NVIDIA multi-node Grace-Hopper and Grace-Grace Evaluation System (Thea)

ChangeLog

Version: v2.0.0
Last update: 27/01/2025

- The `init_on_alloc` setting is `on` for security reasons. This may have performance implications.
- The `perf_event_paranoid` setting is `0` for security reasons. This may have implications when PMUs are collected.
- CPU and GPU power measurements at the level of Superchip are accessible via `nvidia-smi` (GPU) or `hwmon` (CPU).

System Access

Login is password-less and enabled after providing a public SSH key and a public IP from which you will connect. Once these information are provided, direct SSH is possible simply by typing the following command:

```
ssh <username>@gw.hpcadvisorycouncil.com
```

NOTE - Please provide a fixed public IP as the firewall setting is a manual task our sysadmins need to perform. If your IP change, please contact THEA-INFRA team

System Architecture

Thea is seamnglessly integrated in the HPC Advisory Council infrastructure. There is one single entry point and one slurm instance. Multiple "clusters" are de-facto separated slurm partitions.

Full technical GH200 and Grace CPU Superchip specs can be found in the NVIDIA Grace Performance Tuning Guide.

NOTE - Access to partitions is granted based on business requests and approval from NVIDIA Enterprise Product Team.

Hardware: Compute

- **CPU+GPU**: Quanta S74G-2U GH200
  - Grace CPU 72 cores @ 3.4GHz with 480 GB LPDDR5X
  - Hopper GPU with 96 GB HBM3
  - Infiniband NDR400, 1 HCA per node
- **CPU-only**: Supermicro ARS-221GL-NR
  - Dual Grace CPU 72 cores @ 3.4GHz with 480 GB LPDDR5X
  - Infiniband NDR400, 1 HCA per node (attached to socket 0)

Hardware: Storage

- **Home**: `/global/home/users/$USER` (`${HOME}`)
  - Provided via NFS
  - Quota enforced (~20 GBytes)
- **Scratch DDN**: `/global/exafs/users/${USER}`
  - Use for capability purposes (e.g. parallel I/O, checkpoints, container execution)
  - Provided by DDN Exascale NVMe appliance (no backup)
  - Accessible only by all `gh` , `gg` and `ggcompile` nodes
  - Accessible using the environment variable `${SCRATCH_DDN}`
- **Scratch OSS**: `/global/scratch/users/${USER}`
  - Use for capacity purposes (e.g. store/stage large files, transfer data in-out of the cluster)
  - Provided via Open Source Lustre (no backup)
  - Accessible using the environment variable `${SCRATCH_OSS}`
- **Local**: `/local`
  - Provided via NVMe local disk, automatically purged when a job ends
  - Usable cpacity ~830 GBytes

NOTE - Please use `${SCRATCH_DDN}` for I/O purposes.

Software: Base System OS config

All Thea partitions (`gh` , `gg` and `ggcompile`) boot the same compute image:

- Rocky Linux 9.4 with a custom hybrid 6.5.0-1019-nvidia-64k kernel (64k pages enabled)
- CUDA driver 555.42.06 (only on `gh`)
- MLNX_OFED_LINUX-24.04-0.7.0.0
- GNU GCC 11.4.1 (other compilers available via modulefiles)
- ldd (GNU libc) 2.34

Compilation node

To improve the user experience and allow many users concurrently to build their software, we installed a Grace Superchip compilation node that can be used to build both CPU and CPU+GPU software.

The node can access all filesystems (`${HOME}` , `${SCRATCH_DDN}` and `${SCRATCH_OSS}`) plus a local NVMe disk under `/local` . This node is in a special partition called `ggcompile` and oversubscription is allowed. Doing the following to access the node:

```
salloc -p ggcompile -N1 -n144 -t 4:00:00 --oversubscribe
```

NOTE - All login nodes available are x86. To compile codes for aarch64 you need to allocate interactively one compute node.

NOTE - This node has pre-production hardware **NOT** for performance measurements.

Software Environment

Once on the compilation node it is possible to access curated modulefiles by run the following command:

```
. /global/exafs/groups/gh/bootstrap-env.sh
```

Spack has been used to build software with Arm Neoverse V2 optimizations and modern base compilers (GCC 13.3).

To avoid to build tons of extra software centrally, we only provide a minimal set pre-installed compilers and basic libraries (see listed below). A new pre-built software bundle will be deployed every time a new NVHPC SDK is released (the `202501` bundle is associated to the release of NVHPC 25.1) and it contains the currewnt and previous version of NVHPC SDK. Defaul GCC is aligned to which version is supporrted by CUDA.

Module	Description
`gcc/13.3.0-gcc-11.4.1`	GNU 13.3.0 (suggested default)
`acfl/24.10.1-gcc-13.3.0`	Arm HPC Compiler 24.10 (and associated Arm Performance Library
`nvhpc/25.1-gcc-13.3.0`	NVIDIA HPC SDK 25.1 with CUDA-aware MPI (provided by HPCX)
`nvhpc/24.11-gcc-13.3.0`	NVIDIA HPC SDK 24.11 with CUDA-aware MPI (provided by HPCX)
`ucx/1.18.0-gcc-13.3.0`	UCX 1.18.0 built for GNU (cma and xpmem shm transport + IB)
`openmpi/5.0.6-gcc-13.3.0`	OpenMPI 5.0.6 built for GNU (uses UCX)
`armpl-gcc/24.10-gcc-13.3.0`	Arm Performance Library 24.10 for GNIU
`openblas/0.3.29-gcc-13.3.0`	OpenBLAS 0.3.29 built for GNU 13.3.0 (multi-thtead support)
`openblas/0.3.29-nvhpc-25.1`	OpenBLAS 0.3.29 built for NVHPC 25.1 (multi-thtead support)
`openblas/0.3.29-nvhpc-24.11`	OpenBLAS 0.3.29 built for NVHPC 21.11 (multi-thtead support)
`fftw/3.3.10-gcc-13.3.0`	FFTW 3.3.10 built for GNU 13.3.0 (no MPI)
`fftw/3.3.10-nvhpc-25.1`	FFTW 3.3.10 built for NVHPC 24.11 (no MPI)
`fftw/3.3.10-nvhpc-24.11`	FFTW 3.3.10 built for NVHPC 24.11 (no MPI)
`nvpl/24.7-gcc-13.3.0`	NVIDIA Performance Library 24.7 (provides BLAS, LAPACK, FFTW and ScaLAPACK)
`cuda/12.5.1-gcc-13.3.0`	CUDA 12.5.1 (GNU 13.3.0 as host compiler)
	UCX 1.18.0 built for GNU (cma and

`ucx/1.18.0-gcc-13.3.0-cuda-12.5.1`	xpmem shm transport + IB) + GPU DIRECT
`openmpi/5.0.6-gcc-13.3.0-cuda-12.5.1`	CUDA-aware OpenMPI 5.0.6 built for GNU (uses UCX for GPU)
`papi/7.1.0-gcc-13.3.0`	PAPI 7.1.0

We advise users to build extra libraries and applications in `${SCRATCH_DDN}` either from source or by chaining local spack to the centrally provided modules and built libraries / compilers. it is also possible to build singularity environments.

NOTE - `modulefiles` tool is configured to auto-load dependencies. If some key software is missing please ask, we may be able to build it *on-the-fly* using spack.

NOTE - Please always use `gcc/13.3.0-gcc-11.4.1` instead of default GCC provided by the OS.

Data Management

As explained above, `${SCRATCH_DDN}` is only available on the aarch64 nodes (`gg` , `gh` , `ggcompile`). `${SCRATCH_OSS}` is available everywhere.

As result, data needs to be explicitly moved from `${SCRATCH_OSS}` to `${SCRATCH_DDN}` if fast storage is required. This operation can be done interactively on the `ggcompile` node, inside a submission script during job launch or in advance by submitting a job on `ggcompile` that only move data. Below an example of script:

```
#!/bin/bash -l
#SBATCH --ntasks=1
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=1
#SBATCH --nodes=1
#SBATCH --partition= ggcompile
#SBATCH --time=4:00:00
#SBATCH --oversubscribe

# Copy directory DATA in the fast storage maintaining same directory name
rsync -auvx --progress ${SCRATCH_OSS}/DATA ${SCRATCH_DDN}/
```

NOTE - We advice to use either local storage of `${SCRATCH_DDN}` for every I/O intensive operations such as: parallel read/write with HDF5, container bootstrap and build, compilation of large codebases.

Submit and Running Jobs

Slurm is the system job scheduler. Each job has a maximum walltime of 12 hours and nodes are allocated by default in *exclusive* mode (one user allocating always a full node, no sharing). GPU is always visible once a job allocated a node, no need to use any `gres` options.

At present, walltime is limited to 4 hours maximum.

NOTE - Please be mindful of other users when set max job walltime for a job submission, especially for interactive session. Use the `-t` option accordingly. If you need to extend job duration or schedule a reservation for a long job execution, please contact THEA-SUPPORT team.

ALERT - The system has limited numbers of nodes on the `gh` andf `gg` partitions. Jobs are left running without activity will be automatically killed by the administrator without warning. If you use `gg` or `gh` nodes in interactive mode, make sure to end your job/session when work is done. Users that will repetitively leave idle jobs hanging will be temporarily disabled.

Interactive jobs

Example how to allocate a GH200 node

```
salloc -n 72 -N 1 -p gh -t 1:00:00
```

Example how to allocate 2 GH200 nodss

```
salloc -n 144 -N 2 -p gh -t 1:00:00
```

Example how to allocate a specific GH200 node

```
salloc -n 72 -N 1 -p gh -w gh004 -t 1:00:00
```

Example how to allocate 2 specific GH200 nodes

```
salloc -n 144 -N 2 -p gh -w gh002,gh004 -t 1:00:00
```

Example how to allocate 4 GH200 nodes and exclude a specific one

```
salloc -n 288 -N 4 -p gh -x gh001 -t 1:00:00
```

Example how to allocate a Grace-only node

```
salloc -n 144 -N 1 -p gg -t 1:00:00
```

Example how to allocate 4 Grace-only nodes

```
salloc -n 576 -N 4 -p gg -t 1:00:00
```

Batch jobs

Example of script allocating 2 GH200 nodes (2 tasks/node) using `mpirun`

```
#!/bin/bash -l
#SBATCH --ntasks=4
#SBATCH --ntasks-per-node=2
#SBATCH --cpus-per-task=36
#SBATCH --nodes=2
#SBATCH --partition=gh
#SBATCH --time=1:00:00
#SBATCH --exclusive

. /global/exafs/groups/gh/bootstrap-env.sh
module purge
module load openmpi/5.0.6-gcc-13.3.0

mpirun -np 4 --map-by ppr:2:node:PE=36 \
   --report-bindings uname -a
```

Example of script allocating 2 GH200 nodes (2 tasks/node) using `srun`

```
#!/bin/bash -l
#SBATCH --ntasks=4
#SBATCH --ntasks-per-node=2
#SBATCH --cpus-per-task=36
#SBATCH --nodes=2
#SBATCH --partition=gh
#SBATCH --time=1:00:00
#SBATCH --exclusive

. /global/exafs/groups/gh/bootstrap-env.sh

srun --mpi=pmi2 uname -a
```

Example script allocating on 2 Grace-only nodes (full MPI-only) using `mpirun`

```
#!/bin/bash -l
#SBATCH --ntasks=288
#SBATCH --ntasks-per-node=144
#SBATCH --cpus-per-task=1
#SBATCH --nodes=2
#SBATCH --partition=gg
#SBATCH --time=1:00:00
#SBATCH --exclusive

. /global/exafs/groups/gh/bootstrap-env.sh
module purge
module load openmpi/5.0.6-gcc-13.3.0

mpirun -np 288 --map-by ppr:144:node:PE=1 \
   --report-bindings uname -a
```

Example of script allocating 4 Grace-only (MPI+OpenMP) using `mpirun`

```
#!/bin/bash -l
#SBATCH --ntasks=64
#SBATCH --ntasks-per-node=16
#SBATCH --cpus-per-task=9
#SBATCH --nodes=4
#SBATCH --partition=gg
#SBATCH --time=1:00:00
#SBATCH --exclusive

. /global/exafs/groups/gh/bootstrap-env.sh
module purge
module load openmpi/5.0.6-gcc-13.3.0

export OMP_NUM_THREADS=9
mpirun -np 64 --map-by ppr:16:node:PE=9 \
   --report-bindings uname -a
```

Example of script allocating 4 Grace-only (MPI-only, under-populated) using `mpirun`

```
#!/bin/bash -l
#SBATCH --ntasks=512
#SBATCH --ntasks-per-node=128
#SBATCH --cpus-per-task=1
#SBATCH --nodes=4
#SBATCH --partition=gg
#SBATCH --time=1:00:00
#SBATCH --exclusive

. /global/exafs/groups/gh/bootstrap-env.sh
module purge
module load openmpi/5.0.6-gcc-13.3.0

export OMP_NUM_THREADS=1
mpirun -np 512 --map-by ppr:64:socket:PE=1 \
   --report-bindings uname -a
```

Working with Singularity Containers

Singularity is the only container engine present at the moment. Docker or enroot workflows need to be adapted to run (as user) on Thea.

Run interactively pre-staged Singularity containers

(1) Allocate an interactive node

```
salloc -n 1 -N 1 -p gh -t 1:00:00
```

(2) Select container and invoke singularity `run`

```
export CONT="/global/exafs/groups/gh/sif/pytorch-25.01-py3.sif"
singularity run --nv "${CONT}"
```

TIP - `/local` is not persistent, remember to copy back into `${SCRATCH_DDN}` what is important to keep before job ends. Impossible to recover any data lost from `/local`.

Run interactively pre-staged Singularity containers

```
export CONT="/global/exafs/groups/gh/sif/pytorch-25.01-py3.sif"
srun --mpi=pmi2 -N 1 -n 1 --ntasks-per-node=1 -p gh -t 4:00:00 \
    singularity -v run --nv "${CONT}" python my_benchmark_script.py
```

NOTE - The current path where srun and singularity are executed is automatically exposed inside the container.

How to squash and run a NGC container into a new read-only Singularity image

TIP - Building a container is a very intense I/O operation, it is better to leverage `/local` when possible but remember to copy your sif image or sandbox folder back to `${SCRATCH_DDN}` before the job is completed otherwise all files are lost.

1. Allocate an interactive node

```
salloc -n 1 -N 1 -p gh -t 1:00:00
```

2. Set additional env variables

Make sure singularity pull operates entirely from `/local` for performance reasons and capacity constrains

```
export APPTAINER_TMPDIR=/local/${SLURM_JOBID}/_tmp_singularity
export APPTAINER_CACHEDIR=/local/${SLURM_JOBID}/_cache_singularity
rm -rf ${APPTAINER_TMPDIR} && mkdir -p ${APPTAINER_TMPDIR}
rm -rf ${APPTAINER_CACHEDIR} && mkdir -p ${APPTAINER_CACHEDIR}
```

3. Pull locally singularity image

```
singularity pull pytorch-25.01-py3.sif docker://nvcr.io/nvidia/pytorch:25.01-p
```

How to create a Singularity Sandbox and run / repackage a new container image

1. Grab one node in interactive mode

```
salloc -n 1 -N 1 -p gh -t 2:00:00
```

2. Identify which container to extend via a sandbox and prep the environment

```
export CONT_DIR=/global/exafs/groups/gh/sif
export CONT_NAME="pytorch-25.01-py3.sif"
export APPTAINER_TMPDIR=/local/${SLURM_JOBID}/_tmp_singularity
export APPTAINER_CACHEDIR=/local/${SLURM_JOBID}/_cache_singularity
rm -rf ${APPTAINER_TMPDIR} && mkdir -p ${APPTAINER_TMPDIR}
rm -rf ${APPTAINER_CACHEDIR} && mkdir -p ${APPTAINER_CACHEDIR}
```

3. Make a copy of base container as reading and verifying it is faster on local disk

```
cp ${CONT_DIR}/${CONT_NAME} /local/${SLURM_JOBID}/
```

4. Create a Singularity definition file

Start with the original NGC container as base image and add extra packages in the `%post` phase

```
cat > custom-pytorch.def << EOF
Bootstrap: localimage
From: /local/${SLURM_JOBID}/${CONT_NAME}

%post
    apt-get update
    apt-get -y install python3-venv
    pip install --upgrade pip
    pip install transformers accelerate huggingface_hub
EOF
```

After this there are two options:

5A. Create the sandbox on the persistent storage

TIP - Use this method if you want to customise your image by bulding manually software or debugging a failing `pip` command.

```
cd ${SCRATCH_DDN}
singularity build --sandbox custom-python-sandbox custom-pytorch.def
```

When completed, run on an interactive node via

```
singularity shell --nv custom-python-sandbox
```

5B. Create a new SIF image

TIP - Use this method if you want to create a read-only image to run workloads and you are confident all `%post` steps can run successfully without manual intervention.

```
cd ${SCRATCH_DDN}
singularity build custom-pytorch.def ./custom-pytorch.sif
```

When completed, run on an interactive node via

```
singularity shell --nv custom-pytorch.sif
```