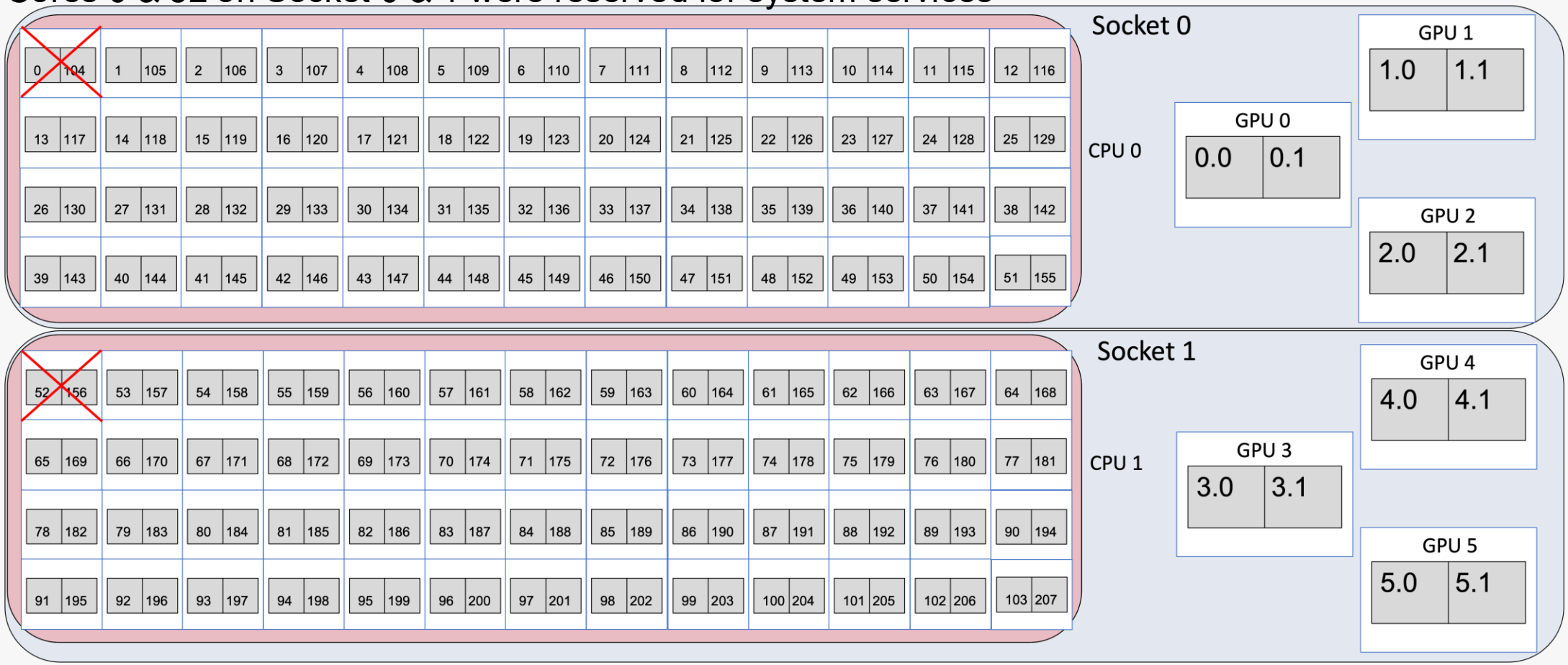
(Recap) Know your Node Topology

- Aurora node has dual socket CPUs
- Each socket houses – 3x 128GB PVC GPUs (2x 64GB tiles each)
- Two memory types on SPR CPU socket – 512GB DDR5 & 64GB HM2e



CPU core binding options for Aurora

- ✓ Know the topology when running on dual-socket Sapphire Rapids (SPR) CPUs and 2-tile Intel Data Center Max Series (PVC) GPUs
- ✓ Cores 0 & 52 on Socket 0 & 1 were reserved for system services



CPU core binding options for Aurora

- Bind MPI processes to the nearest GPU and nearest CPUs for optimal configuration
- MPI ranks binding scheme - Separated by Colon (:)
- Optimal configuration is to explicitly bind to cores via "**--cpu-bind=list:**" option for mpiexec so as to evenly distribute the ranks over the dual sockets of SPR

- Most common configuration:
- 12 MPI ranks per node
  - 6 ranks per socket
  - 1-8 CPU cores/OMP threads per rank

- **export CPU_BIND_SCHEME="--cpu-bind=list:1-8:9-16:17-24:25-32:33-40:41-48:53-60:61-68:69-76:77-84:85-92:93-100"**
- mpiexec -n 12 -ppn 12 **${CPU_BIND_SCHEME}** ./EXE

CPU core binding options for Aurora

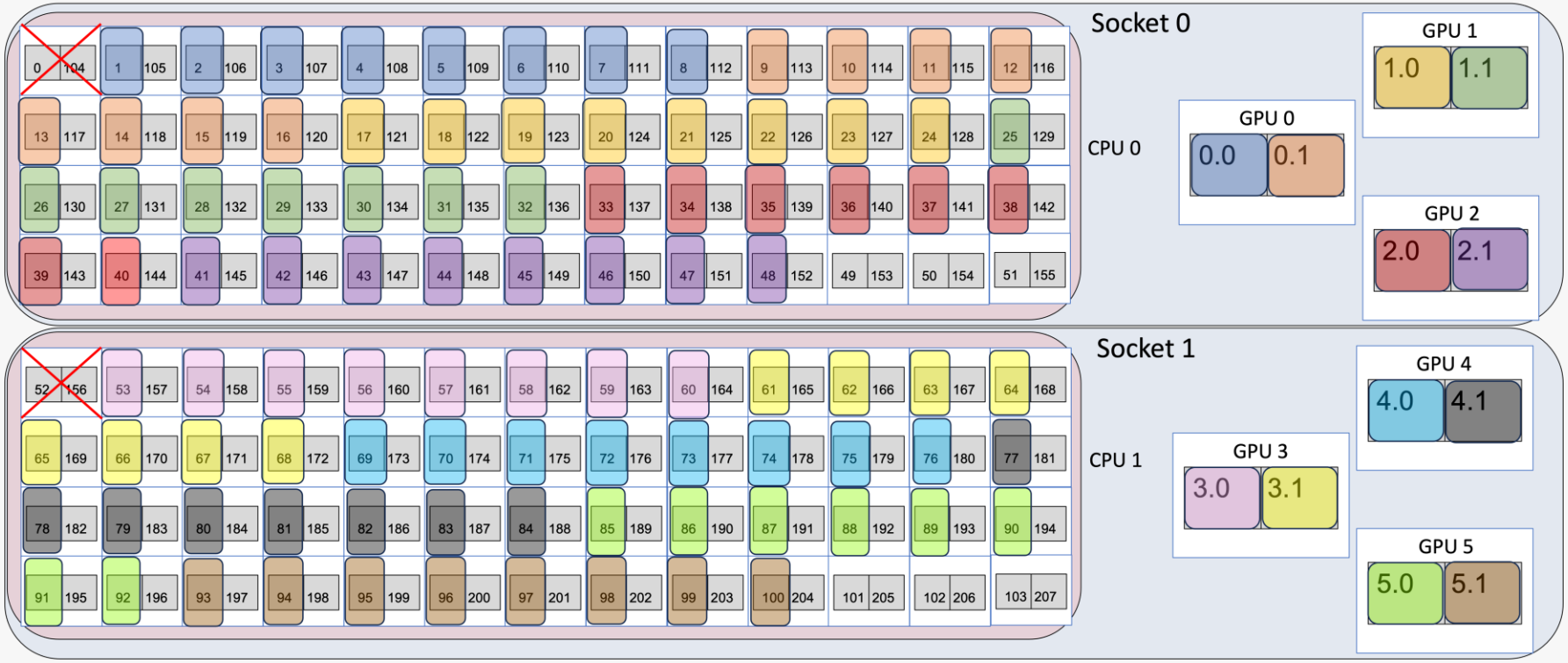
- **export CPU_BIND_SCHEME="--cpu-bind=list:1-8:9-16:17-24:25-32:33-40:41-48:53-60:61-68:69-76:77-84:85-92:93-100"**
- mpiexec -n 12 -ppn 12 **${CPU_BIND_SCHEME}** ./EXE

GPU binding options for Aurora

- In addition to CPU-core binding, GPU-binding options to MPI-task is important
- Optimal setup: To bind MPI-task to closest GPU on each socket.
  - Socket 0 – GPU 0,1,2 & Socket 1 – GPU 3,4,5
- Further, binding to a specific tile (Tile 0 & Tile 1) would yield an optimal configuration
- Device discovery of GPUs are controlled by an environment variable ZE_AFFINITY_MASK (similar to CUDA_VISIBLE_DEVICES)

GPU binding options for Aurora

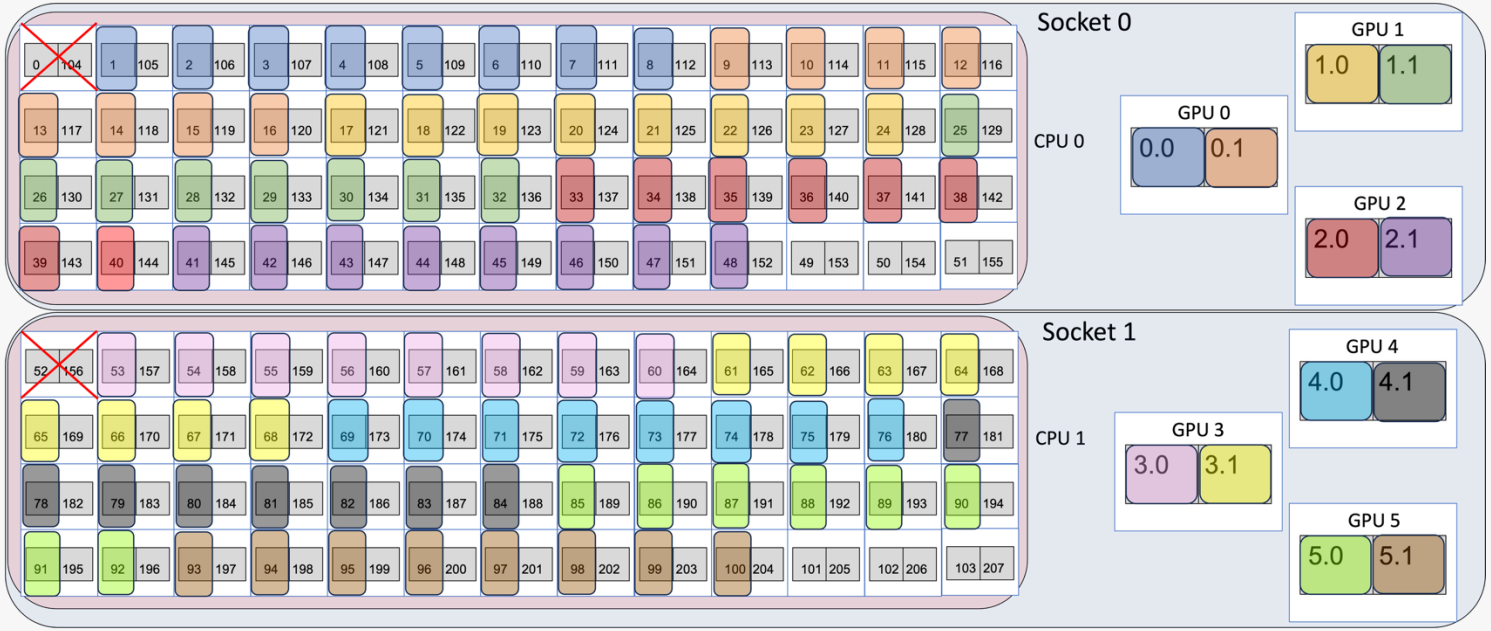
- Two options to bind GPU-tiles of PVC on Aurora
- (a) Using ALCF's helper script - gpu_tile_compact.sh (added by default to $PATH). More robust
- (b) Using MPICH command line option: --gpu-bind=list. Has some limitations



GPU binding options for Aurora (gpu_tile_compact.sh)

- ALCF has designed a wrapper script to bind (6 GPUs x 2 tile) 12 GPU-tiles to the MPI-ranks by setting the environment variable ZE_AFFINITY_MASK for each MPI-rank
- mpiexec -n 12 -ppn 12 ${CPU_BIND_SCHEME} gpu_tile_compact.sh ./EXE
- Such the mapping between MPI ranks to GPU-tiles is as follows

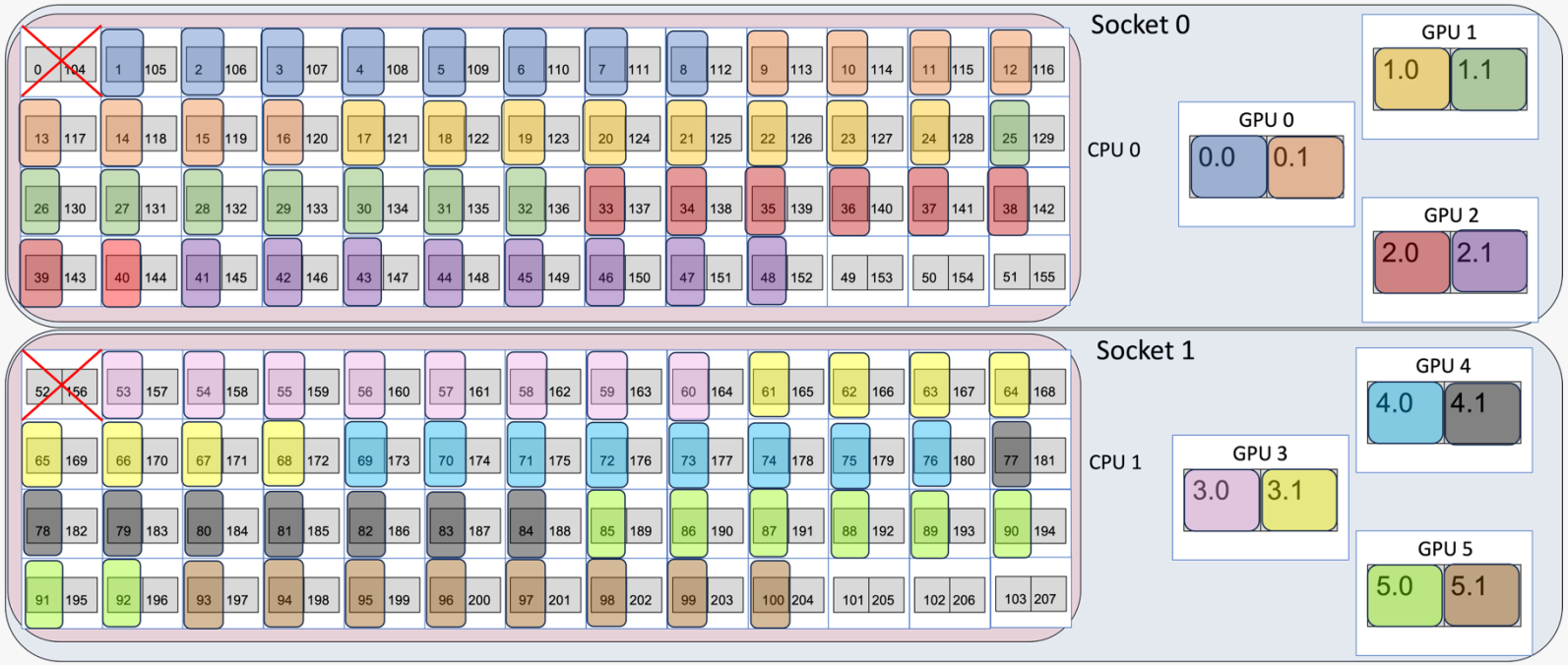
  MPI (Rank 0) → GPU 0 : Tile 0
  MPI (Rank 1) → GPU 0 : Tile 1
- MPI (Rank 2) → GPU 1 : Tile 0
  MPI (Rank 2) → GPU 1 : Tile 1
- …
- MPI (Rank 10) → GPU 5 : Tile 0
- MPI (Rank 11) → GPU 5 : Tile 1



GPU binding options for Aurora (via gpu-bind=list)

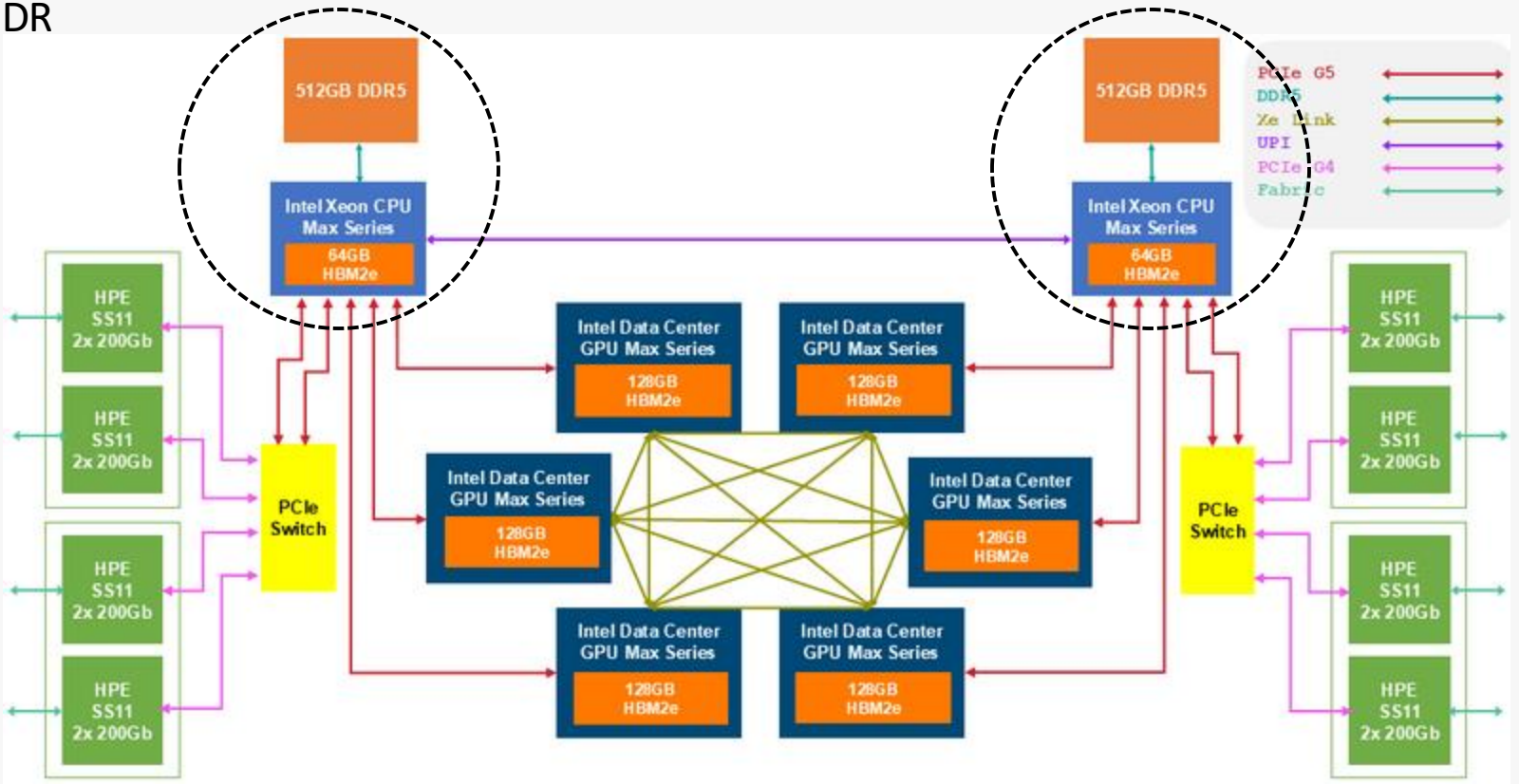
- Bind to each tile of PVC explicitly for optimal performance. (Format – GPU_ID.Tile_ID)
- **export GPU_BIND_SCHEME="--gpu-bind=list:0.0:0.1:1.0:1.1:2.0:2.1:3.0:3.1:4.0:4.1:5.0:5.1"**
- mpiexec -n 12 -ppn 12 ${CPU_BIND_SCHEME} **${GPU_BIND_SCHEME}** ./EXE

- (Limitation) Only works with this device hierarchy mode, ZE_FLAT_DEVICE_HIERARCHY=COMPOSITE (default option on Aurora). Wouldn't work with ZE_FLAT_DEVICE_HIERARCHY=FLAT mode set for frameworks module



NUMA memory binding options for Aurora

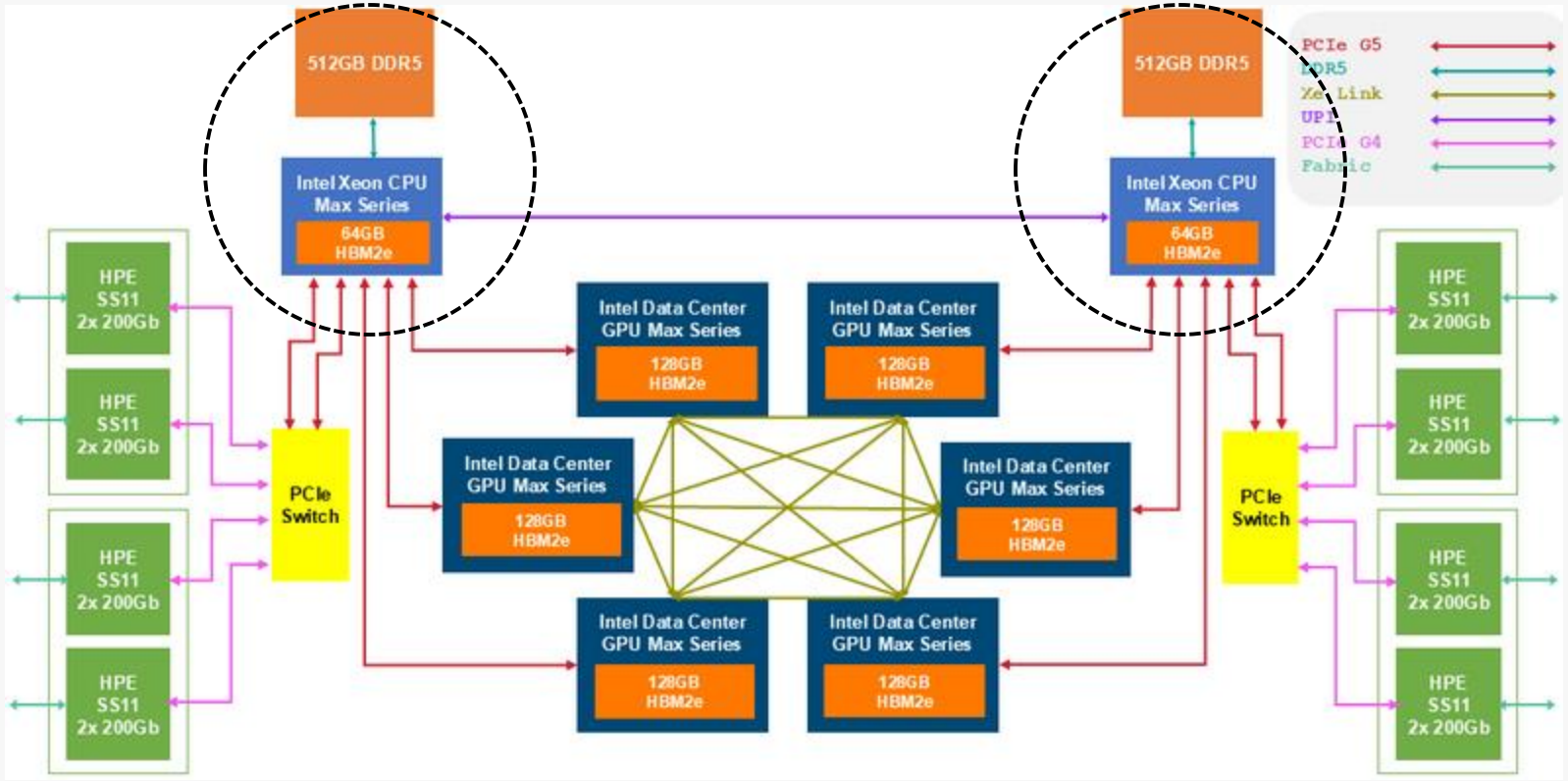
- Each SPR socket on Aurora has 512GB DDR5 and 64GB of HBM2e memory
- Optimal configuration is to have each rank be placed closer to DDR5, HBM2e memory banks on each socket (in-addition to CPU-binding & GPU-binding)
- Memory (de)allocations defaults to DDR
- Each MPI rank can be
- configured to access:
- (a) DDR5-only
  (slower, but large-capacity)
- (b) HBM2e-only
- (faster, but limited-capacity)
- (c) Both DDR5 & HBM2e
- (complex setup via using
- libnuma APIs)



NUMA memory binding options for Aurora

- ✓ Each SPR socket on Aurora has 512gb DDR5 and 64gb of HBM2e memory
- ✓ Socket 1 has NUMA nodes with IDS: 0(DDR5) & 2(HBM2e)
- ✓ Socket 2 has NUMA nodes with IDs: 1(DDR5) & 3(HBM2e)

```
$ numactl -H | grep -i "MB"
  node 0 size: 515524 MB
  node 0 free: 497948 MB
  node 1 size: 514994 MB
  node 1 free: 497013 MB
  node 2 size: 65536 MB
  node 2 free: 65424 MB
  node 3 size: 65536 MB
  node 3 free: 65433 MB
```



NUMA memory binding options for Aurora

- Each SPR socket on Aurora has 512gb DDR5 and 64gb of HBM2e memory
- Socket 1 has NUMA nodes 0(DDR5) & 2(HBM2e)
- Socket 2 has NUMA nodes 1(DDR5) & 3(HBM2e)

- Using DDR5 only:
  export MEM_BIND_SCHEME="--mem-bind=list:0:0:0:0:0:0:1:1:1:1:1:1"

- Using HBM2e only:
  export MEM_BIND_SCHEME="--mem-bind=list:2:2:2:2:2:2:3:3:3:3:3:3"

- Using both DDR5 & HBM2e:
- export MEM_BIND_SCHEME="--mem-bind=list:0,2:0,2:0,2:0,2:0,2:0,2:1,3:1,3:1,3:1,3:1,3:1,3"
  - Each MPI-rank binds to both DDR & HBM memory to its closest memory banks on each socket.
  - Please note memory doesn't allocate from DDR to HBM when the size exceeds on DDR.
  - One would have to explicitly use libnuma APIs to explicitly allocate for both DDR and HBM

Optimal settings for Aurora

- Be very explicit with MPICH binding options for optimal performance on Aurora
- The following is the most common configuration: 12 MPI-ranks per node:
- export CPU_BIND_SCHEME="--cpu-bind=list:1-8:9-16:17-24:25-32:33-40:41-48:53-60:61-68:69-76:77-84:85-92:93-100"
- export MEM_BIND_SCHEME="--mem-bind=list:0,2:0,2:0,2:0,2:0,2:1,3:1,3:1,3:1,3:1,3"
- mpiexec -n 12 -ppn 12 ${CPU_BIND_SCHEME} ${MEM_BIND_SCHEME} gpu_tile_compact.sh ./EXE

(OR)

- export GPU_BIND_SCHEME="--gpu-bind=list:0.0:0.1:1.0:1.1:2.0:2.1:3.0:3.1:4.0:4.1:5.0:5.1"
- mpiexec -n 12 -ppn 12 ${CPU_BIND_SCHEME} ${MEM_BIND_SCHEME} ${GPU_BIND_SCHEME} ./EXE