NOVEMBER 16, 2026

Introduction to NSF NCAR HPC

Derecho and Casper background, technology overview, and running on HPC systems

Brett Neuman, NSF NCAR / CISL / HPCD



Agenda

1. **Systems**
   Derecho, Casper, access, services
2. **Storage**
   GLADE and login nodes
3. **Agentic AI**
   Assistants on shared systems
4. **PBS jobs**
   Scripts, queues, charges
5. **Software**
   Modules and conda
6. **Compile & run**
   MPI, OpenMP, GPUs

Systems · Two systems, two jobs

Derecho

HPE Cray EX supercomputer, installed 2023

COMPUTE NODES

2,570

- 2,488 CPU nodes: 2 × 64-core AMD Milan, 235 GB usable
- 82 GPU nodes: 64-core Milan, 4 × A100 40 GB, 487 GB
- 323,712 cores, 692 TB memory, 19.87 PF peak
- HPE Slingshot 11 network, 200 Gb/s per port
- 60 PB Lustre scratch at 300 GB/s
- 8 login nodes, 2 of them with A100s

Casper

Heterogeneous analysis and visualization cluster

COMPUTE NODES

~180

- 126 high-throughput nodes, 34 to 62 cores
- 8 large-memory nodes, up to 1.5 TB
- 18 visualization nodes: GP100, A100 or L40
- 22 dense GPU nodes: V100, A100, H100, MI300A
- Shared nodes: pay only for what you request
- 3 login nodes

NCAR's High-Performance Computing, Data, & Analysis Resources

HPC Systems

Installed mid-2023



Cray/HPE
2570 Nodes, ***323,712 CPU Cores****, 692 TB total memory*
328 NVidia A100 GPUs *provides 20% overall performance,* ***19.87 PFlop/s***



NCAR Wyoming Supercomputing Center (NWSC)
Collaboration between NSF NCAR and University of Wyoming

Visit the on site museum or
Virtual 360° Tour!



Data Analysis & Visualization



Casper: *heterogeneous system for data analysis & viz.*
- 126 High-Throughput Computing nodes
- 18 visualization nodes with accelerated graphics
- 4 nodes for GDEX Research Data processing
- 8 1.5TB large memory nodes
- 22 dense GPU nodes for GPGPU or AI/ML
- Includes V100, A100, H100, and MI300A (AMD)

projectpythia.org

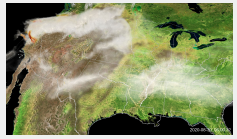
PROJECT PYTHIA

OPEN OnDemand

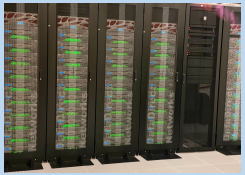
CISL develops specialized visualization software & services for Earth Science applications

GeoCAT
geocat.ucar.edu

VAPOR Visualization & Analysis Platform
vapor.ucar.edu



High Performance Storage



GLADE & Campaign Storage
- **150PB** long-term, online storage
- 17,464 hard drives
- 56 servers

gdex.ucar.edu



Derecho 'scratch' Storage
- **60PB** short-term storage
- Principally supports HPC jobs
- 5,088 hard drives
- 24 servers

Stratus Object Storage
- **5PB object storage** system
- 588 hard drives
- 6 servers

Quasar Tape Library
- **35PB long term archival** storage
- 22 IBM TS1160 tape drives
- 1774 20TB tape cartridges
- 216 hard drives
- 2PB disk cache
- 5 data mover servers



Kubernetes Cloud

CIRRUS
NSF NCAR Cloud Infrastructure for Remote Research, Universities and Scientists

Kubernetes-based cloud platform hosted at NWSC & the Mesa Lab in Boulder

Includes S3-compatible object storage
- Provides access to buckets and objects rather than POSIX compliant filesystem
- **246 TB** of storage across the system
- **18 nodes total**, with 5 nodes hosting one NVIDIA A2 GPU, 5 nodes with one A10 GPU

Systems · Logging in with SSH

```
ssh -Y username@derecho.hpc.ucar.edu
ssh -Y username@casper.hpc.ucar.edu
ssh username@cron.hpc.ucar.edu
```

- **Derecho** puts you on 1 of 8 login nodes, **Casper** on 1 of 3; **cron** is for crontab workflows
- At `ncar-two-factor:` type your CIT password (nothing echoes), then approve the Duo push
- `-Y` forwards X11 for simple GUIs; `$HOME`, `$WORK` and `$SCRATCH` are the same on both systems
- Login problems: the NSF NCAR help desk, help.ucar.edu

Systems · Which system for which work?

Use Derecho for
Big, tightly coupled runs

- MPI models: CESM, WRF, MPAS, MOM6
- Jobs that fill whole nodes: 128 cores or 4 GPUs
- Multi-node A100 training and GPU codes
- Production runs, up to 12 h (100 h on `preempt`)

Use Casper for
Everything around the run

- Post-processing: Xarray, Dask, NCO, CDO
- Jupyter, GUIs and remote desktops
- Single-GPU ML, GPU benchmarking
- Large-memory serial work, HTC job arrays

Systems · Node types at a glance

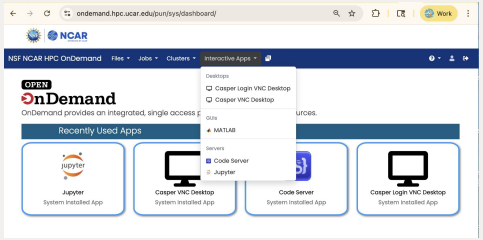
SYSTEM	NODE TYPE	NODES	CORES	MEMORY	GPUS PER NODE	RUNS IN
Derecho	CPU	2,488	128	235 GB	none	`cpu`
Derecho	GPU	82	64	487 GB	4 × A100 40 GB	`gpu`
Casper	High-throughput	126	34 or 62	354 or 733 GB	none	`htc`
Casper	Large memory	8	36 or 62	1.5 TB	none	`largemem`
Casper	Visualization	18	36 or 48	384 or 733 GB	1 × GP100, A100 40 GB or L40	`vis`
Casper	ML & GPGPU	20	36 to 128	768 to 1,152 GB	4–8 × V100, 4 × A100 or H100	`nvgpu`
Casper	AMD APU	2	96	470 GB	4 × MI300A	`amdgpu`

Memory is what a job can use; select statements per type: the docs' "Casper node types".

Systems · Beyond the terminal

Open OnDemand
ondemand.hpc.ucar.edu

- Shells, JupyterLab, VNC desktops, MATLAB
- Code Server, file browser, job viewer



JupyterHub
jupyterhub.hpc.ucar.edu

- Casper login, Casper batch, Derecho
- Retiring by the end of 2026
- OnDemand's Jupyter app replaces it

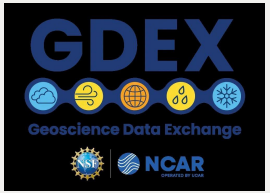
VS Code
Three ways in

- **Remote-SSH** lands on a login node
- `qvscode` (Casper): moves the session to a compute node
- **Code Server** app in OnDemand

Systems · Services around the systems

GDEX
Geoscience Data Exchange

- NSF NCAR's curated, FAIR, CoreTrustSeal data commons
- Reanalyses (ERA5…), model output, AI-ready sets
- On GLADE at `/glade/gdex/data`; or THREDDS, APIs, HTTP
- gdex.ucar.edu



CIRRUS
NSF NCAR's Kubernetes cloud

- Hosted at NWSC and the Mesa Lab
- Web services, science gateways, long-running apps
- S3-compatible object storage; complements HPC
- CIRRUS documentation



Storage · GLADE spaces and `gladequota`

SPACE	PATH	QUOTA	BACKUP	PURGED	FOR
Home	`/glade/u/home/$USER`	50 GB	yes	never	scripts, source, settings
Work	`/glade/work/$USER`	2 TB	no	never	builds, conda envs
Scratch	`/glade/derecho/scratch/$USER`	30 TB	no	180 days unread	run directories, output
Campaign	`/glade/campaign/<project>`	allocated	no	90 days after it ends	long-term project data

- **`gladequota`** shows your use of each: e.g. work 1.48 of 2.00 TiB (74%), home 84.6 of 100 GiB (85%)
- Scratch also caps you at 10 M files; move data with Globus, `scp` or `rsync`

Storage · Login nodes vs. compute nodes

Login nodes
Shared by everyone logged in

- Edit code and scripts
- Small compiles, data transfers
- Submit and watch jobs

⚠ Arbiter2 throttles past **8 cores** or **10 GB**, and emails you

Compute nodes
Yours for the job

- Model runs, big builds, test suites
- Analysis of large data, ML training
- Interactive work: `qinteractive -A <project>`
- One command: `qcmd -A <project> -- make -j 8`

† Also be kind to GLADE: no `find`, `du` or `ls -R` across shared trees such as `/glade/campaign`.

Agentic AI · You own what your assistant runs

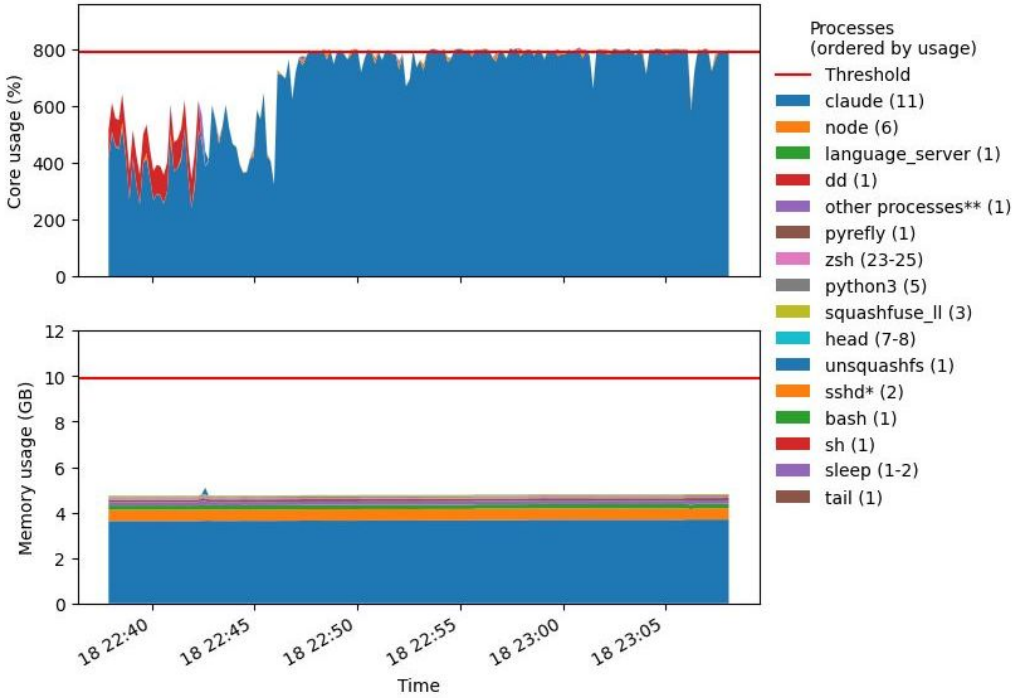
Do
Claude Code, Codex, Gemini CLI, Copilot

- ✓ Approve each command; read job scripts and `qsub`s
- ✓ Send heavy work to compute nodes: `qcmd`, `qinteractive`
- ✓ Confine it to one project directory
- ✓ Put temporary files in `$SCRATCH`
- ✓ Check your allocation and quotas afterwards

Don't
On login nodes and GLADE

- ✗ Use auto-accept or "run without asking" modes
- ✗ Let it `find`, `grep -r` or `du` from `/glade` roots
- ✗ Let it `chmod 777` or bulk-create files
- ✗ Run it beside SSH keys, tokens or restricted data
- ✗ Let it start a large experiment unreviewed

Agentic AI · A runaway assistant on a login node



Arbiter2's view of one login session: 11 `claude` processes held it at the 8-core (800%) limit for 20 minutes.

Agentic AI · Give it the rules: `AGENTS.md`

```
# Working on NSF NCAR HPC shared systems
- Never search or recurse at /glade, /glade/work,
  /glade/u/home or /glade/derecho/scratch roots
- Heavy work goes to a compute node:
  qcmd -A <PROJECT> -- make -j 4
- Show me every qcmd/qsub before submitting
- Cap builds at make -j 2 to 4
- Big output to scratch or work, never $HOME
- Never read SSH keys, tokens or passwords
- Use module load; no system packages
```

- A short file at the project root that most assistants read automatically
- `AGENTS.md` works across tools; Claude Code reads `CLAUDE.md`, which can import it
- The docs give a complete template to copy

† Using Agentic Coding Assistants, in the docs' Best Practices section.

PBS jobs · Anatomy of a batch script

The script

my_run.pbs, submitted with qsub

```
#!/bin/bash
#PBS -N my_run
#PBS -A <project_code>
#PBS -q main
#PBS -l walltime=01:00:00
#PBS -l select=2:ncpus=128:mpiprocs=128
#PBS -j oe

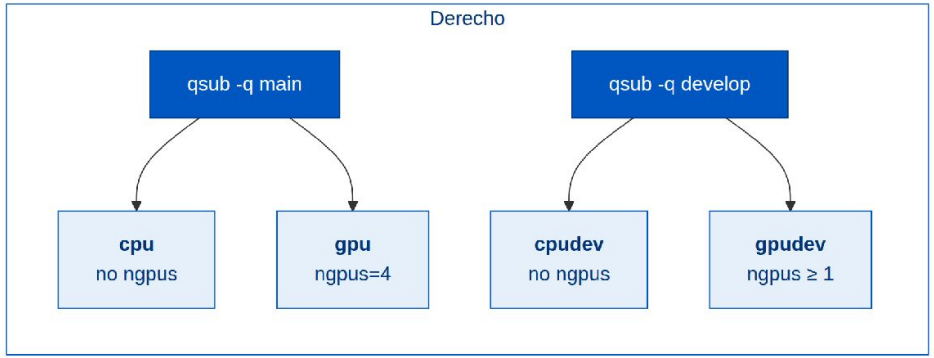
module reset
mpiexec -n 256 -ppn 128 ./model.exe
```

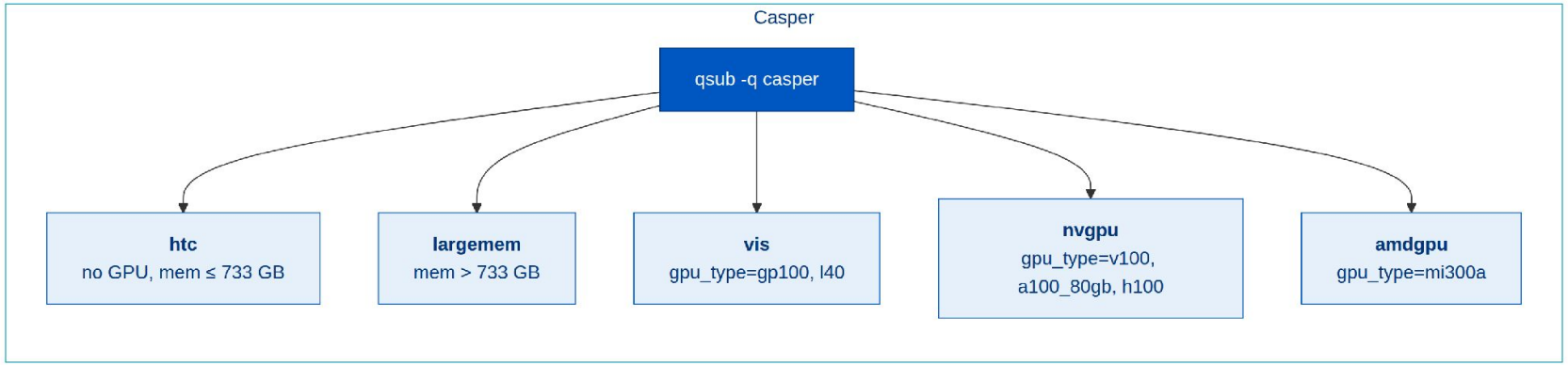
What each line does

Every #PBS line also works on the qsub command line

- `-N`: the job's name
- `-A`: the project allocation to charge
- `-q`: the queue to submit to
- `-l walltime`: time limit; you pay for time used
- `-l select`: 2 nodes, 128 cores and 128 MPI ranks each
- `-j oe`: stdout and stderr in one file
- Then: load your modules and launch

PBS jobs · How your job is routed





PBS jobs · Derecho queues

SUBMIT TO	YOUR `select` ASKS FOR	RUNS IN	WALLTIME	LIMITS	NODES
`main`	no `ngpus`	`cpu`	12 h	2,488 nodes; 128 cores + 235 GB each	exclusive
`main`	`ngpus=4`	`gpu`	12 h	82 nodes; 4 GPUs + 487 GB each	exclusive
`develop`	no `ngpus`	`cpudev`	6 h	256 cores per user	shared
`develop`	`ngpus` ≥ 1	`gpudev`	6 h	8 GPUs per user	shared
`preempt`	either	`pcpu`, `pgpu`	100 h	as `main`; killed for higher-priority work	exclusive

- **Use `develop` for** compiling, debugging, short tests and interactive sessions
- **`develop` limits**: 6 h walltime; at most 256 cores or 8 GPUs running per user; job arrays of 4 indices
- **Not for** production runs, or to skip the wait in `main`: it is small and shared by everyone

PBS jobs · Casper queues

SUBMIT TO	YOUR `select` ASKS FOR	RUNS IN	WALLTIME	LIMITS
`casper`	no GPU, `mem` ≤ 733 GB	`htc`	24 h	144 cores per job; 720 cores, 7,200 GB per user
`casper`	`mem` > 733 GB	`largemem`	24 h	1 node; 3 running jobs
`casper`	`gpu_type=gp100` or `l40`	`vis`	24 h	GPU rendering and VNC; 2 GPUs
`casper`	`gpu_type=v100`, `a100_80gb`, `h100`	`nvgpu`	24 h	32 GPUs (8 H100)
`casper`	`gpu_type=mi300a`	`amdgpu`	24 h	8 GPUs; whole nodes

Shared nodes: ask for exactly what you need, e.g. `select=1:ncpus=4:mem=40gb:ngpus=1:gpu_type=a100_80gb`

PBS jobs · How a job is charged

Derecho `main` and `preempt` charge whole nodes

1 core requested on 1 node still costs 128 core-hours an hour. Fill the node.

- **Exclusive** (Derecho `main`, `preempt`): hours × nodes × 128 cores (or 4 GPUs) × priority factor
- **Shared** (Casper, Derecho `develop`): hours × cores (or GPUs) you requested
- GPU jobs draw GPU-hours only; CPU jobs core-hours
- Charged for walltime **used**, not requested

JOB_PRIORITY	FACTOR
`premium`	1.5
`regular`	1.0
`economy`	0.7
`preempt` queue	0.2

PBS jobs · Submit, watch, and cancel

`qsub`
Submit a job

```
$ qsub cpu_demo.pbs
8129266.desched1
$ qsub -q casper@casper-pbs gpu_casper.pbs
6589483.casper-pbs
```

`qdel`: cancel a job by its ID

```
$ qdel 8129267
```

`qstat`
Watch your jobs

```
$ qstat -u $USER
$ qstat -u $USER @casper
6589483.casper* bneuman nvgpu gpu_casper
  -- 1 4 40gb 00:10 Q --
```

- `Q` queued, `R` running
- `qstat -x` adds recently finished jobs

PBS jobs · Look back with `qhist`

```
$ qhist -u $USER -d 30
Job ID       User       Queue    Nodes  NCPUs NGPUs     End      Mem    CPU   Elap
------------ ---------- -------- ----- ------ ----- ------- -------- ------ ------
8018932      bneuman    cpudev       1      8     0 30-1623    13.70  12.38   1.01
8019294      bneuman    cpudev       1      8     0 30-1806    48.04  12.38   2.03
8047922      bneuman    M8039202     1    128     0 02-1000     0.25   0.37   0.01
8048040      bneuman    M8039202     2    256     0 02-1120     0.26  54.87   1.22
```

- Memory (GB), CPU use (%) and elapsed hours per job: right-size your next request
- `qhist -l` prints every field; `qhist -j <id>` one job

Software · Module commands

COMMAND	DOES
`module avail` (`ml av`)	what you can load **now**, given what is loaded
`module list` (`ml`)	what is loaded
`module load netcdf` (`ml netcdf`)	load; `module unload` removes
`module swap intel gcc/14.3.0`	change compiler: the libraries below it follow
`module spider esmf`	search **every** branch, and how to reach it
`module show netcdf`	what a module sets
`module reset`	back to the defaults (`ncarenv/25.10`)

Software · The module tree

1

`ncarenv/25.10`

The stack: loaded at login, sticky

2

Core

Compilers, `cuda`, `conda`, `nco`, `cdo`, `cmake`

3

Compiler-dependent

`netcdf`, `hdf5`, `fftw`, `esmf`, MPIs

4

MPI-dependent

`netcdf-mpi`, `parallelio`, `hdf5-mpi`

† Each level appears in `module avail` only once the level above is loaded; `module spider` finds every branch.

Software · Python with conda

```
module load conda
conda activate npl-2025b

conda create -n my-env python=3.12 \
    xarray dask netcdf4 ipykernel
conda activate my-env
python -m ipykernel install \
    --user --name=my-env
```

- **`module load conda`**: never install your own miniconda on GLADE
- **NCAR Python Library** (`npl-2025a`, `npl-2025b`): a ready Earth-science stack with Xarray, Dask, MetPy, Cartopy, GeoCAT, UXarray, xESMF
- **Kernels**: `npl` is in Jupyter on OnDemand and JupyterHub; `ipykernel install` adds your own
- **uv** for pure-PyPI projects: `conda deactivate; module load uv`

Compile & run · Compilers and wrappers

COMPILER	MODULE	SERIAL: C / FORTRAN	MPI	OPENMP	GPU OFFLOAD
Intel oneAPI	`intel`	`icx` / `ifx`	`mpicc`, `mpif90`	`-qopenmp`	OpenMP `-fopenmp-targets=…`
GNU	`gcc`	`gcc` / `gfortran`	`mpicc`, `mpif90`	`-fopenmp`	`-fopenacc`
NVIDIA HPC SDK	`nvhpc`	`nvc` / `nvfortran`	`mpicc`, `mpif90`	`-mp`	`-acc`, `-mp=gpu`, CUDA
Cray CCE	`cce`	`cc` / `ftn`	`mpicc`, `mpif90`	`-fopenmp`	`-h acc` (Fortran)

`ncarcompilers` adds include and library flags for loaded modules: `ifx model.f90 -lnetcdff` just works.

On Derecho use `-march=core-avx2`, never `-xHost`.

Compile & run · Building the examples

```
$ module reset && mpicc --version | head -1
Intel(R) oneAPI DPC++/C++ Compiler 2025.2.1 (2025.2.0.20250806)
$ mpicc -O2 -o hello_mpi hello_mpi.c
$ mpicc -O2 -qopenmp -o hello_hybrid hello_hybrid.c
```

```
$ module load nvhpc cuda
$ nvfortran -acc -Minfo=accel -gpu=cc80 -o saxpy_acc saxpy_acc.f90
saxpy:
      7, Generating copy(y(:)) [if not already present]
         Generating copyin(x(:)) [if not already present]
         Generating NVIDIA GPU code
          8, !$acc loop gang, vector(128) ! blockidx%x threadidx%x
$ mpicc -O2 -o hello_gpu hello_gpu.c -lcudart
```

Compile & run · Pure MPI

```
#PBS -q main
#PBS -l select=2:ncpus=128:mpiprocs=128

mpiexec -n 256 -ppn 128 \
    --cpu-bind core ./hello_mpi
```

- `-n` total ranks, `-ppn` ranks per node: match the `select` line
- `--cpu-bind core` pins each rank to a core

```
$ cat cpu_demo.o8129266
rank    0 of    8 on dec2433
rank    1 of    8 on dec2433
rank    2 of    8 on dec2433
rank    3 of    8 on dec2433
rank    4 of    8 on dec2434
rank    5 of    8 on dec2434
rank    6 of    8 on dec2434
rank    7 of    8 on dec2434
```

† Output from a small `develop` run: 2 nodes × 4 ranks.

Compile & run · Hybrid MPI + OpenMP

```
#PBS -l select=2:ncpus=128:mpiprocs=32:ompthreads=4

export OMP_PLACES=threads
export OMP_PROC_BIND=close

mpiexec -n 64 -ppn 32 -d 4 \
    --cpu-bind depth ./hello_hybrid
```

- `-d 4`: 4 threads per rank; `depth` keeps them together
- Or `mpibind ./hello_hybrid` reads the `select` line for you

```
$ cat cpu_demo.o8129266
rank 0 thread 0/2 on dec2433 core  96
rank 0 thread 1/2 on dec2433 core  98
rank 1 thread 0/2 on dec2433 core  99
rank 1 thread 1/2 on dec2433 core 100
rank 2 thread 0/2 on dec2434 core  48
rank 2 thread 1/2 on dec2434 core  49
rank 3 thread 0/2 on dec2434 core  50
rank 3 thread 1/2 on dec2434 core  51
```

† Output from `develop`: 4 ranks × 2 threads.

Compile & run · GPUs, with and without MPI

```
#PBS -q main
#PBS -l select=2:ncpus=64:mpiprocs=4:ngpus=4

module load nvhpc cuda
export MPICH_OFI_NIC_POLICY=GPU
./saxpy_acc
mpiexec -n 8 -ppn 4 \
    set_gpu_rank ./hello_gpu
```

- `set_gpu_rank`: one GPU per rank
- `MPICH_OFI_NIC_POLICY=GPU`: each rank on its GPU's nearest NIC

```
$ ./saxpy_acc
n = 67108864   y(1) =   4.00
$ mpiexec -n 4 set_gpu_rank ./hello_gpu
rank 0: GPU 0, A100 80GB, bus 01
rank 1: GPU 1, A100 80GB, bus 41
rank 2: GPU 2, A100 80GB, bus 81
rank 3: GPU 3, A100 80GB, bus c1
```

† Output from Casper's casper39, sent from Derecho with `-q casper@casper-pbs`; lines trimmed.

Getting help

- **Docs**: ncar-hpc-docs.readthedocs.io
- **Tickets**: Research Computing help desk, rchelp.ucar.edu, (303) 497-2400
- **Community**: the NHUG Slack, ncar.pub/join_nhug_slack
- **Full tutorial**: *Introduction to NSF NCAR HPC for Tutorials*, tinyurl.com/NCAR-HPC-Training

- **Hands-on**: `git clone https://github.com/NCAR/Intro-to-HPC.git`
- **Acknowledge** Derecho in papers: doi:10.5065/qx9a-pg09
- **Docs fixes**: the pencil icon, or issues at github.com/NCAR/HPC-Docs
- **SAMuel**: Track your jobs, projects, and system status sam.hpc.ucar.edu



Thank you!

Questions: rchelp.ucar.edu

Docs: ncar-hpc-docs.readthedocs.io