Tuolumne Early Users Water Cooler 12-10-2024

Scott Futral (futral2@llnl.gov)

LLNL-PRES-2001638

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract DE-AC52-07NA27344. Lawrence Livermore National Security, LLC

Lawrence Livermore National Laboratory

Tuolumne

Lawrence Livermore
National Laboratory
TUOLUMNE
Mike Carter
Photography

Agenda

- Welcome and Introduction to the Tuolumne Early Users Water Cooler – Scott F
- El Cap Center of Excellence role – Ramesh Pankajakshan
- User Support for Tuo Early Access and going forward – Elsa Gonsiorowski
- Rabbit storage ‘update’ – Elsa Gonsiorowski
- Essential Information for Success Using Tuolumne – John Gyllenhaal
- User questions and feedback on current state of Tuolumne – The Users

System Facts

- https://hpc.llnl.gov/hardware/compute-platforms/tuolumne
- 1,098 batch nodes
- 46 debug nodes
- 8 login nodes (alias ‘tuo’ works if you are like me and can’t spell.)
- AMD APU Architecture AMD MI300A
- Lustre file system /p/lustre5

Lawrence Livermore National Laboratory

El Capitan Center of Excellence

Judy Hill (judy@llnl.gov)

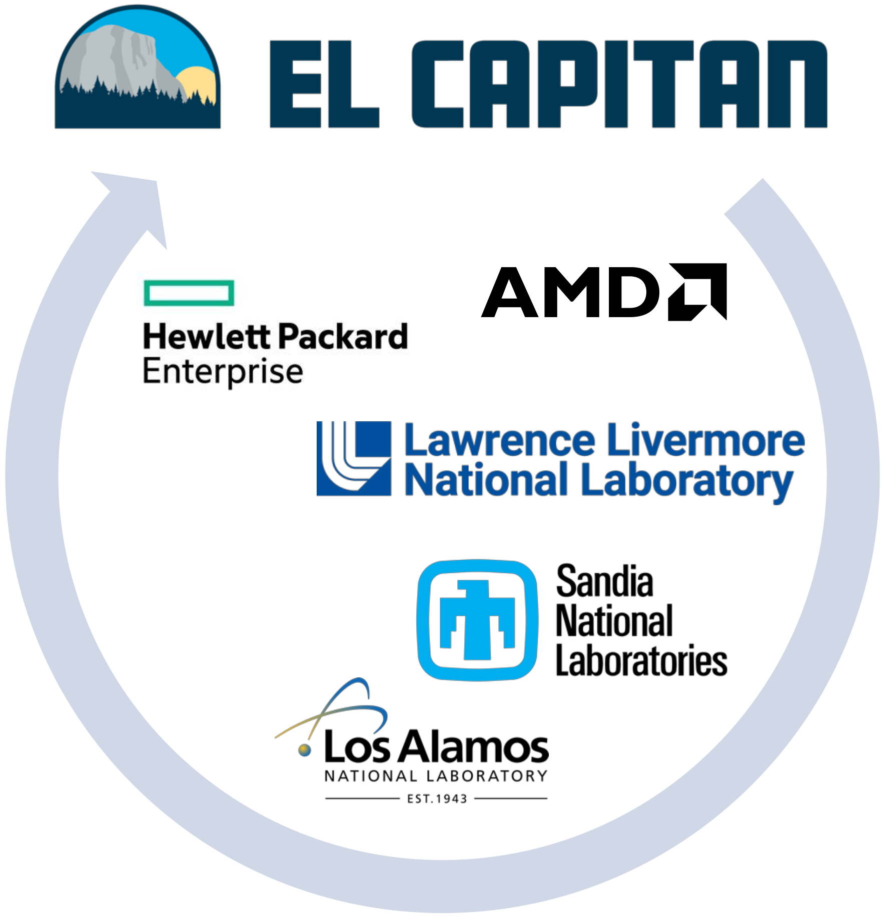
Ramesh Pankajakshan (ramesh@llnl.gov)

LLNL-PRES-2001638

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract DE-AC52-07NA27344. Lawrence Livermore National Security, LLC

A Center of Excellence is a close partnership between the DOE Laboratories and the vendor partner(s)

- Established joint work plans, information sharing, and collaboration mechanisms
- Dedicated vendor staff worked alongside lab code teams
  - Some staff assigned to work at lab sites
- Labs provided access to our codes
  - Including classified codes for those with security clearance
- Vendors provided NDA information and early access to hardware and software


EL CAPITAN
Hewlett Packard Enterprise
AMD
Lawrence Livermore National Laboratory
Sandia National Laboratories
Los Alamos National Laboratory EST. 1943

Forming a Center of Excellence has become a recognized best practice for large DOE system procurements

El Capitan COE Resources Available to Tuolumne Users

- For Tri-Lab Users:
  - El Capitan Confluence Pages
  - El Capitan COE E-mail Distribution Lists
  - El Capitan Mattermost

For access, E-mail:

Judy Hill (judy@llnl.gov)

Ramesh Pankajakshan (ramesh@llnl.gov)

- For All Users:
  - (Non-NDA) El Capitan Webinars
    - Next: Dec 17 at 2pm: OpenMP Webinar presented by HPE
  - El Capitan Hackathons, space-permitting
    - Next: Jan 28 – 30 at LLNL

Contact us if you need access instructions to any of these communication mechanisms

Reminder: COE Information is still subject to NDA restrictions

- Do: Discuss this material with other Tri-Lab staff members with “need to know”

- Don’t: Share this information with academics, collaborators or others outside LANL, LLNL, and SNL
  - Exception: CORAL-2 working groups, Others who have NDA/Contractual relations with CORAL-2. **ASK FIRST!!**

- **Results from machines (including Tuolumne) must be approved by HPE/AMD for publication or presentations. This includes results in public github repos.**

- Reminders here: https://lc.llnl.gov/confluence/x/qg2CJw

Don’t fix bugs later; fix them now.

– Steve Maguire

Report *any* and *all* issues to the LC Hotline!

(925) 422-4531

- The Hotline will work with the El Capitan Center of Excellence to report compiler bugs and other issues to HPE and AMD

Lawrence Livermore National Laboratory

User Support, Documentation Rabbit Status

Tuolumne Early Users Water Cooler

December 10, 2024

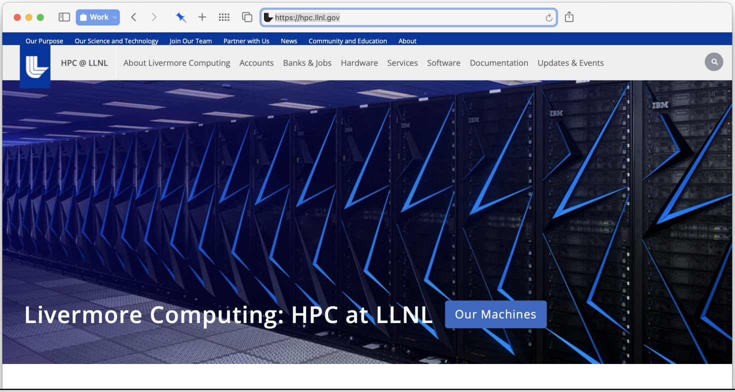
Elsa Gonsiorowski

LLNL-PRES-2001638

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract DE-AC52-07NA27344. Lawrence Livermore National Security, LLC

LC Documentation

https://hpc.llnl.gov



Using El Capitan Systems

Navigate to Documentation > Users Guides > Using El Capitan Systems

- QuickStart
  - C++ Code Examples
  - Fortran Code Examples
  - Explicit Paths Build Example
- Announcements and Presentations
- Known Issues
- Hardware Overview
- MPI Overview
- GPU Programming
- Compilers and User Environment
  - LC Magic Modules Guide
  - Cray Modules Guide
  - Spack Guide
- Running Jobs with Flux and mpibind
- Debugging Tools
- Performance Tools
- Math Libraries
- File Systems and Rabbits

LC Hotline Support

Contact us 1-925-422-4531

Technical Consultants

- option 1
- lc-hotline@llnl.gov

Account Specialists

- option 2
- lc-support@llnl.gov

Visit us B453 R1103 (Q-clearance area)

Hours

Monday–Friday
8am–12pm, 1–4:45pm
Except LLNL holidays

El Capitan Systems Vendor Support

- Rob Noska, HPE
- Austin Ellis, AMD

LC Hotline Support

Contact us 1-925-422-4531

Technical Consultants

- option 1
- lc-hotline@llnl.gov

Account Specialists

- option 2
- lc-support@llnl.gov

Visit us B453 R1103 (Q-clearance area)

Hours

Monday–Friday
8am–12pm, 1–4:45pm
Except LLNL holidays

Rabbit Rack-local Storage

- 1 Rabbit node:
  - 18 SSDs
  - 1 AMD Epyc CPU
  - PCIe connection to local compute nodes
  - Connected to rack-level switch

Tuolumne Static Setup

- Available to all jobs
- `/l/ssd` on compute nodes
- 763GB, node-local access

- *Test users wanted*
- Can be used similar to network-attached storage, for cross-node access
- Will be able to support Lustre, XFS, GFS2
- Configurable storage via Flux orchestration
- Will be able to perform data staging via Cray DW directives

Lawrence Livermore National Laboratory

Essential Information for Success Using Tuolumne (Flux, Compilers, Settings, and More!)

John Gyllenhaal, Livermore Computing (LC)

Tuolumne Watercooler
Tuesday December 10th, 2024

LLNL-PRES-2001638
This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract DE-AC52-07NA27344. Lawrence Livermore National Security, LLC

Lawrence Livermore National Laboratory

Goals and Scope

- Present concise and actionable rules of thumb for using El Cap systems like Tuolumne
  - Present key concepts and commands for compiling applications and running on El Cap systems
  - Make you aware of the existence of several complex technical details that may impact your application
    - If your application is impacted, will indicate how to reach out to get focused help

- Aiming for biggest impact with a “short” presentation
  - Presenting the key points of dozens of hours of presentations over the last few years
    - There will be longer, deeper dives in the future
  - Will not dive into all the juicy technical details or be able provide full answers to questions
    - The “real” answers to “why” are often quite complex, confusing, time consuming, and requiring NDAs

- No NDA required for this talk’s info
  - Often limits discussion of the technical details that drive the rules of thumb presented

- Snapshot in time
  - Systems still evolving as new functionality comes online (some things changing this week!)

Where to get help and key documentation for today's talk (Take a screen shot now!)

- Point of contact for getting help or asking questions
  - LC Hotline: lc-hotline@llnl.gov or 925-422-4531
- Tuolumne's web quickstart guide
  - https://hpc.llnl.gov/documentation/user-guides/using-el-capitan-systems/quickstart
- Run and scheduling jobs with Flux
  - https://hpc-tutorials.llnl.gov/flux/
  - https://hpc.llnl.gov/banks-jobs/running-jobs/batch-system-cross-reference-guides
- Accessing LC systems smoothly with ssh
  - https://dev.llnl.gov/security-access/ssh/
- Accellerating PyTorch, TensorFlow, or any ML code using RCCL
  - https://lc.llnl.gov/confluence/display/LC/2023/06/02/RCCL+performance+on+Tioga#RCCLperformanceonTioga-InstalltheAWS-OFI-RCCLplugin
  - https://lc.llnl.gov/confluence/display/LC/Distributed+PyTorch+on+CORAL+2+systems#DistributedPyTorchonCORAL2systems-Extra:InstallAWS-OFI-RCCLplugin
  - Sample build script /collab/usr/global/tools/rccl/toss_4_x86_64_ib_cray/rocm-6.2.1/buildme

Accessing Tuolumne (Tuo) and ssh key hints and gotchas

- Can use ssh tuo.llnl.gov or ssh tuolumne.llnl.gov from outside LC without VPN
  - Or you can bounce through oslic.llnl.gov if that doesn't work for you
- You cannot use ssh keys to connect to LC supercomputers from outside LC
  - Must use a two-factor authentication to get into an LC supercomputer
    - Can proxy through initial connection (usually oslic for CZ)
  - https://dev.llnl.gov/security-access/ssh/ has ssh config files and info that may help
    - Ian Lee's .ssh/config files are FOR PLACEMENT ON YOUR WORKSTATION ONLY NOT LC SUPERCOMPUTERS
    - Placing in your LC .ssh/config file very common mistake that breaks CORAL1 and causes unnecessary authentications
- We require you to use 4k bits (or larger) rsa ssh keys (without passphrase) on LC
  - ssh-keygen -t rsa -b 4096 -N “”
    - LC has common home directories, so empty passphrase does not reduce security and is approved for use at LC
  - Will enable you to ssh between LC machines without password or passphrase
- Do not put LC private keys on different zones (don't share CZ and RZ private keys)
  - Do not copy your CZ LC .ssh/id_rsa file outside the CZ!
    - Do not put on RZ or on non-LC machine

High-level El Cap MI300A APU-based Node Overview

- Four sockets per node
  - Each socket contains one powerful GPU, 21 user cores (+ 3 system) (2 HW threads/core), and 128 GB HBM3 memory
    - Roughly 120GB memory per socket available to users (OS and system processes use the rest)
  - Each node has 4 GPUs, 84 user cores and 12 system cores and 512GB HBM3 memory
  - GPUs, CPUs, OS, ram disks (i.e., /tmp) all share same HBM3 "GPU" memory (no DRAM) and unified address space
  - Four NUMA domains per node
    - Best practice: 4 MPI ranks per node, each using 1 GPU with memory and cores from same socket via binding
- Binding is critical, everything must be on same socket for good performance
  - Microbenchmarks run up to 10X slower if GPU accessing memory from different socket
    - Although you can actually access all 480GB avail memory from one GPU, it will be slow!
    - Using hipMalloc for most or all memory allocations gives great binding and page size settings automatically
  - Running 3X+ slower than expected is usually due to bad binding, often due to missing flux run option --exclusive (-x)
    - More binding, page size, and running with flux guidance in later slides
- May be able to share a single GPU with up to 4 MPI tasks with caveats (and luck)
  - 2X hardware-based GPU sharing usually works by default, env variables can often allow efficient 4X GPU sharing
    - If you need to share GPUs (i.e., UQ of tiny runs), reach out for more details on what to try (not covered more in this overview)
  - Performance falls off cliff if GPU falls back to software-based context switching of GPU
    - May fall off performance cliff for other reasons. 10% or less overhead expected for working-well cases.

Maximizing Application Performance under the Flux Scheduler

- Pro tip: Always use **--exclusive** (or **-x**) with flux run and don't specify other constraints
  - Only specify --nodes=# (or -N #) and --ntasks=# (or -n #) or --tasks-per-node=#
  - Do **NOT** explicitly specify --cores-per-task=# (or -c #) or --gpus-per-task=# (or --g #) for HPC runs
    - Those -c and -g options are for packing nodes for UQ or regression testing, may yield poor performing bindings
  - **--exclusive** (or **-x**) tells flux to optimally divide node resources between tasks for performance using mpibind
    - In general, need 4 (or multiple of 4) tasks per node for optimal performance due to 4 sockets

- **84** (of 96) cores dedicated to user processes, 12 cores reserved for system and lustre
  - Those 84 cores to bind to currently specified by MPIBIND_RESTRICT (must be set to scale well):
    - **MPIBIND_RESTRICT**=1-7,9-15,17-23,25-31,33-39,41-47,49-55,57-63,65-71,73-79,81-87,89-95,97-103,105-111,113-119,121-127,129-135,137-143,145-151,153-159,161-167,169-175,177-183,185-191
  - To actually be able to use all 96 cores, unset MPIBIND_RESTRICT (but will have a lot of noise at scale)
    - To enable use all of cores, we have to tell flux about all 96 of them which is why MPIBIND_RESTRICT needed

- The 'srun' wrapper for flux automatically adds --exclusive for you
  - srun -N 4 --ntasks-per-node=4 …

Early access mode on Tuolumne, socially scheduled

- 1100 nodes in pbatch, 24 hour time limit
  - Currently no technical limits on job sizes and job quantity to maximize flexibility
    - Be a good neighbor, ask for advice, and do not monopolize Tuolumne, many users need to be able to run big jobs
- Be a good neighbor!
  - Do no use more than half of UP nodes without permission (550 nodes if all up, 512ish better target max size)
    - Goal: Enable 2 different users to be able to run big jobs at once
    - It is early days, unlikely all nodes will stay up and available (e.g., 1039 pbatch available nodes up on 11/9/24)
  - You may idle a large number of nodes if requested node count cannot be fulfilled until other big job finishes
    - Flux backfill scheduling can help if you pick a run time shorter than when that big job is expected to start
    - Please monitor your jobs to make sure they are not unexpectedly blocking everyone (common issue but backfill helps)
  - If need to run a large number of small GPU jobs, bundle them into larger allocations if possible (flux makes this easy)
    - Much better on scheduler and for enabling large jobs (flux was designed to do this for UQ and regression tests)
- 44 nodes in pdebug, 1 hour time limit
  - For building codes, code development, debugging, and testing
  - You can build and test in pbatch if need longer run time
  - Please do not do production work in pdebug (i.e., running chained jobs 24 hours a day)

Always allocate Interactive Compute Nodes for Compiling, Testing, and Debugging

- Always allocate a pdebug (or pbatch) compute node for compiling and testing your code
  - Please do not run big compiles or application runs on the login nodes!
    - Crashing or OOMing login nodes can kill all running jobs and app runs can get login node GPUs into bad states
  - We understand that this may be the different than other supercomputer sites policies

- The ‘pdebug’ pool is only for compiling/testing/debugging, not production work
  - Use as few nodes as feasible (no more than half total) and do not block queue with large requests
    - Use ‘pbatch’ allocations if need to debug or test large node count jobs or long running jobs

- Please do not game the system
  - We rely on social contracts and good neighbor polices not strong technical controls
    - Continuing to abuse login nodes, file systems, batch queues, etc., after being warned not to has gotten users banned
  - Hogging pdebug nodes with obvious production work most common form of abuse, do not abuse pdebug!
    - We know those idle nodes look tempting, but you are preventing others from getting their work done

- Need help or system seems hung, contact LC Hotline lc-hotline@llnl.gov or 925-422-4531
  - Please feel free to ask how you get can something done “the right way”

How to avoid “bad” nodes for your code

- What to do if some nodes appear problematic for your application?
  - Please report it to the LC Hotline and tell us the symptoms to see if it needs to be replaced
  - Tell flux to NOT schedule your allocations on those node(s) with --requires=-host:name1,name2,…
    - Note the **-** before the -host. This indicates do not use those hosts.

- Batch and interactive examples if tuolumne1005,tuolumne1007 in pdebug appear bad
  - flux batch --queue=pdebug --requires=-host:tuolumne1005,tuolumne1007
  - flux --parent alloc --nodes=2 --queue=pdebug --time-limit=1h --requires=-host:tuolumne1005,tuolumne1007

- You can also request specific nodes (=host, no - before host) but please use sparingly
  - Will delay launching of your allocation and can be hard to tell why delay in flux queries
  - These constraints puts higher load on flux scheduler, so not want queue flooded with these request
  - You must specify exactly all the nodes desired to run on with --requires=host:name1,name2,…
    - flux --parent alloc --nodes=2 --queue=pdebug --time-limit=1h --requires=host:tuolumne1006,tuolumne1008

- Currently no way to specify nodes to avoid or use with salloc/sbatch “slurm” wrappers

PyTorch, TensorFlow, or using RCCL, need to use AWS Plugin

- RCCL (ML-optimized communications) using TCP sockets is very slow on CORAL2's network
  - AWS wrote open-source RCCL Plugin (librccl-net.so) to use libfabric to greatly accelerates RCCL on slingshot
  - Currently you need to build it for your application if using RCCL/PyTorch/TensorFlow
    - May bundle with future LC rocm installs (exploring tradeoffs and testing options)
    - LD_LIBRARY_PATH needs to point to the directory that contains the plugin library

- Instructions for building and using AWS RCCL plugin here:
  - Example of build, testing, and verifying RCCL on Tioga (remember to use --exclusive or -x on Tuolumne)
    - **https://lc.llnl.gov/confluence/display/LC/2023/06/02/RCCL+performance+on+Tioga#RCCLperformanceonTioga-InstalltheAWS-OFI-RCCLplugin**
  - PyTorch specific instructions:
    - **https://lc.llnl.gov/confluence/display/LC/Distributed+PyTorch+on+CORAL+2+systems#DistributedPyTorchonCORAL2systems-Extra:InstallAWS-OFI-RCCLplugin**
  - A very useful sample build script: **/collab/usr/global/tools/rccl/toss_4_x86_64_ib_cray/rocm-6.2.1/buildme**

How to choose between CC, crayCC, and magic compiler wrappers

- cc/CC/ftn interface provides "CRAY" magic with everything specified by Cray module files
  - Automatically add -lxpmem, -lmpi_gtl_hsa, libsci, etc. based on modules loaded (9 modules loaded by default)
  - Useful for those that value the traditional cray interface
  - Best practice: Have same Cray and rocm modules loaded when compiling and running executable
- New craycc/crayCC/crayftn and mpicc/mpicxx/mpifort provides module-less Cray environment
  - Common interface but YOU must link in -lxpmem, -lmpi_gtl_hsa, libsci, etc. explicitly for best performance
  - Enables cmake, autotools, spack in cray environment but need to add extra link options for performance
  - Best practice: Add RPATHs to your all your libraries for consistent running of executable
- The -magic wrappers provide "LC" magic where environment is totally ignored
  - Load compiler with -magic extension (cce/18.0.1-magic or rocmcc/6.2.1-magic) to get LC magic
    - mpicc/mpicxx/mpifort automatically switch to magic versions with -magic compiler loaded
  - Auto adds RPATHs to compilers and rocm and adds -lxpmem -lmpi_gtl_hsa (but not libsci or Cray libraries)
  - Best practice: Use full path to compiler in build system (should have -magic in path)
    - CXX= /usr/tce/packages/cray-mpich/cray-mpich-8.1.31-rocmcc-6.2.1-magic/bin/mpicxx

Why don’t we load the rocm module by default?

- **Having a ROCM_PATH set to a different rocm than your application uses can cause errors!**
  - AMD designed ROCM_PATH to allow plugging in debug rocm .so files at runtime
    - So setting ROCM_PATH to a different rocm can cause your application to mix two different rocm library sets
  - Everyone else (cce, cmake, etc) uses ROCM_PATH to find which rocm you are using

- **The -magic compilers add link line options to make executable ignore ROCM_PATH**
  - -L/opt/rocm-6.2.1/lib -Wl,-rpath,/opt/rocm-6.2.1/lib -lamdhip64 -lhsakmt -lhsa-runtime64 -lamd_comgr
  - We recommend you add something like the above to your link line (for the correct rocm, of course)

Spindle on by default starting this afternoon (12/10/2024)

- Linux and python .so search algorithm at scale causes denial-of-service attack on file systems
  - Without mitigation, can take hours to start executable and makes LC filesystems unresponsive during that time
  - Rapid small-scale runs of many python launches can cause same file systems issues
    - 4 nodes running 10 executable invocations per core per second used 99.9% of file system resources until mitigated
  - Please do not initialized CONDA, python, or spack in your .bashrc file.   Can cause huge file system load!

- Main symptom is slow launch times (or sysadmins killing your job and contacting you)
  - With Spindle, full El Cap scale launches typically take less than 2 minutes
  - Without mitigation, full El Cap scale launches take > 2 hours and make all CZ filesystems very slow
  - If it is taking more than 2 minutes to launch your application, please reach out to us

- Spindle uses advanced algorithms at launch time to mitigate file system load
  - Multiple levels of optimization (default ‘medium’)
    - Spindle works around glibc bugs uncovered with fastload2 (previous automatic mitigation technique)
  - If suspect Spindle causing issues, spindle can be turned off with env variable SPINDLE_FLUXOPT=disable
    - Many other options like flux run -o spindle.level=off or per-user configuration in ~/.spindle/spindle.conf

Turning on GPU-aware MPI and using xpmem can significantly increase MPI performance

- Starting with cray-mpich/8.1.30 (July 2024), -magic compilers automatically add libraries
  - -lxpmem -L/opt/cray/pe/mpich/8.1.30/gtl/lib -lmpi_gtl_hsa -Wl,-rpath,/opt/cray/pe/mpich/8.1.30/gtl/lib
  - -lxpmem doubles bandwidth on node using cpu-level MPI
  - Can turn off with --no-xpmem and/or --no-gtl if suspect libraries causing issues or unexpected slowdowns
- If not using -magic compilers, recommend add to link line (here for cray-mpich/8.1.31)
  - -lxpmem -L/opt/cray/pe/mpich/8.1.31/gtl/lib -lmpi_gtl_hsa -Wl,-rpath,/opt/cray/pe/mpich/8.1.31/gtl/lib
- Turn on GPU-aware MPI by setting env variable MPICH_GPU_SUPPORT_ENABLED=1
  - Must have GTL library linked in for this to work (punts otherwise)
  - Yields another 2X+ on-node memory bandwidth increase over xpmem
- ABI used by GTL changed with rocm/6.0
  - rocm/5.7.1 requires use of older cray-mpich/8.1.27
  - ABI stable after rocm/6.0 (but beta testing found issues and got them fixed before GA versions).

hipMalloc, HSA_XNACK=1, and 2mb pages options

- **GPU performs best with 2mb pages and requires pages touched by GPU mapped into GPU**
  - — Use hipMalloc when possible to allocate memory, get 2mb pages mapped in GPU that can be used on CPU
    - Doing this will greatly simplify your life and maximize GPU performance

- **CPUs and GPUs share memory but cpu-based allocators not auto mapped to GPU**
  - — setting env variable HSA_XNACK=1 will page-fault in CPU pages into GPU (with slight overhead)
  - — CPU default 4k page size will cause ~15% performance overhead on GPU due to GPU TLB size

- **Enabling transparent huge pages makes most > 2mb CPU allocations have 2mb pages**
  - — Must be enabled at compute node allocation time
    - flux alloc --setattr=thp=always -N1
    - salloc -N1 --thp=always
  - — Recommended starting point if allocating memory on CPU to be used on GPU (need HSA_XNACK=1 also)

- **Linking in -lhugetlbfs and enabling in allocation can put < 2mb CPU allocations in 2mb pages**
  - — Must be enabled at compute node allocation time and can coexist with thp
    flux --parent alloc --setattr=hugepages=512G --setattr=thp=always -N 1
    - alloc --hugepages=512G --thp=always -N 1
    - Must also set env variable HUGETLB_MORECORE=yes (and need HSA_XNACK=1 also)

Burning Questions?