Tools in action – an example with Pytorch

Samuel Antao

LUMI Training Course
May 30 – June 2

AMD
together we advance_

Agenda

1. Intro to Pytorch and its dependencies
2. Controlling affinity
3. Profiling – rocprof and omnitools.
4. Debugging

Pytorch highlight

- Official page: https://pytorch.org/
- Code: https://github.com/pytorch/pytorch
- Python™-based framework for machine learning
  - Auto-differentiation on tensor types
- GPU-enabled
  - ROCm support for MI250x (and others)
  - Hipification as part of the build system
    - C/C++ libraries with proper bindings for Python
  - Python code doesn't need changing – using the same CUDA conventions
- Other related packages:
  - Torch vision/audio, many others
  - APEX – multiprecision library
    - https://github.com/ROCmSoftwarePlatform/apex

Pytorch install – base environment

```
module purge
module load CrayEnv
module load PrgEnv-cray/8.3.3
module load craype-accel-amd-gfx90a
module load cray-python

# Default ROCm – more recent versions known to work (e.g. ROCm 5.5.0)
module load rocm/5.2.3.lua
```

Pytorch install – system python

- Natively
  - cray-python module
  - pip3 install -t $(pwd)/pip-installs2 --pre torch==1.13.1 --extra-index-url https://download.pytorch.org/whl/rocm5.2/
  - PYTHONPATH=$(pwd)/pip-installs2 python -c 'import torch; print(torch.cuda.device_count())'

Pytorch install – virtual env

- virtual env
  - cray-python module
  - python -m venv --system-site-packages cray-python-virtualenv
  - source cray-python-virtualenv/bin/activate
  - pip3 install --pre torch==1.13.1 --extra-index-url https://download.pytorch.org/whl/rocm5.2/
  - python -c 'import torch; print(torch.cuda.device_count())'

Pytorch install – conda env

- conda env
  - https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-x86_64.sh
  - bash Miniconda3-latest-Linux-x86_64.sh
  - source miniconda3/bin/activate
  - conda create -n pytorch python=3.8
  - conda activate pytorch
  - conda install --only-deps pytorch
  - pip3 install --pre torch==1.13.1 --extra-index-url https://download.pytorch.org/whl/rocm5.2/
  - python -c 'import torch; print(torch.cuda.device_count())'

Pytorch install – conda env – from source

- conda env
  - https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-x86_64.sh
  - bash Miniconda3-latest-Linux-x86_64.sh
  - source miniconda3/bin/activate
  - conda create -n pytorch-build python=3.8
  - conda activate pytorch-build
  - conda install --only-deps pytorch
  - conda install -y cmake mkl-include
  - Load source and build:

```
git clone -b v1.13.1 --recursive https://github.com/pytorch/pytorch
cd pytorch
git submodule sync
git submodule update --init --recursive --jobs 0


nice python3 tools/amd_build/build_amd.py
CMAKE_PREFIX_PATH=$CONDA_PREFIX:$CMAKE_PREFIX_PATH \
  PYTORCH_ROCM_ARCH=gfx90a \
  CMAKE_MODULE_PATH=$CMAKE_MODULE_PATH:$(pwd)/pytorch/cmake/Modules_CUDA_fix \
  LIBRARY_PATH=$CONDA_PREFIX/lib:$LIBRARY_PATH LDFLAGS="-ltinfo" \
  PYTORCH_ROCM_ARCH="gfx90a" \
  RCCL_PATH=$ROCM_PATH/rccl \
  RCCL_DIR=$ROCM_PATH/rccl/lib/cmake \
  hip_DIR=${ROCM_PATH}/hip/cmake/ \
  REL_WITH_DEB_INFO=1 \
  nice python3 setup.py bdist_wheel
```

  - conda create -n pytorch-from-source --clone pytorch-build
  - conda activate pytorch-from-source
  - pip install dist/torch-*.whl LD_LIBRARY_PATH=$CONDA_PREFIX/lib:$LD_LIBRARY_PATH python -c 'import torch; print(torch.cuda.device_count())'

AMD together we advance_

Controlling device visibility

- Controlling visibility
  - HIP_VISIBLE_DEVICES=0,1,2,3 python -c 'import torch; print(torch.cuda.device_count())'
  - ROCR_VISIBLE_DEVICES=0,1,2,3 python -c 'import torch; print(torch.cuda.device_count())'
  - SLURM sets ROCR_VISIBLE_DEVICES
  - Implications of both ways of setting visibility – blit kernels and/or DMA

  - Considerations:
    - Does my app expects GPU visibility to be set in the environment?
    - Does my app expects arguments to define target GPUs
    - Does my app make any assumption on the device based on other information:
      - MPI rank
      - CPU-range
      - Auto-determined
    - How many processes using the same GPU:
      - Contention vs occupancy
      - Runtime scheduling limits
      - Increased scheduling complexity
      - Imbalance

Testing affinity

- What CPUs I have available and their NUMA domain?
  - lscpu
- What GPUs I have
  - rocm-smi –showtopo

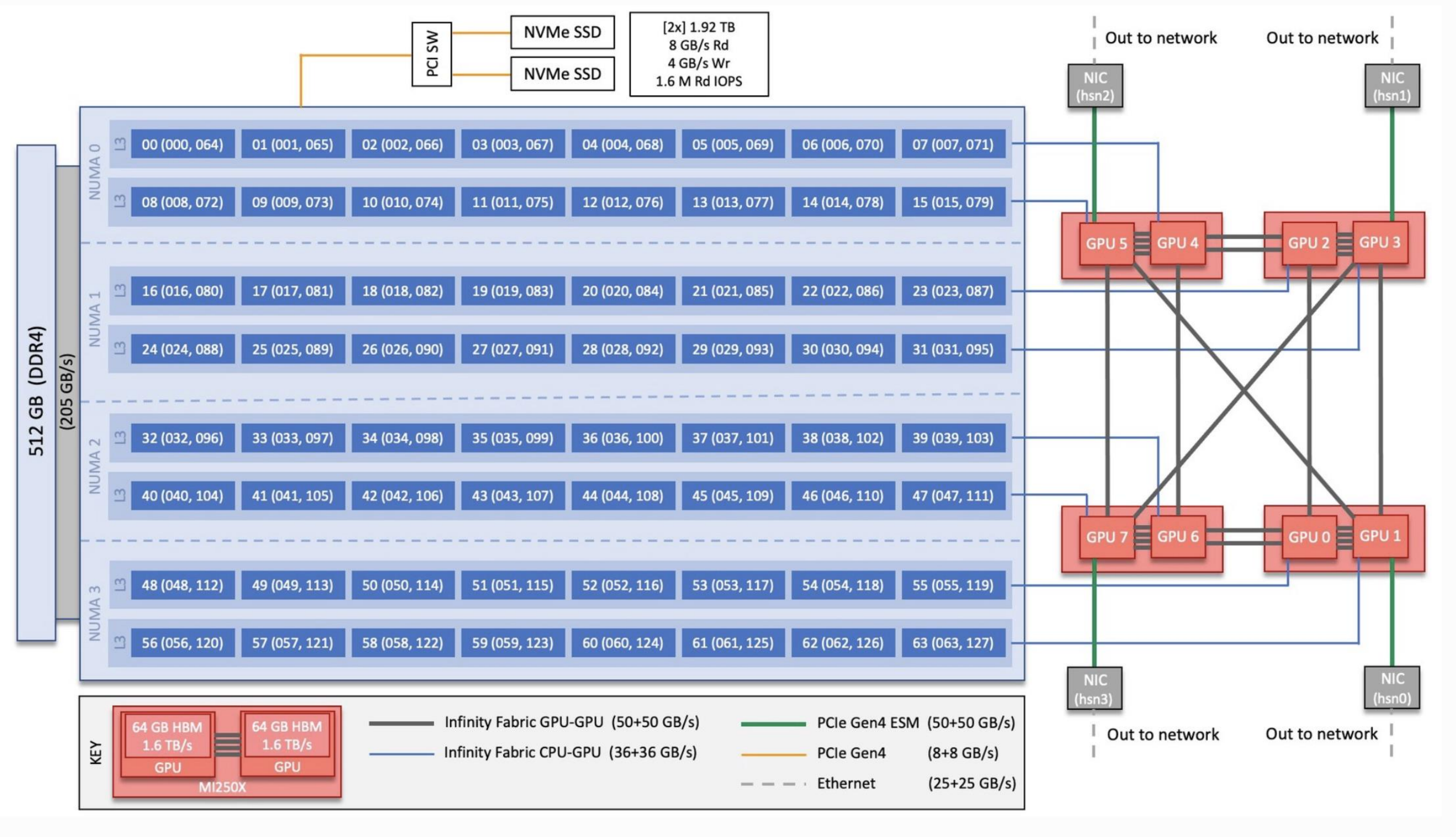
NUMA node0 CPU(s):	0-15,64-79
NUMA node1 CPU(s):	16-31,80-95
NUMA node2 CPU(s):	32-47,96-111
NUMA node3 CPU(s):	48-63,112-127

GPU[0]	: (Topology) Numa Node: 3
GPU[0]	: (Topology) Numa Affinity: 3
GPU[1]	: (Topology) Numa Node: 3
GPU[1]	: (Topology) Numa Affinity: 3
GPU[2]	: (Topology) Numa Node: 1
GPU[2]	: (Topology) Numa Affinity: 1
GPU[3]	: (Topology) Numa Node: 1
GPU[3]	: (Topology) Numa Affinity: 1
GPU[4]	: (Topology) Numa Node: 0
GPU[4]	: (Topology) Numa Affinity: 0
GPU[5]	: (Topology) Numa Node: 0
GPU[5]	: (Topology) Numa Affinity: 0
GPU[6]	: (Topology) Numa Node: 2
GPU[6]	: (Topology) Numa Affinity: 2
GPU[7]	: (Topology) Numa Node: 2
GPU[7]	: (Topology) Numa Affinity: 2

[Public]

Testing affinity

- ORNL topology - https://docs.olcf.ornl.gov/systems/crusher_quick_start_guide.html



May 30 - June 2

AMD
together we advance_

Testing affinity

- Check what SLURM is giving us:
  - `N=2 ;  salloc -p small-g --threads-per-core 1 --exclusive -N $N --gpus $((N*8)) -t 2:00:00  --account project_465000524 --mem 0`
  - `srun -c 7 -N 2 -n 16 --gpus 16 bash -c 'echo "$SLURM_PROCID -- GPUS $ROCR_VISIBLE_DEVICES -- $(taskset -p $$)"'`

```
 0 -- GPUs 0,1,2,3,4,5,6,7 -- pid 29793's current affinity mask: fe
 1 -- GPUs 0,1,2,3,4,5,6,7 -- pid 29794's current affinity mask: 7f00
 2 -- GPUs 0,1,2,3,4,5,6,7 -- pid 29795's current affinity mask: 3f8000
 3 -- GPUs 0,1,2,3,4,5,6,7 -- pid 29796's current affinity mask: 1fc00000
 4 -- GPUs 0,1,2,3,4,5,6,7 -- pid 29797's current affinity mask: fe0000000
 5 -- GPUs 0,1,2,3,4,5,6,7 -- pid 29798's current affinity mask: 7f000000000
 6 -- GPUs 0,1,2,3,4,5,6,7 -- pid 29799's current affinity mask: 3f80000000000
 7 -- GPUs 0,1,2,3,4,5,6,7 -- pid 29800's current affinity mask: 1fc000000000000
 8 -- GPUs 0,1,2,3,4,5,6,7 -- pid 35973's current affinity mask: fe
 9 -- GPUs 0,1,2,3,4,5,6,7 -- pid 35974's current affinity mask: 7f00
10 -- GPUs 0,1,2,3,4,5,6,7 -- pid 35975's current affinity mask: 3f8000
11 -- GPUs 0,1,2,3,4,5,6,7 -- pid 35976's current affinity mask: 1fc00000
12 -- GPUs 0,1,2,3,4,5,6,7 -- pid 35977's current affinity mask: fe0000000
13 -- GPUs 0,1,2,3,4,5,6,7 -- pid 35978's current affinity mask: 7f000000000
14 -- GPUs 0,1,2,3,4,5,6,7 -- pid 35979's current affinity mask: 3f80000000000
15 -- GPUs 0,1,2,3,4,5,6,7 -- pid 35980's current affinity mask: 1fc000000000000
```

Not good! Allocations crossing NUMA domains!!

RCCL attempt using only high-speed-interfaces

- The problem – on startup we see:
  NCCL error in: /pfs/lustrep2/projappl/project_462000125/samantao/pytorch-example/pytorch/torch/csrc/distributed/c10d/ProcessGroupNCCL.cpp:1269, unhandled system error, NCCL version 2.12.12

- Checking error origin:
  - export NCCL_DEBUG=INFO
  - NCCL INFO NET/Socket : Using **[0]nmn0:10.120.116.65<0>** [1]hsn0:10.253.6.67<0> [2]hsn1:10.253.6.68<0> [3]hsn2:10.253.2.12<0> [4]hsn3:10.253.2.11<0>
  - NCCL INFO /long_pathname_so_that_rpms_can_package_the_debug_info/data/driver/rccl/src/init.cc:1292

- The fix:
  - export NCCL_SOCKET_IFNAME=hsn0,hsn1,hsn2,hsn3

Pytorch example app – transformer – training MLPerf

- Translation benchmark
  - Bind script
    - N=1 ; salloc -p small-g --threads-per-core 1 --exclusive -N $N --gpus $((N*8)) -t 0:10:00 --account project_465000524 --mem 0
    - run_training_lumi.sh
    - bind_launch.py
  - Explicit numactl within SLURM environment
    - Leveraging numactl library within launching script
    - Set bind to none to have all resources available for each spawn process
  - SLURM launch
    - Use binding masks to control bind
    - Recreate Pytorch distributed environment from SLURM environment
    - Watch out for SLURM inconsistencies.

Pytorch example app - profiling

- Rocprof
  - Hip trace

- Reduce profiling scope and load large profiles
  - Break epoch loops
  - Break iteration loops
  - trace_processor for large profiles

RCCL AWS-CXI plugin

- RCCL relies on runtime plugin-ins to connect with some transport layers
  - Libfabric – provider for Slingshot

- Hipified plugin adapted from AWS OpenFabrics support available
- https://github.com/ROCmSoftwarePlatform/aws-ofi-rccl
- 3-4x faster collectives

- Plugin needs to be pointed at by the loading environment

```
module use /pfs/lustrep2/projappl/project_462000125/samantao-public/mymodules
module load aws-ofi-rccl/rocm-5.2.3.lua
Or
export LD_LIBRARY_PATH=/pfs/lustrep2/projappl/project_462000125/samantao-public/apps-rocm-5.2.3/aws-ofi-rccl
(will detect librccl-net.so)
```

- Verify the plugin is detected.

```
 export NCCL_DEBUG=INFO
 export NCCL_DEBUG_SUBSYS=INIT
# and search the logs for:
[0] NCCL INFO NET/OFI Using aws-ofi-rccl 1.4.0
```


AMD
together we advance_

Omnitrace

- Obtain more thorough trace information and visualization
  - https://github.com/AMDResearch/omnitrace
  - OpenSUSE 15.3 build mainly compatible with LUMI environment

- Module files to help assist, e.g.
  - module use /pfs/lustref1/flash/project_462000125/samantao/lumi-training/omnitools/omnitrace-1.10.0-opensuse-15.3-ROCm-50200-PAPI-OMPT-Python3/share/modulefiles
  - module load omnitrace

- Configuration file:
  - omnitrace-avail -G omnitrace.cfg –all
  - Use OMNITRACE_CONFIG_FILE environment variable to point to it
  - Override environment with command line arguments

- Sampling the Python™ and C/C++ parts of the code
  - omnitrace-python-3.8 -c <configuration path>/omnitrace.cfg – script.py"
  - omnitrace-sample --trace –c <configuration path>/omnitrace.cfg -- python –u ./scripts.py
  - Match omnitrace-python with your Python version.

Omniperf

- Obtain detail kernel performance counters
  - https://github.com/AMDResearch/omniperf
  - OpenSUSE 15.3 build mainly compatible with LUMI environment
- Module files to help assist, e.g.
  - module use /flash/project_462000125/samantao/lumi-training/omnitools/omniperf-modulefiles
  - module load omniperf
- Configuration and build:
  - Omniperf requirements must be installed in a Python version and environment compatible with the one used by the target app.
  - E.g. make sure omniperf requirements exist within same conda environment.
  - Sampling the Python and C/C++ parts of the code
- Omniperf needs replaying the application
  - Complicated to profile individual ranks as all need replaying.
- Profile with:
  - omniperf profile -n pytorch --device 0 --roof-only -- $(which python) -u
- Analyze with:
  - omniperf analyze -p workloads/pytorch/mi200/ --gui

Rocgdb

- https://github.com/ROCm-Developer-Tools/ROCgdb/
  - Branches for given ROCm releases: e.g. rocm-5.2.x
- Two main use cases
  - Connecting into a hanging process
  - Progress up to breakpoint or segfault
- ROCm provides rocgdb – you may need your own gdbserver.

- Using gdbserver
  - gdbserver can be issued conveniently as a profile tool
  - Launch with:
    - gdbserver --once $(hostname):12345 ./my_command
  - Attach with
    - rocgdb -x gdb.commands ./my_command
  - Leverage gdb commands file to automate startup
    - target remote target_host:12345

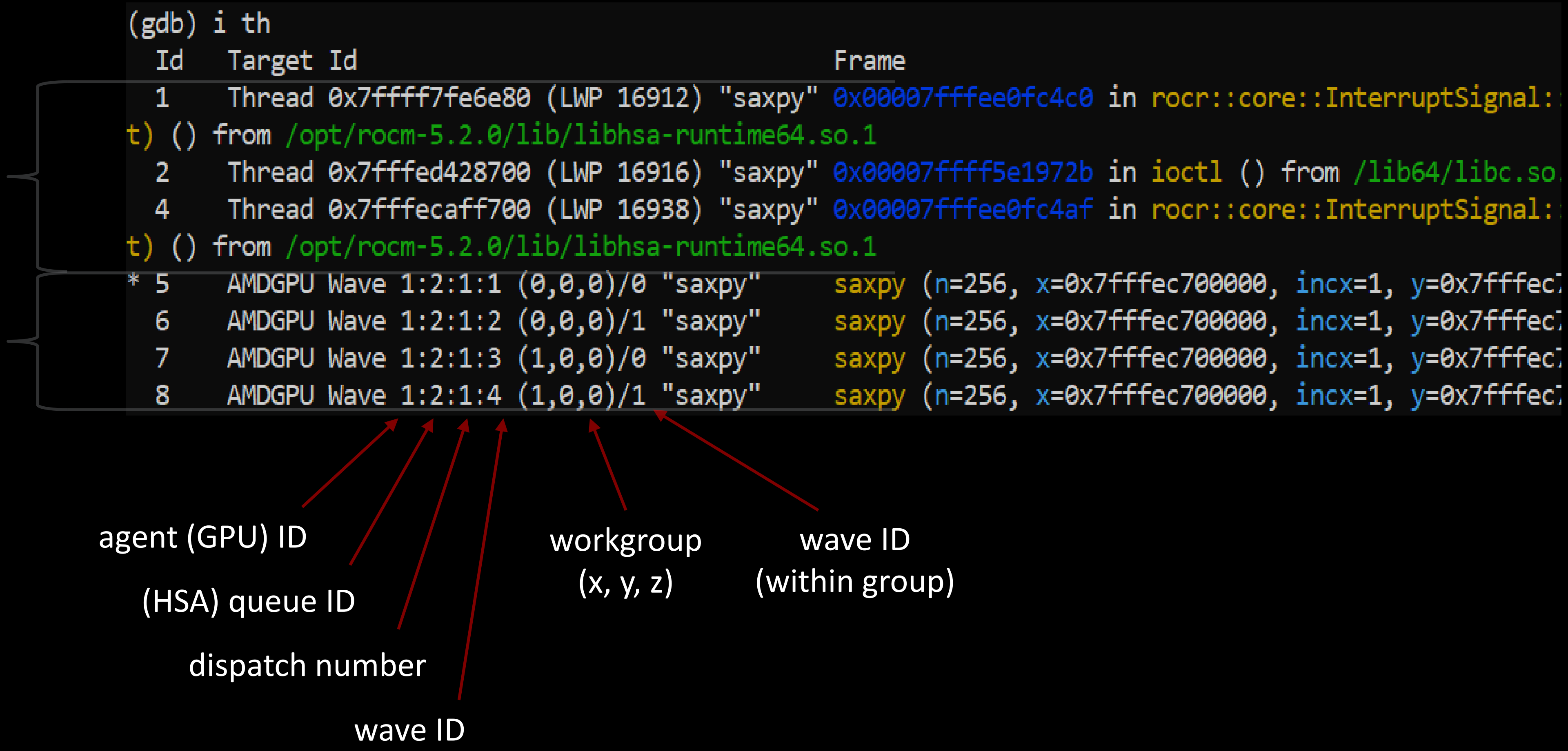
- Two examples:
  - Hanging in collective (RCCL)
  - Connect to remote gdb session – breakpoints

- Limitations
  - GPU pending breakpoints over gdbserver may not work.

- Staring session in specific to nodes
  - srun --interactive --pty /bin/bash   (only works for first node of allocation)
  - srun --pty --jobid <jobid> -w <target_node> --mem=0 --oversubscribe –interactive -n 1 -c 63 --gpus-per-task=0 /usr/bin/bash (GPU's won't be visible)

Wave details

`agent-id:queue-id:dispatch-num:wave-id (work-group-x,work-group-y,work-group-z)/work-group-thread-index`



AMD
together we advance_

Disclaimer

The information presented in this document is for informational purposes only and may contain technical inaccuracies, omissions, and typographical errors. The information contained herein is subject to change and may be rendered inaccurate for many reasons, including but not limited to product and roadmap changes, component and motherboard version changes, new model and/or product releases, product differences between differing manufacturers, software changes, BIOS flashes, firmware upgrades, or the like. Any computer system has risks of security vulnerabilities that cannot be completely prevented or mitigated. AMD assumes no obligation to update or otherwise correct or revise this information. However, AMD reserves the right to revise this information and to make changes from time to time to the content hereof without obligation of AMD to notify any person of such revisions or changes.

THIS INFORMATION IS PROVIDED ‘AS IS.” AMD MAKES NO REPRESENTATIONS OR WARRANTIES WITH RESPECT TO THE CONTENTS HEREOF AND ASSUMES NO RESPONSIBILITY FOR ANY INACCURACIES, ERRORS, OR OMISSIONS THAT MAY APPEAR IN THIS INFORMATION. AMD SPECIFICALLY DISCLAIMS ANY IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, OR FITNESS FOR ANY PARTICULAR PURPOSE. IN NO EVENT WILL AMD BE LIABLE TO ANY PERSON FOR ANY RELIANCE, DIRECT, INDIRECT, SPECIAL, OR OTHER CONSEQUENTIAL DAMAGES ARISING FROM THE USE OF ANY INFORMATION CONTAINED HEREIN, EVEN IF AMD IS EXPRESSLY ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

Third-party content is licensed to you directly by the third party that owns the content and is not licensed to you by AMD. ALL LINKED THIRD-PARTY CONTENT IS PROVIDED “AS IS” WITHOUT A WARRANTY OF ANY KIND. USE OF SUCH THIRD-PARTY CONTENT IS DONE AT YOUR SOLE DISCRETION AND UNDER NO CIRCUMSTANCES WILL AMD BE LIABLE TO YOU FOR ANY THIRD-PARTY CONTENT. YOU ASSUME ALL RISK AND ARE SOLELY RESPONSIBLE FOR ANY DAMAGES THAT MAY ARISE FROM YOUR USE OF THIRD-PARTY CONTENT.

Questions?

AMD together we advance_

AMD