OAK RIDGE National Laboratory

Crusher Tips & Tricks

Tom Papatheodore

HPC Engineer

System Acceptance & User Environment Group

Oak Ridge Leadership Computing Facility (OLCF)

ECP BOF Days – May 10, 2022

U.S. DEPARTMENT OF ENERGY

GPU-to-~~GPU~~ GCD Performance Expectations

- Is your application GPU-bound or data-transfer-bound?
  - If GPU-bound, is it compute-bound or memory-bound?
    - Consider Compute performance, HBM, and HBM bandwidth in table
    - Also consider register usage, occupancy, launch configuration parameters
    - This is where kernel profiling can be helpful
  - If data-transfer-bound, consider CPU-to-GPU bandwidth in table.

GPU-to-GPU Comparison

	Summit	Crusher (per GCD)	Crusher / Summit
Compute performance	~7.8 TF	~26 TF	~3.3X
HBM	16 GB	64 GB	4X
HBM bandwidth	0.9 TB/s	1.6 TB/s	~1.8X
CPU-to-GPU bandwidth	50 GB/s	36 GB/s	~0.7X
L1 Cache	Up to 128 KB	16 KB	0.125X – 0.5X
L2 Cache	6 MB	8 MB	1.3X

Slurm Tips – Flags and Completed Jobs

`-u` flag gives unbuffered output, which can be helpful when debugging.

`-l` flag prepends the task ID to lines of stdout.

To show completed jobs in a specific time period, specify a start (`-S`) and end (`-E`) time

- The default time window depends on other options (see `man sacct`)

```
$ sacct --user=tpapathe -S 2022-05-05T00:01 -E 2022-05-05T23:59 | awk '$0 !~ /\./'

JobID           JobName  Partition    Account  AllocCPUS      State ExitCode
------------ ---------- ---------- ---------- ---------- ---------- --------
107814       interacti+      batch     stf016        128  COMPLETED      0:0
107836       interacti+      batch     stf016        128  COMPLETED      0:0
107849       hello_job+      batch     stf016        128  COMPLETED      0:0
107858       hello_job+      batch     stf016        256  COMPLETED      0:0
107866       hello_job+      batch     stf016        256  COMPLETED      0:0
107867       hello_job+      batch     stf016        256 CANCELLED+      0:0
107868       hello_job+      batch     stf016        256  COMPLETED      0:0
107965       MatrixTra+      batch     stf016        128  COMPLETED      0:0
107966          rocprof      batch     stf016        128     FAILED      6:0
107969          rocprof      batch     stf016        128     FAILED      6:0
```

Then you can drill into more details about each job with the `-j` flag and customized output.

Slurm Tips – Capture Job Information

```
$ cat submit.sh
#!/bin/bash

#SBATCH -A stf016
#SBATCH -J job-name
#SBATCH -o %x-%j.out
#SBATCH -t 5
#SBATCH -p batch
#SBATCH -N 1

scontrol show job ${SLURM_JOBID}

srun -n1 ./p2p/p2p --correct

sacct -j ${SLURM_JOBID} -o jobid%20,Start%20,elapsed%20
```

```
$ cat job-name-108121.out

JobId=108121 JobName=job-name
   UserId=tpapathe(5987) GroupId=tpapathe(8654) MCS_label=N/A
   Priority=200 Nice=0 Account=stf016 QOS=normal
   JobState=RUNNING Reason=None Dependency=(null)
   Requeue=0 Restarts=0 BatchFlag=1 Reboot=0 ExitCode=0:0
   RunTime=00:00:00 TimeLimit=00:05:00 TimeMin=N/A
   SubmitTime=2022-05-05T17:34:51 EligibleTime=2022-05-05T17:34:51
   AccrueTime=2022-05-05T17:34:51
   StartTime=2022-05-05T17:35:03 EndTime=2022-05-05T17:40:03 Deadline=N/A
   SuspendTime=None SecsPreSuspend=0 LastSchedEval=2022-05-05T17:35:03 Scheduler=Backfill
   Partition=batch AllocNode:Sid=login2:106343
   ReqNodeList=(null) ExcNodeList=(null)
   NodeList=crusher048
   BatchHost=crusher048
   NumNodes=1 NumCPUs=128 NumTasks=1 CPUs/Task=1 ReqB:S:C:T=0:0:*:1
   TRES=cpu=128,node=1,billing=128
   Socks/Node=* NtasksPerN:B:S:C=0:0:*:* CoreSpec=*
   MinCPUsNode=1 MinMemoryNode=0 MinTmpDiskNode=0
   Features=(null) DelayBoot=00:00:00
   OverSubscribe=NO Contiguous=0 Licenses=(null) Network=(null)
   Command=/autofs/nccs-svm1_home1/tpapathe/capture-job-info/submit.sh
   WorkDir=/autofs/nccs-svm1_home1/tpapathe/capture-job-info
   StdErr=/autofs/nccs-svm1_home1/tpapathe/capture-job-info/job-name-108121.out
   StdIn=/dev/null
   StdOut=/autofs/nccs-svm1_home1/tpapathe/capture-job-info/job-name-108121.out


...<APPLICATION OUTPUT>...


 JobID                Start              Elapsed
-------------------- ------------------- ----------
               108121  2022-05-05T17:35:03   00:02:34
         108121.batch  2022-05-05T17:35:03   00:02:34
        108121.extern  2022-05-05T17:35:03   00:02:34
            108121.0   2022-05-05T17:35:04   00:02:32
```

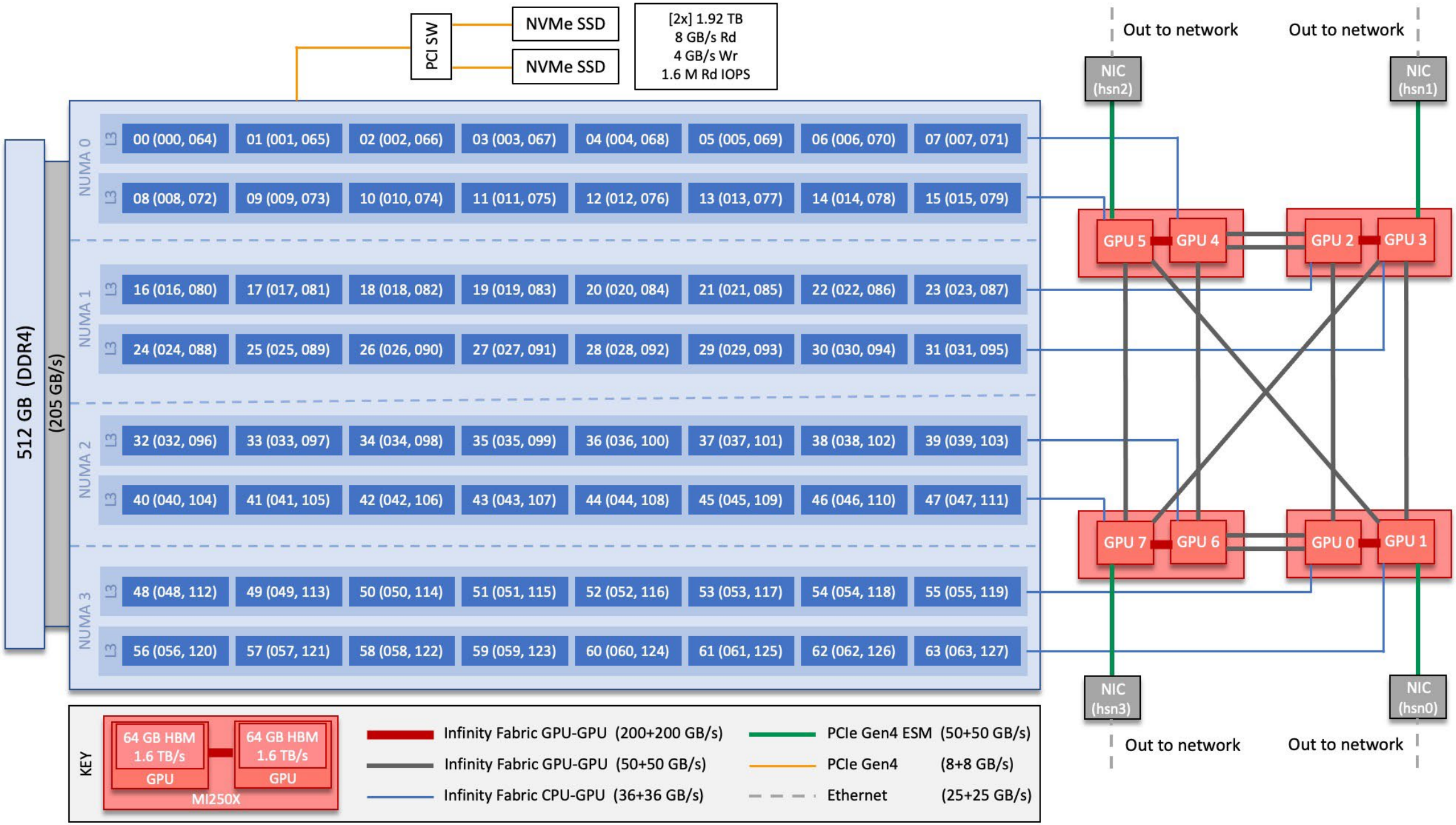
Add other information as needed.

- e.g., `module -t list`

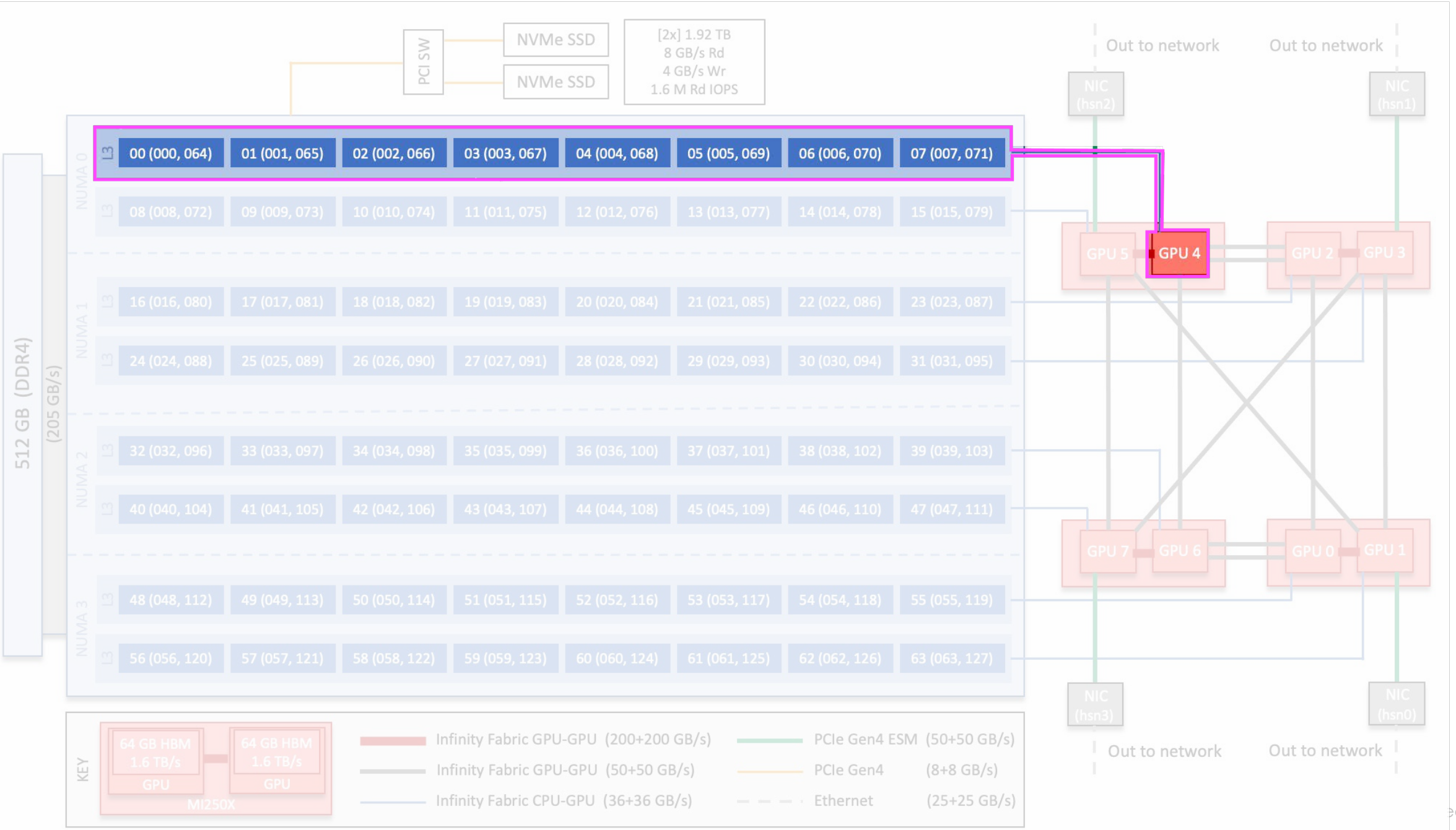
Slurm Tips – Test CPU/GPU Binding Before Running

```
[crusher: ~]$ module load rocm
[crusher: ~]$ module load craype-accel-amd-gfx90a

[crusher: ~]$ git clone https://code.ornl.gov/olcf/hello_jobstep.git
...
[crusher: ~]$ cd hello_jobstep/

[crusher: ~/hello_jobstep]$ make
crusher
CC -std=c++11 -fopenmp --rocm-path=/opt/rocm-4.5.0 -x hip -D__HIP_ARCH_GFX90A__=1 --offload-arch=gfx90a -I/opt/rocm-
4.5.0/include -c hello_jobstep.cpp
CC -fopenmp --rocm-path=/opt/rocm-4.5.0 -L/opt/rocm-4.5.0/lib -lamdhip64 hello_jobstep.o -o hello_jobstep

[crusher: ~/hello_jobstep]$ OMP_NUM_THREADS=2 srun -N1 -n8 -c8 --gpus-per-node=8 --gpu-bind=closest ./hello_jobstep | sort

MPI 000 - OMP 000 - HWT 001 - Node crusher166 - RT_GPU_ID 0 - GPU_ID 4 - Bus_ID d1
MPI 000 - OMP 001 - HWT 004 - Node crusher166 - RT_GPU_ID 0 - GPU_ID 4 - Bus_ID d1
MPI 001 - OMP 000 - HWT 008 - Node crusher166 - RT_GPU_ID 0 - GPU_ID 5 - Bus_ID d6
MPI 001 - OMP 001 - HWT 012 - Node crusher166 - RT_GPU_ID 0 - GPU_ID 5 - Bus_ID d6
MPI 002 - OMP 000 - HWT 018 - Node crusher166 - RT_GPU_ID 0 - GPU_ID 2 - Bus_ID c9
MPI 002 - OMP 001 - HWT 021 - Node crusher166 - RT_GPU_ID 0 - GPU_ID 2 - Bus_ID c9
MPI 003 - OMP 000 - HWT 025 - Node crusher166 - RT_GPU_ID 0 - GPU_ID 3 - Bus_ID ce
MPI 003 - OMP 001 - HWT 028 - Node crusher166 - RT_GPU_ID 0 - GPU_ID 3 - Bus_ID ce
MPI 004 - OMP 000 - HWT 032 - Node crusher166 - RT_GPU_ID 0 - GPU_ID 6 - Bus_ID d9
MPI 004 - OMP 001 - HWT 037 - Node crusher166 - RT_GPU_ID 0 - GPU_ID 6 - Bus_ID d9
MPI 005 - OMP 000 - HWT 040 - Node crusher166 - RT_GPU_ID 0 - GPU_ID 7 - Bus_ID de
MPI 005 - OMP 001 - HWT 044 - Node crusher166 - RT_GPU_ID 0 - GPU_ID 7 - Bus_ID de
MPI 006 - OMP 000 - HWT 049 - Node crusher166 - RT_GPU_ID 0 - GPU_ID 0 - Bus_ID c1
MPI 006 - OMP 001 - HWT 052 - Node crusher166 - RT_GPU_ID 0 - GPU_ID 0 - Bus_ID c1
MPI 007 - OMP 000 - HWT 059 - Node crusher166 - RT_GPU_ID 0 - GPU_ID 1 - Bus_ID c6
MPI 007 - OMP 001 - HWT 060 - Node crusher166 - RT_GPU_ID 0 - GPU_ID 1 - Bus_ID c6
```

https://code.ornl.gov/olcf/hello_jobstep

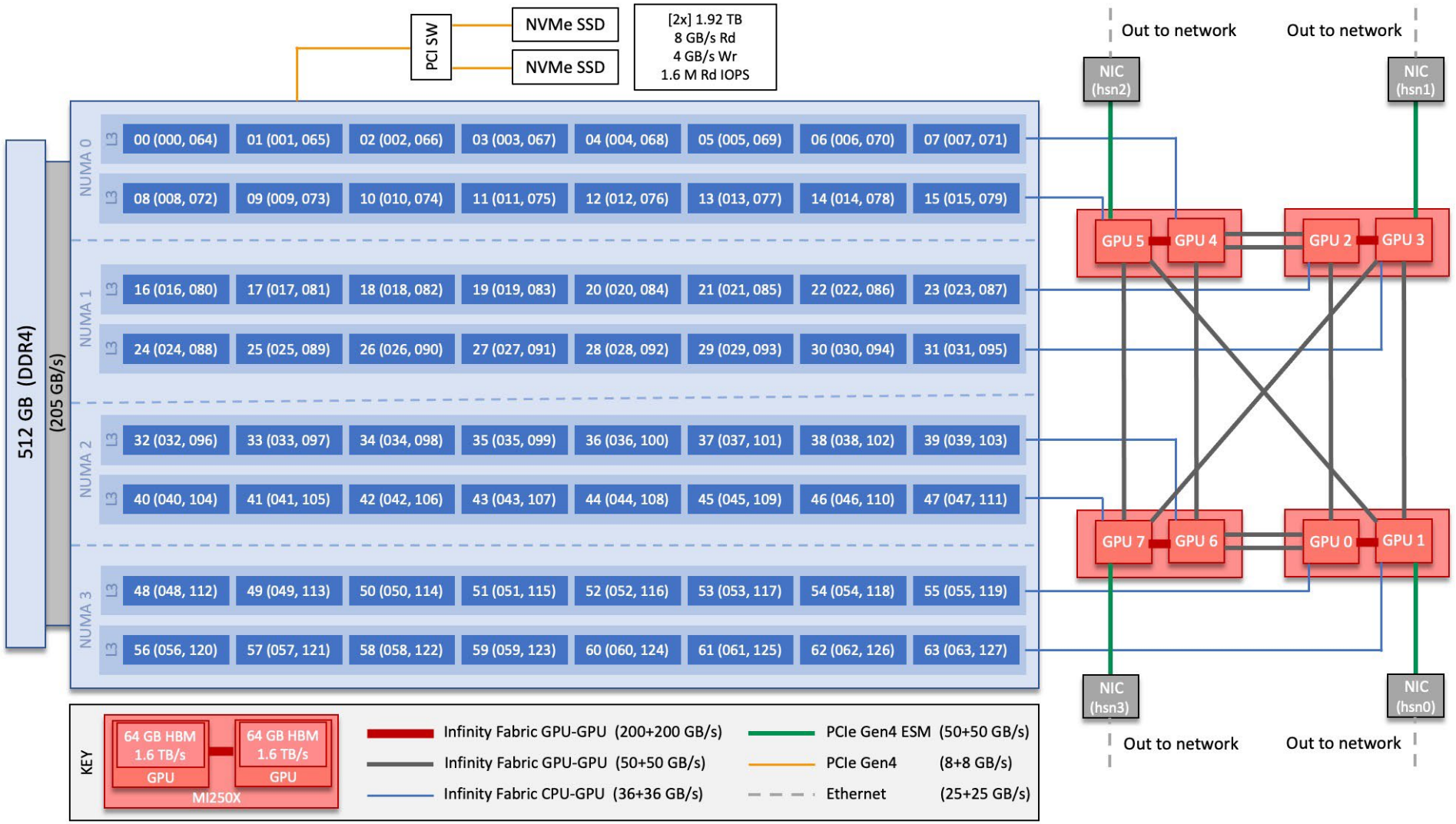

PCI SW
NVMe SSD
NVMe SSD
[2x] 1.92 TB
8 GB/s Rd
4 GB/s Wr
1.6 M Rd IOPS
512 GB (DDR4)
(205 GB/s)
NUMA 0
L3 00 (000, 064) 01 (001, 065) 02 (002, 066) 03 (003, 067) 04 (004, 068) 05 (005, 069) 06 (006, 070) 07 (007, 071)
L3 08 (008, 072) 09 (009, 073) 10 (010, 074) 11 (011, 075) 12 (012, 076) 13 (013, 077) 14 (014, 078) 15 (015, 079)
NUMA 1
L3 16 (016, 080) 17 (017, 081) 18 (018, 082) 19 (019, 083) 20 (020, 084) 21 (021, 085) 22 (022, 086) 23 (023, 087)
L3 24 (024, 088) 25 (025, 089) 26 (026, 090) 27 (027, 091) 28 (028, 092) 29 (029, 093) 30 (030, 094) 31 (031, 095)
NUMA 2
L3 32 (032, 096) 33 (033, 097) 34 (034, 098) 35 (035, 099) 36 (036, 100) 37 (037, 101) 38 (038, 102) 39 (039, 103)
L3 40 (040, 104) 41 (041, 105) 42 (042, 106) 43 (043, 107) 44 (044, 108) 45 (045, 109) 46 (046, 110) 47 (047, 111)
NUMA 3
L3 48 (048, 112) 49 (049, 113) 50 (050, 114) 51 (051, 115) 52 (052, 116) 53 (053, 117) 54 (054, 118) 55 (055, 119)
L3 56 (056, 120) 57 (057, 121) 58 (058, 122) 59 (059, 123) 60 (060, 124) 61 (061, 125) 62 (062, 126) 63 (063, 127)
Out to network
Out to network
NIC (hsn2)
NIC (hsn1)
GPU 5
GPU 4
GPU 2
GPU 3
GPU 7
GPU 6
GPU 0
GPU 1
NIC (hsn3)
NIC (hsn0)
Out to network
Out to network
KEY
64 GB HBM 1.6 TB/s
GPU
64 GB HBM 1.6 TB/s
GPU
MI250X
Infinity Fabric GPU-GPU (200+200 GB/s)
Infinity Fabric GPU-GPU (50+50 GB/s)
Infinity Fabric CPU-GPU (36+36 GB/s)
PCIe Gen4 ESM (50+50 GB/s)
PCIe Gen4 (8+8 GB/s)
Ethernet (25+25 GB/s)

MPI ranks should target the GPU associated with their L3 cache region



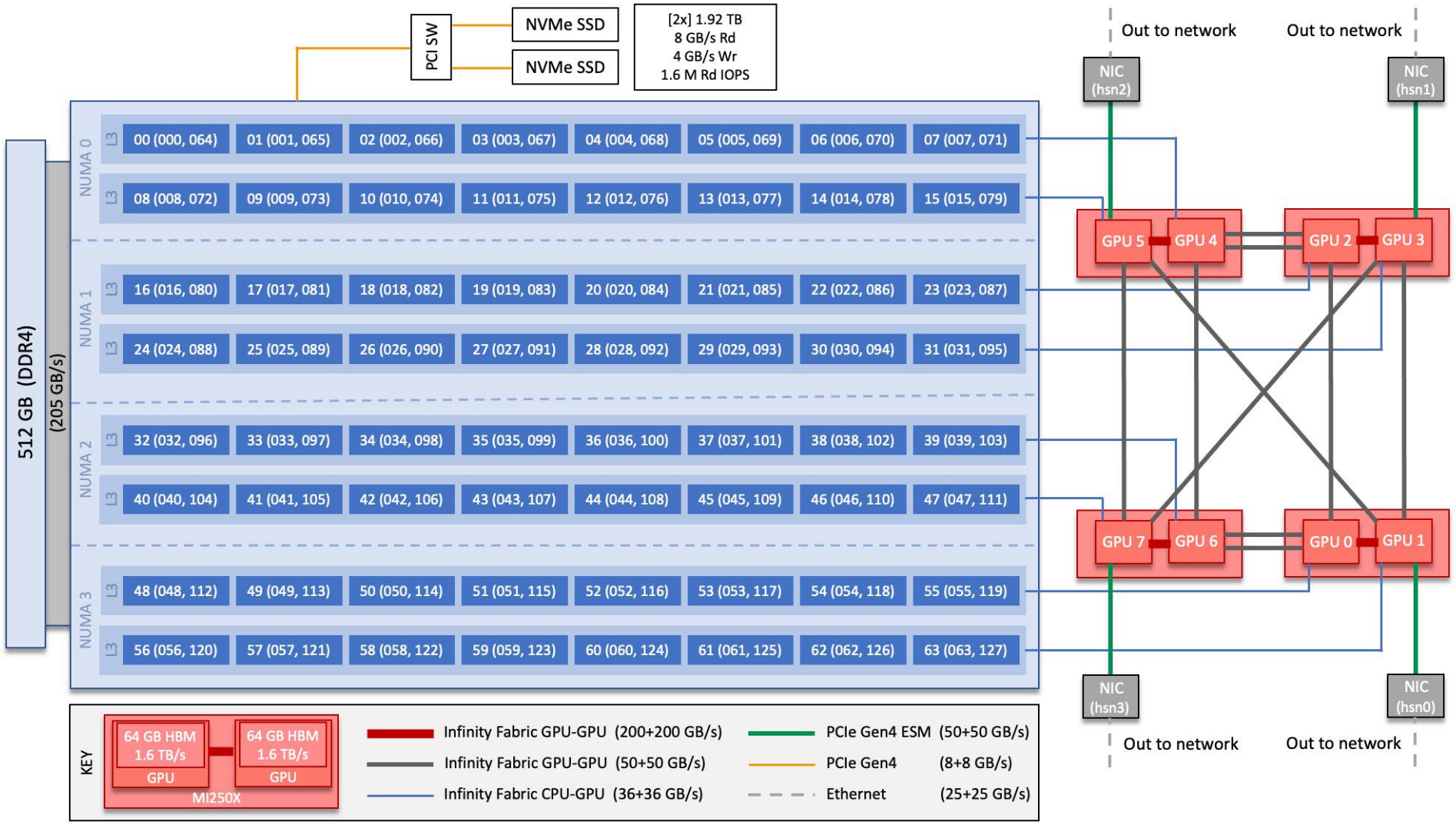
```
$ OMP_NUM_THREADS=1 srun -N1 -n8 -c1 --gpus-per-task=1 ./hello_jobstep | sort
MPI 000 - OMP 000 - HWT 000 - Node crusher124 - RT_GPU_ID 0 - GPU_ID 0 - Bus_ID c1
MPI 001 - OMP 000 - HWT 008 - Node crusher124 - RT_GPU_ID 0 - GPU_ID 1 - Bus_ID c6
MPI 002 - OMP 000 - HWT 016 - Node crusher124 - RT_GPU_ID 0 - GPU_ID 2 - Bus_ID c9
MPI 003 - OMP 000 - HWT 024 - Node crusher124 - RT_GPU_ID 0 - GPU_ID 3 - Bus_ID ce
MPI 004 - OMP 000 - HWT 032 - Node crusher124 - RT_GPU_ID 0 - GPU_ID 4 - Bus_ID d1
MPI 005 - OMP 000 - HWT 040 - Node crusher124 - RT_GPU_ID 0 - GPU_ID 5 - Bus_ID d6
MPI 006 - OMP 000 - HWT 048 - Node crusher124 - RT_GPU_ID 0 - GPU_ID 6 - Bus_ID d9
MPI 007 - OMP 000 - HWT 056 - Node crusher124 - RT_GPU_ID 0 - GPU_ID 7 - Bus_ID de
```

Non-ideal mapping

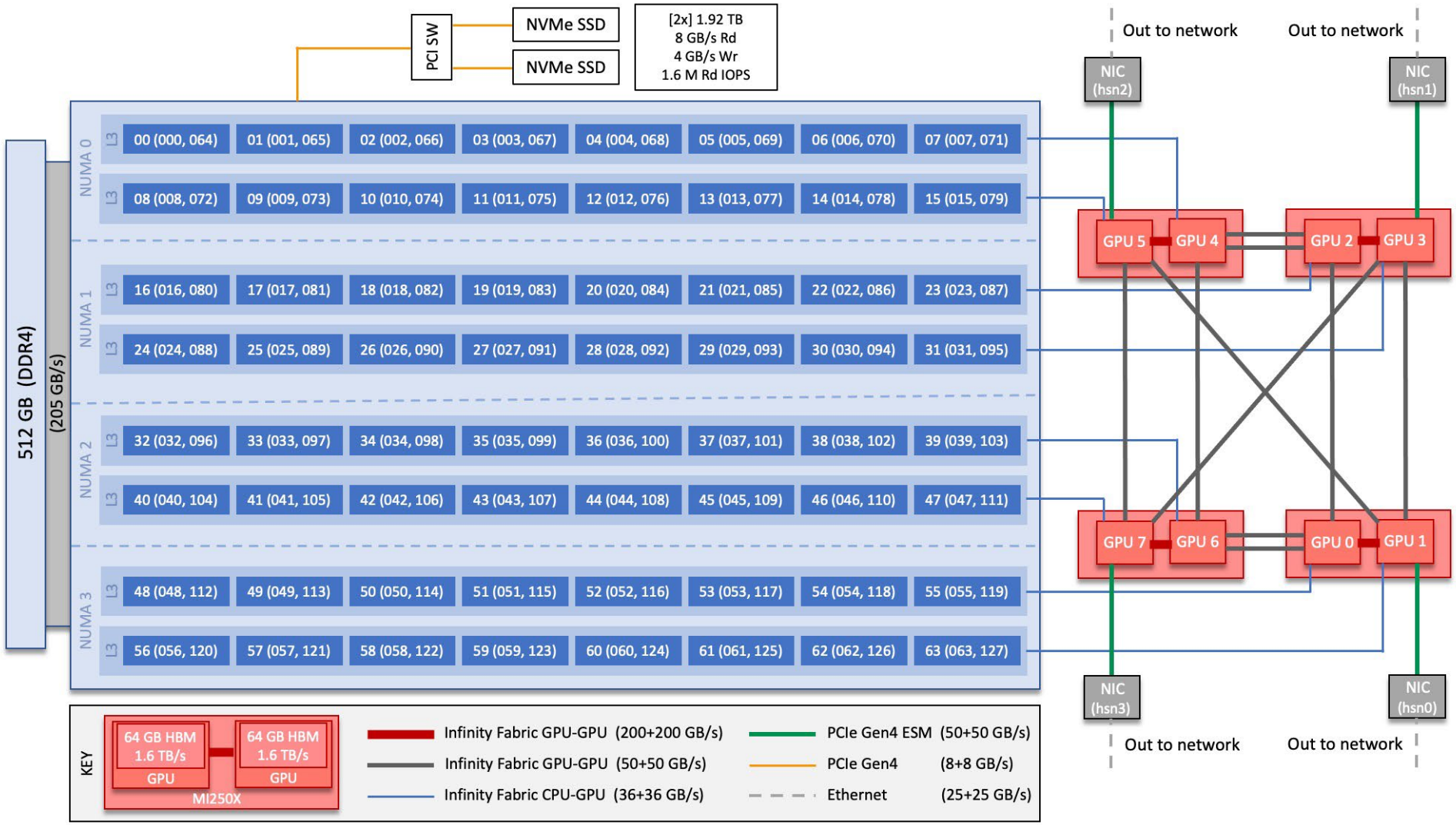


```
$ OMP_NUM_THREADS=1 srun -N1 -n8 -c1 --gpus-per-task=1 --gpu-bind=closest ./hello_jobstep | sort
MPI 000 - OMP 000 - HWT 000 - Node crusher124 - RT_GPU_ID 0 - GPU_ID 4 - Bus_ID d1
MPI 001 - OMP 000 - HWT 008 - Node crusher124 - RT_GPU_ID 0 - GPU_ID 5 - Bus_ID d6
MPI 002 - OMP 000 - HWT 016 - Node crusher124 - RT_GPU_ID 0 - GPU_ID 2 - Bus_ID c9
MPI 003 - OMP 000 - HWT 024 - Node crusher124 - RT_GPU_ID 0 - GPU_ID 3 - Bus_ID ce
MPI 004 - OMP 000 - HWT 032 - Node crusher124 - RT_GPU_ID 0 - GPU_ID 6 - Bus_ID d9
MPI 005 - OMP 000 - HWT 040 - Node crusher124 - RT_GPU_ID 0 - GPU_ID 7 - Bus_ID de
MPI 006 - OMP 000 - HWT 048 - Node crusher124 - RT_GPU_ID 0 - GPU_ID 0 - Bus_ID c1
MPI 007 - OMP 000 - HWT 056 - Node crusher124 - RT_GPU_ID 0 - GPU_ID 1 - Bus_ID c6
```

Ideal mapping



```
$ OMP_NUM_THREADS=1 srun -N1 -n8 -c1 --gpus-per-node=8 ./hello_jobstep | sort
MPI 000 - OMP 000 - HWT 000 - Node crusher124 - RT_GPU_ID 0,1,2,3,4,5,6,7 - GPU_ID 0,1,2,3,4,5,6,7 - Bus_ID c1,c6,c9,ce,d1,d6,d9,de
MPI 001 - OMP 000 - HWT 008 - Node crusher124 - RT_GPU_ID 0,1,2,3,4,5,6,7 - GPU_ID 0,1,2,3,4,5,6,7 - Bus_ID c1,c6,c9,ce,d1,d6,d9,de
MPI 002 - OMP 000 - HWT 016 - Node crusher124 - RT_GPU_ID 0,1,2,3,4,5,6,7 - GPU_ID 0,1,2,3,4,5,6,7 - Bus_ID c1,c6,c9,ce,d1,d6,d9,de
MPI 003 - OMP 000 - HWT 024 - Node crusher124 - RT_GPU_ID 0,1,2,3,4,5,6,7 - GPU_ID 0,1,2,3,4,5,6,7 - Bus_ID c1,c6,c9,ce,d1,d6,d9,de
MPI 004 - OMP 000 - HWT 032 - Node crusher124 - RT_GPU_ID 0,1,2,3,4,5,6,7 - GPU_ID 0,1,2,3,4,5,6,7 - Bus_ID c1,c6,c9,ce,d1,d6,d9,de
MPI 005 - OMP 000 - HWT 040 - Node crusher124 - RT_GPU_ID 0,1,2,3,4,5,6,7 - GPU_ID 0,1,2,3,4,5,6,7 - Bus_ID c1,c6,c9,ce,d1,d6,d9,de
MPI 006 - OMP 000 - HWT 048 - Node crusher124 - RT_GPU_ID 0,1,2,3,4,5,6,7 - GPU_ID 0,1,2,3,4,5,6,7 - Bus_ID c1,c6,c9,ce,d1,d6,d9,de
MPI 007 - OMP 000 - HWT 056 - Node crusher124 - RT_GPU_ID 0,1,2,3,4,5,6,7 - GPU_ID 0,1,2,3,4,5,6,7 - Bus_ID c1,c6,c9,ce,d1,d6,d9,de
```



Incorrect mapping: can result in all 8 MPI ranks targeting the same GPU.

Debugging Issues on a Compute Node

Terminal 1

```
$ salloc -A stf016 -t 60 -N 1
salloc: Pending job allocation 108355
salloc: job 108355 queued and waiting for resources
salloc: job 108355 has been allocated resources
salloc: Granted job allocation 108355

$ make
Current system: crusher
hipcc --amdgpu-target=gfx90a  -c p2p.cpp
hipcc --amdgpu-target=gfx90a  p2p.o -o p2p

$ srun -n8 -c8 --gpus-per-node=8 --gpu-bind=closest ./p2p --correct
...
...
```

Once on a compute node, you can use `rocm-smi`, `gdb`, `gstack`, etc.

Get process IDs from `ps -ef | grep <username>`, `top`, etc.

Terminal 2

```
$ squeue | grep tpapathe
108355     batch interact tpapathe  R      13:58      1 crusher001

$ ssh crusher001
$ hostname
crusher001

$ rocm-smi --showpids
================== ROCm System Management Interface ==================
============================ KFD Processes ============================
KFD process information:
PID     PROCESS NAME    GPU(s)    VRAM USED    SDMA USED    CU OCCUPANCY
65592   p2p             8         2197880832   0            0
===================================================================

$ rocm-smi --showmemuse
================== ROCm System Management Interface ==================
======================== Current Memory Use =========================
GPU[0]                  : GPU memory use (%): 1
GPU[0]                  : Memory Activity: 363340274
GPU[1]                  : GPU memory use (%): 2
GPU[1]                  : Memory Activity: 363699843
GPU[2]                  : GPU memory use (%): 0
GPU[2]                  : Memory Activity: 361751979
GPU[3]                  : GPU memory use (%): 4
GPU[3]                  : Memory Activity: 363537625
GPU[4]                  : GPU memory use (%): 0
GPU[4]                  : Memory Activity: 363009653
GPU[5]                  : GPU memory use (%): 0
GPU[5]                  : Memory Activity: 362536691
GPU[6]                  : GPU memory use (%): 0
GPU[6]                  : Memory Activity: 362110208
GPU[7]                  : GPU memory use (%): 0
GPU[7]                  : Memory Activity: 361753816
===================================================================
```

rocprof Tips

Making sure you actually ran on a GPU:

```
$ srun ... rocprof --stats ... ./a.out
```

Viewing trace files

- chrome://tracing
- https://ui.perfetto.dev/

rocprof output with MPI:

Creates 1 output file *per process*

```
$ cat rocprof_wrapper.sh
#!/bin/bash
rocprof --stats --hip-trace -o my_output_${SLURM_PROCID}.csv "$@"

$ srun ... ./rocprof_wrapper.sh ./a.out
```

Creates output file from specific process only

```
$ cat rocprof_wrapper_single.sh
#!/bin/bash
if [ ${SLURM_PROCID} -eq 0 ];then
    rocprof --stats --hip-trace -o my_output_${SLURM_PROCID}.csv "$@"
else
    "$@"
fi

$ srun ... ./rocprof_wrapper_single.sh ./a.out'
```

GPU HW Atomics

HW atomics can be much faster than sw-enabled atomics.

Importantly, users must explicitly request HW atomics via the `-munsafe-fp-atomics` flag. Otherwise, atomic operations will be performed via CAS loops.

As the name of the flag implies, this might be unsafe to do in some situations, and this is determined by the "granularity" of memory allocated by the user. The "granularity" in this case refers to the coherency between the memory being used inside a kernel and the memory in the rest of the system.

- With *coarse-grained* memory, coherence with the rest of the system (CPU and GPUs) is obtained at synchronization points. (e.g., `hipDeviceSynchronize`)
- *Fine-grained* memory allows CPU and GPU (and multiple GPUs) to synchronize while the GPU kernel is running.

Only coarse-grained memory can use HW atomics. If HW atomics are requested on fine-grained memory, they will (silently) produce a no-op (i.e., give incorrect results).

We're currently working w/AMD to provide training & docs on this topic.

Managed Memory – XNACK

Works by using either zero-copy memory access or page migration on page fault. To understand which will be used, users need to know about XNACK.

XNACK allows AMD GPUs to migrate memory pages (allocated in a unified virtual address space) between the CPU and GPU upon page-fault.

Importantly, support for XNACK must be enabled at both compile-time and run-time.

- A code that is compiled with xnack support **and** runs in an environment with XNACK enabled, will migrate pages between the CPU and GPU on page-fault.

- A code that *is not* compiled with xnack support and *does not* run in an environment with XNACK enabled, will still function as expected with managed memory, but the CPU will access the GPU's memory in a zero-copy fashion (CPU reads directly from GPU memory), and vice versa.

XNACK is not enabled by default at runtime, so users unaware of this might find lower-than-expected performance (using managed memory) and not realize why.

xnack is enabled at *compile-time* by default, but to use page-migration XNACK needs to be enabled at *run-time* with `HSA_XNACK=1`

We're currently working w/AMD to provide training & docs on this topic.

Helpful Resources

OLCF Support: help@olcf.ornl.gov

Crusher Quick-Start Guide:

https://docs.olcf.ornl.gov/systems/crusher_quick_start_guide.html

- Known Issues: https://docs.olcf.ornl.gov/systems/crusher_quick_start_guide.html#known-issues
- SW Compatibility: https://docs.olcf.ornl.gov/systems/crusher_quick_start_guide.html#determining-the-compatibility-of-cray-mpich-and-rocm

ECP Frontier COE Confluence:

https://confluence.exascaleproject.org/display/FCOE/Frontier+Center+of+Excellence

- Forum to ask questions of the community – including vendors.

Crusher Office Hours (every Monday):

https://www.olcf.ornl.gov/crusher-office-hours/

- Must sign up, 5 teams per office hour

```
$ man intro_
intro_blacs       intro_blas3       intro_craype-api  intro_irt         intro_libsci      intro_PAPI        intro_scalapack
intro_blas1       intro_cblas       intro_dsmml       intro_lapack      intro_mpi         intro_perftools
intro_blas2       intro_craypat     intro_hugepages   intro_lapacke     intro_papi        intro_pmi
```

Questions?

Slurm Tips – Test CPU/GPU Binding Before Running

```
[crusher: ~]$ module load rocm

[crusher: ~]$ OMP_NUM_THREADS=1 srun -l -N1 -n8 -c8 --gpus-per-node=8 --gpu-bind=closest /bin/bash -c 'echo $(hostname)
$(grep Cpus_allowed_list /proc/self/status) GPUS: $ROCR_VISIBLE_DEVICES' | sort

0: crusher076 Cpus_allowed_list: 0-7 GPUS: 4
1: crusher076 Cpus_allowed_list: 8-15 GPUS: 5
2: crusher076 Cpus_allowed_list: 16-23 GPUS: 2
3: crusher076 Cpus_allowed_list: 24-31 GPUS: 3
4: crusher076 Cpus_allowed_list: 32-39 GPUS: 6
5: crusher076 Cpus_allowed_list: 40-47 GPUS: 7
6: crusher076 Cpus_allowed_list: 48-55 GPUS: 0
7: crusher076 Cpus_allowed_list: 56-63 GPUS: 1
```

This command gives some broad details about the CPUs and GPUs your job step's processes have access to.

Clang, Clang Wrappers, and Cray's Compiler Wrappers

Vendor	Programming Environment	Compiler Module	Language	Compiler Wrapper	Compiler
Cray	`PrgEnv-cray`	`cce`	C	`cc`	`craycc`
			C++	`CC`	`craycxx` or `crayCC`
			Fortran	`ftn`	`crayftn`
AMD	`PrgEnv-amd`	`rocm`	C	`cc`	`amdclang`
			C++	`CC`	`amdclang++`
			Fortran	`ftn`	`amdflang`
GCC	`PrgEnv-gnu`	`gcc`	C	`cc`	`${GCC_PATH}/bin/gcc`
			C++	`CC`	`${GCC_PATH}/bin/g++`
			Fortran	`ftn`	`${GCC_PATH}/bin/gfortran`

`-craype-verbose`

NOTE: It is highly recommended to use the Cray compiler wrappers whenever possible, but if for some reason you don't want to use the wrappers…

- Cray compilers, use `craycc`, `crayCC`, and `crayftn` (don't use Cray's bare Clang compilers)
- AMD compilers, use `amdclang`, `amdclang++`, and `amdflang` (don't use AMD's bare Clang compilers)

Another Way to Check if GPU is Being Used

Making sure you actually ran on a GPU (Cray OpenMP Offload only!)

```
$ CRAY_ACC_DEBUG=1 srun -A stf016 -t5 -N1 -n1 -c8 --gpus-per-node=8 --gpu-bind=closest ./vAdd_ompGPU

srun: job 108498 queued and waiting for resources
srun: job 108498 has been allocated resources

ACC: Transfer 3 items (to acc 805306368 bytes, to host 0 bytes) from vAdd_ompGPU.cpp:30
ACC: Execute kernel __omp_offloading_69_158925e9_main_l30_cce$noloop$form from vAdd_ompGPU.cpp:30
ACC: Transfer 3 items (to acc 0 bytes, to host 268435456 bytes) from vAdd_ompGPU.cpp:30

Test passed.
Result              = 1.0000000000000000
Tolerance           = 0.0000000000000100
Array buffer size   = 268435456
Elapsed time (s)    = 0.915606
```