Hewlett Packard
Enterprise

HPE Cray Programming Environment User Guide: CSM on HPE Cray EX Systems (22.06) (S-8005)

Part Number: S-8005
Published: June 2022

HPE Cray Programming Environment User Guide: CSM on HPE Cray EX Systems (22.06) S-8005

Contents

1 Copyright and Version

CPE: 22.06-LocalBuild

Doc git hash: 0cca2aefe06a22116e11965dd9709912c9230352

Generated: Mon Jun 06 2022

2 About the HPE CPE User Guide: CSM on HPE Cray EX Systems

The *HPE Cray Programming Environment User Guide: CSM for HPE Cray EX Systems (S-8005)* includes programming environment and user access concepts, configuration information, component overviews, and relevant examples.

Please note that examples may include Work Load Manager (WLM) commands, and each WLM has its own syntax for interacting with the system. It is beyond the scope of this guide to provide complete details for WLMs. Examples are provided with the caveat that they may be out of sync with changes made by the WLM vendors. For details, see the appropriate WLM documentation.

Important: This guide assumes an EX system running the HPE Cray Operating System (COS). Technical details for a system running SUSE or Red Hat may differ.

This publication is intended for software developers, engineers, scientists, and other programming environment users.

2.1 Release Information

This publication supports CPE 22.06 installed on a system with:

- HPE Cray (EX) Supercomputer 22.03 recipe for CSM
- HPE Cray (EX) Supercomputer 22.02 recipe for CSM

2.2 Record of Revision

New in the CPE 22.06 publication:

- Added *Debug Applications with Sanitizers4hpc to Find Common Errors*
- Added direct links to current documentation that supports the programming environment user in *Available Documentation*

New in the CPE 22.05 publication:

- Added *Control Network Resources on Slingshot Networks with Slurm*
- Added *Control Network Resources on Slingshot Networks with PBS and PALS*

New in the CPE 22.04 publication:

- Modifications to *Parallel Application Launch Service*
  - The `cray` command is deprecated and no longer available; therefore, there is no need to confgure or authenticate before running an application (no `cray init` step)
  - Launch applications using `mpiexec` or `aprun` instead of `cray mpiexec` or `cray aprun`
  - Dedicated PALS commands replace the `cray pals` command
  - The `cray-pals` module must be loaded to use `mpiexec`, `aprun`, or the new PALS commands
- *Run an Application with PBS Pro and PAL* section replaced with *Run an Application with PBS in Interactive Mode* and *Run an Application with PBS in Batch Mode*
- Perftools support for HIP compiler added

New in the CPE 22.03 publication

- The `PrgEnv-nvidia` and `nvidia` module files are being deprecated in favor of `PrgEnv-nvhpc` and `nvhpc`, respectively, and may be removed in a later release. Therefore, the new module file names are used in this publication.
- Updated the current release information for *HPE Performance Analysis Tools User Guide* in *Additional Documentation*

New in the CPE 22.02 publication:

- Added modulefile information to *About the Intel Compiler*

New in the CPE 21.12 publication:

- A workaround for a known Slurm 21.08 bug is provided in *Workaround for Slurm Bug*
- Updated *Parallel Application Launch Service* to indicate that PALS supports the MPIR Process Acquisition Interface

New in the CPE 21.11 publication

- Removed `1.4` from the title, as it is a misnomer.
- Added *CPE Lmod Mixed Compiler Support*
- Added *Additional Documentation*

Publication Title	Date
HPE Cray Programming Environment User Guide: CSM on HPE Cray EX Systems (22.06) S-8005	June 2022
HPE Cray Programming Environment User Guide: CSM on HPE Cray EX Systems (22.05) S-8005	May 2022
HPE Cray Programming Environment User Guide: CSM on HPE Cray EX Systems (22.04) S-8005	April 2022
HPE Cray Programming Environment User Guide: CSM on HPE Cray EX Systems (22.03) S-8005	March 2022
HPE Cray Programming Environment User Guide: CSM on HPE Cray EX Systems (22.02) S-8005	February 2022
HPE Cray Programming Environment User Guide: CSM on HPE Cray EX Systems (21.12) S-8005	December 2021
HPE Cray Programming Environment User Guide: CSM on HPE Cray EX Systems (21.11) S-8005	November 2021
HPE Cray Programming Environment User Guide: CSM on HPE Cray EX Systems (21.10) S-8005	October 2021
HPE Cray Programming Environment User Guide: CSM on HPE Cray EX Systems (21.09) S-8005	September 2021
HPE Cray Programming Environment User Guide: CSM on HPE Cray EX Systems (21.08) S-8005	August 2021
HPE Cray Programming Environment User Guide: CSM on HPE Cray EX Systems (21.07) S-8005 Rev A	July 2021
HPE Cray Programming Environment User Guide: CSM on HPE Cray EX Systems (21.07) S-8005	July 2021
HPE Cray Programming Environment User Guide: CSM on HPE Cray EX Systems (21.06) S-8005	June 2021
HPE Cray Programming Environment User Guide: CSM on HPE Cray EX Systems (21.05) S-8005	May 2021
HPE Cray Programming Environment User Guide: CSM on HPE Cray EX Systems (21.04) S-8005	April 2021
HPE Cray Programming Environment User Guide EX Systems (21.03) S-8005	March 2021
HPE Cray EX Programming Environment User Guide (20.12) S-8005	December 2020
HPE Cray EX Programming Environment User Guide (20.11) S-8005	November 2020
HPE Cray Shasta Programming Environment User Guide (20.08) S-8005	August 2020
Cray Shasta Programming Environment User Guide (20.04) S-8005	April 2020
Cray Shasta Programming Environment User Guide (19.12) S-8005	January 2020

2.3 Available Documentation

HPE provides the following publications to support the programming environment user. They are available at HPE Support Center.

- HPE Cray Compiling Environment Release Overview (14.0) S-5212
- HPE Cray Clang C and C++ Quick Reference Guide (14.0) S-2179
- HPE Cray Fortran Reference Manual (14.0) S-3901
- HPE Performance Analysis Tools User Guide (22.05) S-8014

TIP: Searching for documentation can often lead to numerous hits. To narrow down the search results enter the publication and release number, e.g., "S-8014 22.05" (without quotation marks).

3 HPE Cray Programming Environment Components

The HPE Cray Programming Environment (CPE) provides tools designed to maximize developer productivity, application scalability, and code performance. It includes compilers, analyzers, optimized libraries, and debuggers. It also provides a variety of parallel programming models that allow users to make appropriate choices based on the nature of existing and new applications.

CPE uses build environment containers, providing the ability to compile, launch, and track job status. Containers enable users to store and retrieve files from both the local and shared system storage. Users can access CPE on User Access Node(s) (UAN) or through User Access Instances (UAI).

CPE components include:

- **Cray Compiling Environment (CCE)** - CCE consists of compilers that perform code analysis during compilation and automatically generate highly optimized code. Compilers support numerous command-line arguments to provide manual control over compiler behavior and optimization. Supported languages include Fortran, C and C++, and UPC (Unified Parallel C).
- **Cray Scientific and Math Libraries (CSML)** - CSML is a set of high performance libraries that provide portability for scientific applications by implementing APIs for arrays (NetCDF), sparse and dense linear algebra (BLAS, LAPACK, ScaLAPACK) and fast Fourier transforms (FFTW).
- **Cray Message Passing Toolkit (CMPT)** - CMPT is a collection of software libraries enabling data transfers between nodes running in parallel applications. CMPT comprises the Message Passing Interface (MPI) and OpenSHMEM parallel programming models. CMPT libraries support practical, portable, efficient, and flexible mechanisms for performing data transfers between parallel processes.
- **Cray Environment Setup and Compiling Support (CENV)** - CENV provides libraries that support code compilation and setting up the development environment. It comprises compiler drivers, `hugepages` utility, and the CPE API, which is a software package used for building module files.
- **Cray Performance Measurement & Analysis Tools (CPMAT)** - CPMAT provides tools to analyze the performance and behavior of programs that are run on Cray systems, and a Performance API (PAPI).
- **Cray Debugging Support Tools (CDST)** - CDST provides debugging tools, including `gdb4hpc`, `Valgrind4hpc` and `Sanitizers4hpc`.

3.1 Cray Environment Setup and Compiling Support

HPE Cray Environment (CENV) provides libraries that support code compilation and development environment setup. It comprises compiler drivers, utilities, and the CPE API (`craype-api`), which is a software package used for building module files.

3.1.1 Modules and Modulefiles

CPE Environment Modules enables users to modify their environment dynamically by using modulefiles. The `module` command provides a user interface to the Modules package. The `module` command system interprets modulefiles, which contain Tool Command Language (Tcl) code, and dynamically modifies shell environment variables such as `PATH` and `MANPATH`.

Sites can alternatively enable `Lmod` to handle modules. Both module systems use the same module names and syntax shown in command-line examples.

TIP: Use *either* Environment Modules or `Lmod` on a per-system basis. The systems are mutually exclusive and cannot both run on the same system.

The `/etc/cray-pe.d/cray-pe-configuration.sh` and `/etc/cray-pe.d/cray-pe-configuration.csh` configuration files allow sites to customize the default environment. System administrator can also create modulefiles for a product set to support user-specific needs. For more information about the Environment Modules software package, see the `module(1)` and `modulefile(4)` manpages.

3.1.2 Lmod

In addition to the default Environment Modules system, CPE offers support for Lmod as an alternative module management system. Lmod is a Lua-based module system that loads and unloads modulefiles, handles path variables, and manages library and header files. The CPE implementation of Lmod is hierarchical, managing module dependencies and ensuring any module a user has access to is compatible with other loaded modules. Features include:

- Lmod is set up to automatically load a default set of modules. The default set includes one each of compiler, network, CPU, and MPI modules. Users may choose to load different modules. However, it is recommended that, at minimum, a compiler, network, CPU, and an MPI module be loaded, to ensure optimal assistance from Lmod.
- Lmod supports loading multiple different compiler modules concurrently by loading a dominant core-compiler module and one or more support mixed-compiler modules. See *Lmod Mixed Compiler Support* for details.

- Lmod uses "families" of modules to flag circular conflicts, which is most apparent when module details are displayed through `module show` and when users attempt to load conflicting modules.
- Environment Modules and Lmod modules use the same names, so all command examples work similarly.

TIP: Environment Modules and Lmod are mutually exclusive, and both cannot run on the same system. Contact the system administrator about setting Lmod as the default module management system.

For more Lmod information, see ***The User Guide for Lmod***: https://lmod.readthedocs.io/en/latest/010_user.html.

3.1.3 About Hugepages

Note: This hugepages implementation only works in Cray's Operating System. If using another Linux distro, use the hugepages implementation appropriate for that distro.

Hugepages are virtual memory pages which are bigger than the default base page size of 4K bytes. Hugepages can improve memory performance for common access patterns on large data sets. Hugepages also increase the maximum size of data and text in a program accessible by the high-speed network. Access to huge pages is provided through a virtual file system called `hugetlbfs`. Every file on this file system is backed by hugepages and is directly accessed with `mmap()` or `read()`.

The `libhugetlbfs` library allows an application to use hugepages more easily by directly accessing the `hugetlbfs` file system. A user may enable `libhugetlbfs` to back application text and data segments.

Use hugepages for the following:

- For SHMEM applications, map the static data and/or private heap onto huge pages.
- For applications written in Unified Parallel C and other languages based on the PGAS programming model, to map the static data and/or private heap onto huge pages.
- For MPI applications, map the static data and/or heap onto huge pages.
- For applications using shared memory that are concurrently registered with high-speed network drivers for remote communication.
- For applications doing heavy I/O.
- To improve memory performance for common access patterns on large data sets.

Note: On x86 processors, the only page sizes supported by the processor are 4K, 2M, and 1G.

See the `intro_hugepages(3)` man page for more details.

To use hugepages, load the appropriate `craype-hugepages` at link time. Possible values are:

- `craype-hugepages128K`
- `craype-hugepages512K`
- `craype-hugepages2M`
- `craype-hugepages4M`
- `craype-hugepages8M`
- `craype-hugepages16M`
- `craype-hugepages32M`
- `craype-hugepages64M`
- `craype-hugepages128M`
- `craype-hugepages256M`
- `craype-hugepages512M`
- `craype-hugepages1G`
- `craype-hugepages2G`

For example, to use 2 megabyte `hugepages`:

```
user@hostname> module load craype-hugepages2M
```

3.2 About Cray Compiling Environment

Module: `PrgEnv-cray`

Command: `ftn`, `cc`, `CC`

Compiler-specific manpages: `crayftn(1)`, `craycc(1)`, `crayCC(1)` - available only when the compiler module is loaded

Online help: `ftn -help`, `cc -help`, `CC -help`

Documentation: See *Available Documentation*

To use the Cray Compiling Environment (CCE), load the `PrgEnv-cray` module.

```
user@hostname> module load PrgEnv-cray
```

CCE provides Fortran, C and C++ compilers that perform substantial analysis during compilation and automatically generate highly optimized code. The compilers support numerous command-line arguments that enable manual control over compiler behavior and optimization. For more information about the Cray Fortran, C, and C++ compiler command-line arguments, see the `crayftn(1)`, `craycc(1)`, and `crayCC(1)` manpages, respectively.

PrgEnv modules provide wrappers (`cc`, `CC`, `ftn`) for both CCE and third-party compiler drivers. These wrappers call the correct compiler with appropriate options to build and link applications with relevant libraries as required by modules loaded. (Only dynamic linking is supported.) These wrappers replace direct calls to compiler drivers in Makefiles and build scripts. For more information about compiler pragmas and directives, see the `intro_directives(1)` manpages.

One of the most useful compiler features is the ability to generate annotated loopmark listings showing what optimizations were performed and their locations. Together with compiler messages, these listings can help locate areas in the code that are compiling without error but are not fully optimized. For more detailed information about generating and reading loopmark listings, see `crayftn(1)`, `craycc(1)`, and `crayCC(1)` manpages, the ***Cray Fortran Reference Manual (S-3901)***, and ***HPE Performance Analysis Tools User Guide (S-8014)***. See *Available Documentation* for direct links to these publications.

In many cases, code that is not properly optimizing can be corrected without substantial recoding by applying the right pragmas or directives. For more information about compiler pragmas and directives, see the `intro_directives(1)` man page.

3.3 Third-Party Compilers

CPE supports the following third-party compilers:

- AOCC
- Intel
- GNU
- NVIDIA

The compilers and their respective dependencies, including wrappers and mappings (e.g., mapping `cc` to `gcc` in `PrgEnv-gnu`) are loaded using the `module load <modulename>` command. For example,

```
user@hostname> module load PrgEnv-gnu
```

3.3.1 About AOCC

Module: `PrgEnv-aocc`

Command: `ftn`, `cc`, `CC`

Documentation: https://developer.amd.com/amd-aocc/#userguide

CPE enables, but does not bundle, the AMD Optimizing C/C++ Compiler (AOCC). CPE provides a bundled package of support libraries to install into the programming environment to enable AOCC and CPE utilities such as debuggers and performance tools.

- If not available on the system, contact a system adminstrator to install AOCC and the support bundle.
- To us AOCC, load the `PrgEnv-aocc` module:

  ```
  user@hostname> module load PrgEnv-aocc
  ```

3.3.2 About the Intel Compiler

Module: `PrgEnv-intel`

Command: `ftn`, `cc`, `CC`

Documentation: https://software.intel.com/content/www/us/en/develop/tools/oneapi.html

CPE enables, but does not bundle, the Intel® oneAPI for Linux compiler. CPE provides a bundled package of support libraries to install into the programming environment to enable the Intel compiler and CPE utilities such as debuggers and performance tools.

Intel oneAPI includes their “classic” compilers (`icc`, `icpc` and `ifort`) as well as new versions of each of those (`icx`, `icpx`, and `ifx`). Because `ifx` is an experimental Fortran compiler, Intel encourages users to stay with the “classic” Fortran (`ifort`) compiler along with their new C/C++ (`icx` and `icpx`).

Because all of the Intel compilers come in the same package, the `PrgEng-intel` meta-module now has three options for the “intel” sub-module. They are:

1. **intel** (`icx`, `icpx`, `ifort`) - PrgEnv-intel defaults to this given it is Intel’s recommendation
2. **intel-classic** (`icc`, `icpc`, `ifort`) – All “classic” Intel compilers
3. **intel-oneapi** (`icx`, `icpx`, `ifx`) – All “new” Intel compilers, where `ifx` is “beta” per Intel

- To use the Intel DPC+/C++ compiler, load the `PrgEnv-intel` module:

  ```
  user@hostname> module load PrgEnv-intel
  ```

- To use the Intel C++ Compiler Classic instead, switch to the `intel-classic` module:

  ```
  user@hostname> module swap intel intel-classic
  ```

3.3.3 About GNU

Module: `PrgEnv-gnu`

Command: `ftn`, `cc`, `CC`

Compiler-specific manpages: `gcc(1)`, `gfortran(1)`, `g++(1)` - available only when the compiler module is loaded.

Documentation: https://gcc.gnu.org/onlinedocs/

CPE bundles and enables the open-source GNU Compiler Collection (GCC).

- To use GCC, load the `PrgEnv-gnu` module.

  ```
  user@hostname> module load PrgEnv-gnu
  ```

3.3.4 About the Nvidia Compiler

Module: `PrgEnv-nvhpc`

Command: `ftn`, `cc`, `CC`

Documentation: https://docs.nvidia.com/hpc-sdk/compilers/hpc-compilers-user-guide/index.html

For systems with Nvidia GPU blades, the `PrgEnv-nvhpc` compiler module is available and enables functions for various CPU/GPU combinations:

- CPU-only applications
- CPU/GPU applications with CUDA and CPU code in the same file
- CPU/GPU applications that use OpenMP target offload or OpenACC

3.4 Programming Languages

The following programming languages are bundled with and supported by the CCE:

- **Fortran** - The CCE Fortran compiler supports the Fortran 2018 standard (ISO/IEC 1539:2018), with some exceptions and deferred features as noted elsewhere.

  - Documentation is available in ***HPE Cray Fortran Reference Manual (S-3901)*** and also in manpages, beginning with the `crayftn(1)` manpage. Where information in the manuals differs from the manpage, the information in the manpage is presumed to be more current.
  - For the current direct link to the reference manual, see ***Available Documentation***.
- **C/C++** - The default C/C++ compiler is based on Clang/LLVM.
  - Supports **Unified Parallel C (UPC)**, an extension of the C programming language designed for high performance computing on large-scale parallel systems.
  - Documentation is provided at https://clang.llvm.org/docs/UsersManual.html, in the `clang(1)` manpage, and in ***HPE Cray Clang C and C++ Quick Reference (S-2179)***.
  - For the current direct link to the quick reference, see ***Available Documentation***.

The following third-party programming languages are bundled with the Programming Environment:

- **Python**
- **R**

3.5 Cray Scientific and Math Libraries

Modules: `cray-libsci`, `cray-fftw3`

Manpages: `intro_libsci(3s)`, `intro_fftw3(3)` - available only when the associated module is loaded.

The Cray Scientific and Math Libraries (CSML, also known as LibSci) are a collection of numerical routines optimized for best performance on Cray systems. These libraries satisfy dependencies for many commonly used applications on Cray systems for a wide variety of domains. When the module for a CSML package (such as `cray-libsci` or `cray-fftw`) is loaded, all relevant headers and libraries for these packages are added to the compile and link lines of the `cc`, `ftn`, and `CC` CPE drivers.

The CSML collection contains the following Scientific Libraries:

- BLAS (Basic Linear Algebra Subroutines)
- BLACS (Basic Linear Algebra Communication Subprograms)
- CBLAS (Collection of wrappers providing a C interface to the Fortran BLAS library)
- IRT (Iterative Refinement Toolkit)
- LAPACK (Linear Algebra Routines)
- LAPACKE (C interfaces to LAPACK Routines)
- ScaLAPACK (Scalable LAPACK)
- libsci_acc (library of Cray-optimized BLAS, LAPACK, and ScaLAPACK routines)
- NetCDF (Network Common Data Format)
- FFTW3 (the Fastest Fourier Transforms in the West, release 3)

3.6 Cray Message Passing Toolkit

Module: `cray-mpich`

Manpage: `intro_mpi(3)` - available only when the associated module is loaded.

Website: http://www.mpi-forum.org/

Cray Message Passing ToolKit (CMPT) is a collection of message passing libraries to aid in parallel programming.

MPI is a widely used parallel programming model that establishes a practical, portable, efficient, and flexible standard for passing messages between ranks in parallel processes. Cray MPI is derived from Argonne National Laboratory MPICH and implements the MPI-3.1 standard as documented by the MPI Forum in ***MPI: A Message Passing Interface Standard, Version 3.1***.

MPI supports both OpenFabrics Interfaces (OFI) and Unified Communication X (UCX) network modules with OFI typically run as the default version. In some situations we recommend unloading the OFI module and then loading the UCX module, rerunning, and comparing for optimal performance. These versions are binary compatible; therefore, recompiling or relinking an application is not necessary. In addition to some performance differences where one module might perform better than the other for a given application, the OFI version has a known limitation when establishing initial connections for applications that use an alltoall communication pattern or a many-to-one pattern at very high scale. In these situations we recommend trying the UCX version and comparing it to the performance of running with the OFI network module.

Support for MPI varies depending on system hardware. To see which functions and environment variables the system supports, check the `intro_mpi(3)` manpage. Because the OFI and UCX MPI versions are quite different, different `intro_mpi` man pages are displayed depending on which module is loaded. These man pages are a good source of additional information for the currently running network module, including the different runtime environment variables that can further control performance.

3.7 OpenSHMEMX

Module: `cray-openshmemx`

Manpage: `intro_shmem(3)` - available only when the associated module is loaded.

Website: https://pe-cray.github.io/cray-openshmemx/

OpenSHMEM is a Partitioned Global Address Space (PGAS) library interface specification. OpenSHMEM provides a standard Application Programming Interface (API) for SHMEM libraries to aid portability and facilitate uniform predictable results of OpenSHMEM programs by explicitly stating the behavior and semantics of the OpenSHMEM library calls.

SHMEM has a long history as a parallel programming model. For the past two decades SHMEM library implementations in systems evolved through different generations.

OpenSHMEMX is a proprietary OpenSHMEM implementation that is OpenSHMEM specification version 1.4 compliant. Refer to the `intro_shmem(3)` manpage for more details.

3.8 DSMML

Module: `cray-dsmml`

Manpage `intro_dsmml(3)` - available only when the associated module is loaded.

Website https://pe-cray.github.io/cray-dsmml/

Distributed Symmetric Memory Management Library (DSMML) is a proprietary memory management library. DSMML is a standalone memory management library for maintaining distributed shared symmetric memory heaps for top-level PGAS languages and libraries like Coarray Fortran, UPC, and OpenSHMEM. DSMML allows user libraries to create multiple symmetric heaps and share information with other libraries. Through DSMML, interoperability can be extracted between PGAS programming models.

Refer to the `intro_dsmml(3)` manpage for more details.

3.9 Debugger Support Tools

CPE ships with numerous debugging tools.

3.9.1 HPE Tools

A number of tools are included:

- **Gdb4hpc** - A command line interactive parallel debugger that allows debugging of the application at scale. Helps diagnose hangs and crashes. A good all-purpose debugger to track down bugs, analyze hangs, and determining the causes of crashes.
- **Valgrind4hpc** - A parallel debugging tool used to detect memory leaks and parallel application errors.
- **Sanitizers4hpc** - A parallel debugging tool used to detect memory access or leak issues at runtime using information from LLVM Sanitizers.
- **Stack Trace Analysis Tool (STAT)** - A single merged stack backtrace tool to analyze application behavior at the function level. Helps trace down the cause of crashes.
- **Abnormal Termination Processing (ATP)** - A scalable core file generation and analysis tool for analyzing crashes, with a selection algorithm to determine which core files to dump. Helps determine the cause of crashes.
- **Cray Comparative Debugger (CCDB)** - CCDB is not a traditional debugger, but rather a tool to run and step through two versions of the same application side by side to help determine where they diverge.

All CPE debugger tools support C/C++, Fortran, and Universal Parallel C (UPC).

3.9.2 Third Party Debugging Tools

There are two third-party debugging tools available:

- The **ARM Forge** tool suite (https://www.arm.com/products/development-tools/server-and-hpc/forge)

  TIP: In order for forge-21.0 to successfully launch an application on an HPE Cray EX system running CSM 1.3 or later, perform the following configuration on a UAI/UAN:

  ```
  [user@uai ~]$ export ALLINEA_SESSION_PATH=<ProjectedDirectory>
  ```

  For example:

  ```
  user@hostname> ssh <USERNAME@UAI_IP_ADDRESS>
  [user@uai ~]$ export ALLINEA_SESSION_PATH=/lus/<USERNAME>
  ```

- The **Perforce TotalView** debugger (https://totalview.io)

3.9.3 Tool Infrastructure

CPE provides several tools for tool developers to enhance their own debuggers for use with the CPE:

- **Common Tools Interface (CTI)** - Offers a simple, WLM agnostic API to support tools across all Cray systems.
- **Multicast Reduction Network (MRNET)** - Provides a scalable communication tool for libraries.
- **Dyninst** - Provides dynamic instrumentation libraries.

3.10 Cray Performance Measurement and Analysis Tools

The Cray Performance Measurement and Analysis Tools (CPMAT) suite reduces the time needed to port and tune applications. It provides an integrated infrastructure for measurement, analysis, and visualization of computation, communication, I/O, and memory utilization to help users optimize programs for faster execution and more efficient computing resource usage.

The toolset allows developers to perform sampling, profile, and trace experiments on executables, extracting information at the program, function, loop, and line level. Programs written in Fortran and C/C++ (including UPC) and HIP, with MPI, SHMEM, OpenMP, CUDA, or a combination of these programming languages and models are supported. It supports profiling applications built with CCE, AMD, and GNU compilers.

Performance analysis consists of three basic steps:

1. Instrument the program to specify what kind of data to collect under what conditions.
2. Execute the instrumented executable to generate and capture data.
3. Analyze the resulting data.

There are three programming interfaces available:

- `perftools-lite-*` - Simple interface that produces reports to `stdout`. There are four `perftools-lite` submodules:
  - `perftools-lite` - Lowest overhead sampling experiment identifies key program bottlenecks.
  - `perftools-lite-events` - Produces a summarized trace; a good tool for detailed MPI statistics, including synchronization overhead.
  - `perftools-lite-loops` - Provides loop work estimates (must be used with CCE).
  - `perftools-lite-hbm` - Reports memory traffic information (CCE, x86-64 systems only). See the `perftools-lite(4)` manpage for details.
- `perftools` - Advanced interface provides full-featured data collection and analysis capability, including full traces with timeline displays. It includes the following components:
  - `pat_build` - Utility instruments programs for performance data collection.
  - `pat_report` - After using `pat_build` to instrument the program, set runtime environment variables, and executing the program, use `pat_report` to generate text reports from the resulting data and export the data for use in other applications. See the `pat_report(1)` manpage for details.
  - `CrayPat` runtime library - Collects specified performance data during program execution. See the `intro_craypat(1)` manpage for details.
- `pat_run` - Launches a dynamically linked program instrumented for performance analysis. After successfully run, collected data may be explored further with the `pat_report` and Cray Apprentice2 tools. See the `pat_run(1)` manpage for details.

Also included:

- `PAPI` - The `PAPI` library, from the Innovative Computing Laboratory at the University of Tennessee in Knoxville, is distributed with the performance tools. PAPI allows applications or custom tools to interface with hardware performance counters made available by the processor, network, or accelerator vendor. Performance tools' components use PAPI internally for CPU, GPU and network performance counter collection for derived metrics, observations, and performance reporting. A simplified user interface, which does not require the source code modification of using PAPI directly, is provided for accessing counters.

- Cray Apprentice2 - An interactive X Window System tool for visualizing and manipulating performance analysis data captured during program execution.
- `pat_view` – Aggregates and presents multiple sampling experiments for program scaling analysis. See the `pat_view(1)` manpage for more information.
- Reveal - Extends performance tools technology by combining performance statistics and program source code visualization with compiler optimization feedback to better identify and exploit parallelism, and to pinpoint memory bandwidth sensitivities in an application. Reveal lets users navigate source code to highlighted dependencies or bottlenecks during optimization. Using the program library provided by CCE and the performance data collected, the user can navigate source code to understand which high-level loops could benefit from OpenMP parallelism from loop-level optimizations such as exposing vector parallelism. Reveal provides dependency and variable scoping information for those loops and assists the user with creating parallel directives.

Use performance tools to:

- Identify bottlenecks
- Find load-balance and synchronization issues
- Find communication overhead issues
- Identify loops for parallelization
- Map memory bandwidth utilization
- Optimize vectorization within application code
- Collect application energy consumption information
- Collect scaling information for application code
- Interpret performance data

More information is available in ***HPE Performance Analysis Tools User Guide (S-8014)***. For the current direct link to this publication, see *Available Documentation*.

3.11 About CPE Deep Learning Plugin

Modules: `craype-dl-plugin-py3, craype-dl-plugin-py2`

Commands: `import dl_comm` as cdl, `help(cdl)`, `help(cdl.gradients)`

Manpage: `intro_dl_plugin(3)`

The CPE Deep Learning Plugin (CPE DL Plugin) is a highly tuned communication layer for performing distributed deep learning training. The CPE DL Plugin provides a high performance gradient-averaging operation and routines to facilitate process identification, job size determination, and broadcasting of initial weights and biases. The routines can be accessed through the plugin's C or Python APIs. The Python API provides support for TensorFlow, PyTorch, Keras, and NumPy.

For more information about CPE DL Plugin directives, see the `intro_dl_plugin(3)` manpage.

4 User Access Service

The User Access Service (UAS) is a containerized service managed by Kubernetes that enables application developers to create and run user applications. UAS runs on a non-compute node (NCN) that is acting as a Kubernetes worker node.

Users launch a User Access Instance (UAI) using the `cray` command. Users can also transfer data between the Cray system and external systems using the UAI.

When a user requests a new UAI, the UAS service returns status and connection information to the newly created UAI. External access to UAS is routed through a node that hosts gateway services.

The timezone inside the UAI container matches the timezone on the host on which it is running, For example, if the timezone on the host is set to CDT, the UAIs on that host will also be set to CDT.

Table 2: ***UAS Components***

Component	Function/Description
User Access Instance (UAI)	An instance of UAS container
`uas-mgr`	Manages UAI life cycles

Table 3: ***UAS Container Contents***

Container Element	Components
Operating system	SLES15 SP1
`kubectl` command	Utility to interact with Kubernetes
`cray` command	Command that allows users to create, describe, and delete UAIs

Use `cray uas list` to list the following parameters for a UAI.

Note that the example values below are used throughout the UAS procedures. They are used as examples only. Users should substitute with site-specific values.

Table 4: ***UAS Parameters***

Parameter	Description	Example value
`uai_connect_string`	The UAI connection string	`ssh user@203.0.113.0 -i \ ~/.ssh/id_rsa`
`uai_img`	The UAI image ID	`registry.local/cray/ \ cray-uas-sles15sp1-slurm:latest`
`uai_name`	The UAI name	`uai-user-be3a6770`
`uai_status`	The state of the UAI	`Running: Ready`
`username`	The user who created the UAI	`user`
`uai_age`	The age of the UAI	`11m`
`uai_host`	The node hosting the UAI	`ncn-m001`

4.1 UAS Limitations

Functionality Not Currently Supported by the User Access Service

- Lustre (`lfs`) commands within the UAS service pod
- Executing Singularity containers within the UAS service
- Building Docker containers within the UAS environment
- Building containerd containers within the UAS environment
- `dmesg` cannot run inside a UAI due to container security limitations
- Users cannot `ssh` from `ncn-m001` to a UAI. This is because UAIs use LoadBalancer IPs on the Customer Access Network (CAN) instead of NodePorts and the LoadBalancer IPs are not accessible from `ncn-m001`.

Other Limitations

- There is a known issue where X11 traffic may not forward DISPLAY correctly if the user logs into an NCN node before logging into a UAI.
- The `cray uas uais` commands are not restricted to the user authenticated with `cray auth login`.

Limitations Related To Restarts

Changes made to a running UAI will be lost if the UAI is restarted or deleted. The only changes in a UAI that will persist are those written to an externally mounted file system (such as Lustre or NFS).

A UAI may restart due to an issue on the physical node, scheduled node maintenance, or intentional restarts by a site administrator. In this case, any running processes (such as compiles), Slurm interactive jobs, or changes made to the UAI (such as package installations) are lost.

If a UAI restarts on a node that was recently rebooted, some of the configured volumes may not be ready and it could appear that content in the UAI is missing. In this case, restart the UAI.

5 Available Workload Managers

5.1 Slurm

CPE provides the required infrastructure to support running the Slurm workload manager. Slurm is installed during the installation process and can be manually configured later by a system administrator.

More detailed information, including tutorials, documentation, and an FAQ, can be found on http://slurm.schedmd.com/.

5.1.1 Workaround for Slurm Bug

A known bug exists in Slurm version 21.08 that may result in job launches hanging. However, the bug only exists in Slurm versions 21.08.1 to 21.08.4. It was fixed in version 21.08.5. Try the following if experiencing hanging jobs:

1. Run `srun -V` to determine if the system is running Slurm 21.08.
2. If the system is running Slurm 21.08:
    a. Look for the configuration file `gres.conf` in the Slurm configuration directory, usually `/etc/slurm/gres.conf` or `/etc/opt/slurm/gres.conf`. This file contains a list of the system's valid Generic RESources (GRES) settings.
    b. If a GRES configuration file does not exist, set the environment variable `CTI_SLURM_DAEMON_GRES` to an empty string. (This automatically disables GRES, which is necessary because Slurm 21.08 does not accept `--gres=none` to disable GRES.)
    c. If a GRES configuration file exists, set the environment variable `CTI_SLURM_DAEMON_GRES` to one of the system's valid GRES settings found in `gres.conf`.

5.1.2 Control Network Resources on Slingshot Networks with Slurm

On systems with HPE Slingshot networks using Slurm, network resource allocation for applications are controlled using the `srun --network` option. The network option consists of a list of comma-separated values:

- `def_<rsrc>=<val>` - Per-CPU reserved allocation for this resource.
- `res_<rsrc>=<val>` - Per-node reserved allocation for this resource. If set, overrides the per-CPU allocation.
- `max_<rsrc>=<val>` - Maximum per-node limit for this resource.
- `depth=<depth>` - Multiplier for per-CPU resource allocation. Default is the number of reserved CPUs on the node.

The network resources are:

- `txqs` - Transmit command queues. The default is 3 per-CPU, maximum 1024 per-node.
- `tgqs` - Target command queues. The default is 2 per-CPU, maximum 512 per-node.
- `eqs` - Event queues. The default is 8 per-CPU, maximum 2048 per-node.
- `cts` - Counters. The default is 2 per-CPU, maximum 2048 per-node.
- `tles` - Trigger list entries. The default is 1 per-CPU, maximum 2048 per-node.
- `ptes` - Portable table entries. The default is 8 per-CPU, maximum 2048 per-node.
- `les` - List entries. The default is 134 per-CPU, maximum 65535 per-node.
- `acs` - Addressing contexts. The default is 4 per-CPU, maximum 1024 per-node.

For example, `srun --network def_txqs=4,res_tgqs=128,max_eqs=1024` reserves 4 transmit command queues per CPU, 128 target command queues per node, and limits application usage to 1024 event queues per node.

5.2 PBS Professional

CPE provides the required infrastructure to support running Altair's PBS Professional (PBS Pro) workload manager. PBS Pro uses the Parallel Application Launch Service (PALS) to launch jobs. PBS Pro is installed during installation process and can be manually configured later by a system administrator.

For more detailed information and documentation, go to https://www.pbsworks.com.

5.3 Parallel Application Launch Service

The Parallel Application Launch Service (PALS) is a containerized application that provides a REST API for application launch. PALS serves as a launcher for third-party workload managers (WLMs) that do not provide their own launchers, enabling parallel applications to be run as a unit on multiple compute nodes while coordinating their execution and providing whole-application reporting. PALS enables the WLM to control node usage and allocation, and runs on the compute node alongside the WLM daemon.

- At this time PALS supports only the PBS Professional workload manager.
- PALS does not support the Slurm workload manager, as Slurm provides its own launcher and PMI support plugin.
- PALS supports the MPIR Process Acquisition Interface, which is used by tools such as debuggers and performance analyzers to locate MPI processes that are part of an MPI job.

5.3.1 Launch an Application with PALS

Applications are launched by PALS from a PBS job using the `mpiexec` or `aprun` commands. **TIP:** The `cray-pals` module must be loaded (i.e., `module load cray-pals`) before running either of these commands.

See *Run an Application with PBS Pro in Batch Mode* and *Run an Application with PBS Pro in Interactive Mode* for examples. Further details are also available in the `mpiexec(1)` and `aprun(1)` man pages when the `cray-pals` module is loaded.

5.3.2 The `mpiexec` Command

Options and arguments for `mpiexec` are described in detail in the `mpiexec(1)` man page. **TIP:** Note the following concerning the `--cpu-bind` and `--mem-bind` options.

- The `--cpu-bind` (CPU binding) option is formatted as `[verbose,]<keyword>[:arguments]`, where the valid keywords are:
  - `none` - No CPU binding
  - `numa`, `socket`, `core`, `thread` - Bind ranks to the specified hardware
  - `depth` - Bind ranks to the number of threads in the argument
  - `list` - Bind ranks to colon-separated rangelists of CPUs
  - `mask` - Bind ranks to comma-separate bitmasks of CPUs
- The `--mem-bind` (NUMA-mode memory binding) option is formatted as `[verbose,]<keyword>[:arguments]`, where the valid keywords are:
  - `none` - No memory binding- `local` - Restrict each rank to use only its own NUMA node memory
  - `list` - Bind ranks to colon-separated rangelists of NUMA nodes
  - `mask` - Bind ranks to comma-separate bitmasks of NUMA nodes

5.3.3 Commonly Used PALS Commands

In addition to `mpiexec` and `aprun`, the following PALS commands are commonly used in the command-line interface. **TIP:** The `cray-pals` module must be loaded (i.e., `module load cray-pals`) before running any of these commands. See the `palstat(1)`, `palsig(1)`, `palscmd(1)`, and `palscp(1)` man pages for complete details.

Command	Action
`palstat`	List the applications currently running on the system.
`palstat <APID>`	Gather information about the application `<APID>`.
`palsig -s <SIGNAL> <APID>`	Send a signal to a running application. If `<SIGNAL>` is not specified, the default is `SIGTERM`, to terminate the application. In contrast, to stop a process, enter `palsig -s SIGSTOP <APID>`. To continue a stopped process, enter `palsig -s SIGCONT <APID>`.

5.3.4 PALS Environment Variables

The following environment variables are set by PALS for each application rank, in addition to other application environment variables.

Env Variable	Purpose
`PALS_APID`	Unique application identifier

Env Variable	Purpose
`PALS_RANKID`	Application rank identifier
`PALS_LOCAL_RANKID`	Node local rank identifier
`PALS_DEPTH`	Number of CPUs per rank for the application
`PALS_NODEID`	Application relative node identifier

5.3.5 Control Network Resources on Slingshot Networks with PBS and PALS

On systems with HPE Slingshot networks using PALS, network resource allocation for applications may be controlled using the PALS `mpiexec` or `aprun --network` option. The network option consists of a list of semicolon-separated values:

- `def_<rsrc>=<val>` - Per-CPU reserved allocation for this resource.
- `max_<rsrc>=<val>` - Maximum per-node limit for this resource.

The network resources are:

- `txqs` - Transmit command queues. The default is 2 per-CPU, maximum 1024 per-node.
- `tgqs` - Target command queues. The default is 1 per-CPU, maximum 512 per-node.
- `eqs` - Event queues. The default is 4 per-CPU, maximum 2047 per-node.
- `cts` - Counters. The default is 1 per-CPU, maximum 2047 per-node.
- `tles` - Trigger list entries. The default is 1 per-CPU, maximum 2048 per-node.
- `ptes` - Portable table entries. The default is 8 per-CPU, maximum 2048 per-node.
- `les` - List entries. The default is 16 per-CPU, maximum 16384 per-node.
- `acs` - Addressing contexts. The default is 4 per-CPU, maximum 1023 per-node.

For example, `mpiexec --network def_txqs=4;max_eqs=1024` reserves 4 transmit command queues per CPU and limits application usage to 1024 event queues per node.

6 Log in to an EX System

Users access an EX system through either a User Access Instance (UAI) or a User Access Node (UAN).

6.1 Log in to a User Access Instance (UAI)

UAIs are containerized environments managed by Kubernetes and hosted on a non-compute node (NCN) acting as a worker (such as ncn-m001). The user must create a UAI before they are able to log in.

To log into an existing UAI, use the `uai_connect_string` created in ***Create a UAI from a Specific Image in Legacy Mode***.

```
user@hostname> ssh <USERNAME@UAI_IP_ADDRESS> -i ~/.ssh/id_rsa
```

When challenged for a password, enter the passphrase specified when the ssh key was generated.

6.2 Log in to a User Access Node (UAN)

UANs are bare metal nodes configured by the system administrator and running an image based off the compute node image. To log into a UAN, use the `ssh` command. When challeged for a password, enter your username password.

```
user@hostname> ssh <uan_ip>
Last login: Thu Aug 26 09:28:38 CDT 2021 from 172.96.255.255

This node is running Cray's Linux Environment version 1.3.0

############################################################################

 .d8888b.                               888     888        d8888 888b    888
d88P  Y88b                              888     888       d88888 8888b   888
888    888                              888     888      d88P888 88888b  888
888         888d888 8888b.  888  888    888     888     d88P 888 888Y88b 888
888         888P"      "88b 888  888    888     888    d88P  888 888 Y88b888
```

```
888    888 888    .d888888 888  888      888     888  d88P   888 888  Y88888
Y88b  d88P 888    888  888 Y88b 888      Y88b. .d88P d8888888888 888   Y8888
 "Y8888P"  888    "Y888888  "Y88888       "Y88888P" d88P     888 888    Y888
                                 888
                            Y8b d88P
                             "Y88P"

You have logged into a Cray Shasta Premium User Access Node

Hostname:     uan01
Distribution: SLES 15.1 1
CPUS:         128
Memory:       257.5GB
Configured:   2021-08-26


Please contact your IT system admin for any support requests.
#############################################################################
```

6.3 Differences Between UAI and UAN

	UAI	UAN
Type	Container managed by Kubernetes	Baremetal Server
Single/Multiuser	Single User	Multiuser
Access	- SSH with user ssh keys	- SSH with user login/password
	- UAI must be initially created by user	- Always up, no need for user to create
	- No fixed hostname or IP address	- Fixed hostname and IP address
	- SSH uses non-standard ports	- SSH uses standard ports
Image	- Default image based compute image	- Default image based on compute image
	- Each user can have their own image	- All users share the same image and process space
	- Each user is in their own process space	
craycli	yes	yes
Data Transfer	SCP using UAI IP and Port number and user SSH keys	SCP using UAN and username/password
Troubleshooting	Kubernetes logs and techniques plus Linux techniques	Normal Linux logs and techniques

6.4 Create a UAI from a Specific Image in Legacy Mode

PREREQUISITES

- A public SSH key
- Initialize the cray CLI for non-admin users

OBJECTIVE

Create a UAI that uses a specific, registered image.

PROCEDURE

1. List available UAS images.

   ```
   ncn-m001# cray uas images list
   default_image = "registry.local/cray/cray-uas-sles15sp1-slurm:latest"
   image_list = [ "registry.local/cray/cray-uas-sles15sp1-slurm:latest",
   "registry.local/cray/cray-uas-sles15sp1:latest",]
   ```

Troubleshooting: If the Cray CLI has not been initialized, the CLI commands will not work. See ***Configure the Cray Command Line Interface (CLI)*** in the ***HPE Cray EX System Administration Guide S-8001*** for more information.

2. Create a new UAI.

```
ncn-m001# cray uas create --publickey ~/.ssh/id_rsa.pub
uai_age = "0m"
uai_connect_string = "ssh vers@10.102.10.249"
uai_host = "ncn-m002"
uai_img = "registry.local/cray/cray-uai-compute:latest"
uai_ip = "10.102.10.249"
uai_msg = "ContainerCreating"
uai_name = "uai-vers-1dcf28e5"
uai_status = "Waiting"
username = "vers"[uai_portmap]
```

   To create a UAI with a non-default image, add the `--imagename` argument with the above command.

3. Verify the UAI is in the “Running: Ready” state.

```
ncn-m001:~ # cray uas list
uai_age = "1m"
uai_connect_string = "ssh vers@10.102.10.249"
uai_host = "ncn-m002"
uai_img = "registry.local/cray/cray-uai-compute:latest"
uai_ip = "10.102.10.249"
uai_msg = "ContainerCreating"
uai_name = "uai-vers-1dcf28e5"
uai_status = "Running: Ready"
username = "vers"
```

4. Log in to the UAI with the connection string.

```
user@hostname> ssh <USERNAME@UAI_IP_ADDRESS>
```

6.5 Access a UAI from Multiple Hosts

Adds extra public ssh keys to a UAI to allow access from multiple hosts.

PREREQUISITES

- *Create a UAI from a Specific Image in Legacy Mode*

OBJECTIVE

When a UAI is first created, it can only be accessed with one SSH key. This procedure adds more public keys to a UAI to allow access from multiple hosts.

PROCEDURE

1. Identify the connection string for the UAI.

```
ncn-m001$ cray uas list | grep connect_string
'uai_connect_string": "ssh user@203.0.113.0"
```

2. Copy the new public key (`~/.ssh/<second-key.pub>`) to the UAI.

```
ncn-m001$ scp -i ~/.ssh/id_rsa ~/.ssh/<second-key.pub> <USERNAME@UAI_IP_ADDRESS>:<second-key.pub>
```

3. Add the new key to `/etc/uas/ssh/authorized_keys` in the UAI.

```
ncn-m001$ ssh <USERNAME@UAI_IP_ADDRESS> 'cat ~/<second-key.pub> >> etc/uas/ssh/authorized_keys'
```

4. SSH to the UAI with the new connection string.

```
ncn-m001$ ssh <USERNAME@UAI_IP_ADDRESS>
```

7 Configure the Development Environment with Modules

Each modulefile contains information needed to configure the shell for an application. After the Modules package is initialized, the environment can be modified on a per-module basis using the `module` command. Typically modulefiles instruct the `module` command to alter or set shell environment variables such as `$PATH`, `$MANPATH`, etc. Modulefiles can be shared by many users on a system, and users can create their own to supplement or replace the shared modulefiles.

Add or remove modulefiles from the current environment as needed. The environment changes contained in a modulefile can be summarized through the `module` command as well. If no arguments are given, a summary of the module usage and subcommands are shown. The action for the `module` command to take is described by the subcommand and its associated arguments.

TIP: Unless noted otherwise, the following commands work for both the default module system and Lmod.

Also, note that the modules and modulefiles listed are for demonstration purposes only. Actual versions may differ from those on the current system.

7.1 Get Started

After logging in, load the desired programming environment:

```
user@hostname> module load PrgEnv-<compiler>
```

7.2 List Loaded Modules

Note that module versions are for example purposes only and may vary from those on the system.

```
user@hostname> module list
Currently Loaded Modulefiles:
 1) cce/12.0
 2) cray-libsci/21.08.1.2
 3) cray-mpich/8.1.9
 4) PrgEnv-cray/8.3.1.0
 5) craype/2.7.6
 6) craype-x86-rome
 7) libfabric/1.11.0.0.233
 8) craype-network-ofi
 9) cray-dsmml/0.2.1
10) xpmem/2.2.35-7.0.1.0_1.3__gd50fabf.shasta
11) perftools-base/21.09.0
12) cray-mpich/8.1.9
```

7.3 List Available Programming Modules

```
user@hostname> module avail PrgEnv
--------------------------- /opt/cray/pe/modulefiles ---------------------------
PrgEnv-aocc/8.0.0(default)
PrgEnv-cray/8.0.0(default)
PrgEnv-gnu/8.0.0(default)
```

7.4 List Available Modules

```
user@hostname> module avail
```

List all available modules of a certain type (for example `module avail cce`):

```
user@hostname> module avail cce

--------------------------- /opt/cray/pe/modulefiles ---------------------------
cce/9.1.3
cce/10.0.0
cce/10.0.1 (default)
```

7.5 Load Modules

Load the default version of a module:

```
user@hostname> module load cce
```

Load a specific version of a module:

```
user@hostname> module load cce/<version>
```

7.6 Unload Modules

Remove a module:

```
user@hostname> module unload cray-libsci
```

7.7 Change Module Versions

Swap out the default module for a specific version:

```
user@hostname> module switch cce cce/<version>
```

7.8 Change Module Versions Using the `cpe` Module

The `cpe` module specifies all CPE modules associated with a given monthly CPE release. The module name is `cpe/<date>`, where `<date>` is the release date in the format `yy.mm`. The purpose of `cpe` is to enable users to switch currently loaded CPE modules to the version provided in a given monthly release by using a single command. All subsequently loaded modules treat the version associated with `cpe/<date>` as the default version.

For example, if `cray-mpich/8.1.3` and `cray-libsci/21.03.1.1` are included in cpe/21.03 (March 2021) and the user currently has `cray-mpich/8.1.2` and `cray-libsci/20.12.1.2` loaded, the following command switches both to the March 2021 versions:

```
user@hostname> module load cpe/21.03
```

TIP: Unloading `cpe` will NOT restore the previously loaded module versions and, in fact, will have no effect on the currently loaded modules. To compensate for this deficiency, `restore_system_default` scripts are provided in the `cpe` directory. Usage:

```
user@hostname> source /opt/cray/pe/cpe/21.03/restore_system_defaults.sh
```

7.9 Change Programming Environments

To change between Programming Environments, use module swap:

```
user@hostname> module swap PrgEnv-cray PrgEnv-gnu
```

7.10 Show Module Information

Display information about module conflicts and links:

```
user@hostname> module show perftools
-------------------------------------------------------------------
/opt/cray/pe/perftools/20.08.0/modulefiles/perftools:

prereq          perftools-base
conflict        perftools
conflict        perftools-lite
conflict        perftools-lite-sample
conflict        perftools-lite-events
conflict        perftools-lite-loops
conflict        perftools-lite-gpu
conflict        perftools-lite-hbm
conflict        perftools-preload
setenv          CRAYPAT_COMPILER_OPTIONS 1
setenv          CRAYPAT_SUBMODULE_VARS CRAYPAT_COMPILER_OPTIONS
```

```
-------------------------------------------------------------------

user@hostname> module show PrgEnv-cray
-------------------------------------------------------------------
/opt/cray/pe/modulefiles/PrgEnv-cray/8.0.0:

conflict         PrgEnv-amd
conflict         PrgEnv-aocc
conflict         PrgEnv-cray
conflict         PrgEnv-gnu
conflict         PrgEnv-intel
conflict         PrgEnv-nvidia
setenv           PE_ENV CRAY
setenv           cce_already_loaded 1
module           swap cce/11.0.4
module           switch cray-libsci cray-libsci/21.04.1.1
module           switch cray-mpich cray-mpich/8.1.4
module           load craype
module           load craype-x86-rome
module           load craype-network-ofi
module           load cray-dsmml
module           load perftools-base
module           load xpmem
module           load cray-mpich
module           load cray-libsci
setenv           CRAY_PRGENVCRAY loaded
```

7.11 Swap Other Programming Environment Components

For products that contain dynamically linked libraries such as MPI, switching the module environment does not completely change the run time environment, because the dynamic libraries are cached by the runtime linker as specified by `/etc/ld.so.conf`. To use a nondefault version of a dynamic library at run time, prepend <CRAY_LD_LIBRARY_PATH> to <LD_LIBRARY_PATH>.

The following commands revert the environment to an earlier version of `cray-mpich` 8.0:

```
user@hostname> module swap cray-mpich/8.0.5.4 cray-mpich/8.0.3

user@hostname> export LD_LIBRARY_PATH=$<CRAY_LD_LIBRARY_PATH>:$<LD_LIBRARY_PATH>
```

7.12 Lmod Mixed Compiler Support

CPE Lmod supports loading multiple different compiler modules concurrently by loading a dominant core-compiler module and one or more support mixed-compiler modules. Core-compiler modules are located in the core directory. Loading a core-compiler module sets Lmod hierarchy variables. After the core-compiler module is loaded, `module avail` lists the mixed-compiler modules available for loading.

CPE's Lmod mixed compiler support provides the flexibility for users to choose which compiler modules (including user generated compiler modules) to mix together; however, only compiler modules released by CPE are supported. Because users can generate their own compiler modules, CPE cannot guarantee that all mix-compilers shown for core-compiler modules are compatible.

Example:

Loading the CCE and GCC modules together with CCE as the dominant core compiler and GCC as the supporting mixed compiler (Note that command output text is abbreviated, and module versions are generalized.):

```
user@hostname> module load PrgEnv-cray
user@hostname> module avail
...
----- /opt/cray/pe/lmod/modulefiles/mix_compilers ----
   ...
    gcc-mixed/[version]
   ...
```

```
user@hostname> module load gcc-mixed
user@hostname> module list

Currently Loaded Modules:
  #)cce/[version] #)PrgEnv-cray/[version] #)gcc-mixed/[version]
```

8 Transfer Application Data

8.1 Copy Application Data from an External Workstation to a UAI

PREREQUISITES

- UAS is running
- The file that needs to be transferred from the system has been created

OBJECTIVE

Move application data from an external workstation, such as a laptop to a User Access Instance (UAI). Application data is stored on shared storage that is mounted onto the UAI on which the user is logged in.

PROCEDURE

1. Log in to the NCN.
2. Retrieve the UAI port number and IP address.

   ```
   user@ncn> cray uas list | grep connect_string

   `uai_connect_string = "ssh <USERNAME@UAI_IP_ADDRESS>
   ```

3. Transfer data from the workstation to the UAI with a password.

   ```
   user@uan> scp -i ~/.ssh/id_rsa <fileName.txt> <USERNAME@UAI_IP_ADDRESS>:
   ```

   The system returns a message after the file is completely transferred:

   ```
   fileName.txt              100%   30     1.9KB/s   00:00
   ```

4. Verify that the file transferred successfully.
   a. Connect to the UAI with the connection string.

      ```
      user@ncn> ssh <USERNAME@UAI_IP_ADDRESS>
      ```

   b. Switch to the home directory where the application data was saved on the Cray.
   c. List the files in the directory.

      ```
      [user@uai ~]$ ls -ltr <fileName.txt>
      -rw-r--r-- 1 user 1049 30 Sep  7 20:35 <fileName>
      ```

8.2 Copy Application Data from a UAI to an External Workstation

PREREQUISITES

- The user has a running User Access Instance (UAI).

OBJECTIVE

Moving application data from the User Access Instance (UAI) to an external workstation, such as a laptop. Application data is stored on shared storage that is mounted onto the UAI on which the user is logged in.

PROCEDURE

1. Log in to the UAI with the connection string.

   ```
   $ ssh <USERNAME@UAI_IP_ADDRESS>
   ```

2. Transfer data from the UAI to the workstation.

   ```
   [user@uai ~]$ scp <fileName.txt> <USERNAME>@<workstation-hostname>:<path>
   ```

   The system returns a message after the file is completely transferred:

   ```
   <fileName.txt>              100%   30     1.9KB/s   00:00
   ```

3. Verify the file was transferred successfully.
   a. Log on to the workstation.

b. Switch to the directory on the workstation where the application data was saved.

c. List the files in the directory.

```
$ ls -ltr <fileName.txt>
-rw-r--r-- 1 <USERNAME> 1049 30 Sep  7 20:35 <fileName.txt>
```

9 Compile an Application

9.1 Build An MPI Application

PREREQUISITES

- CPE must be loaded.

OBJECTIVE

Build an MPI application using CPE's compiler driver cc.

LIMITATIONS

- Only dynamic linking is supported.
- Vendor-specific compiler commands such as gcc are not supported.

PROCEDURE

1. Verify that the correct modules are loaded. Note that module versions are for example purposes only and may vary from those on the system.

   ```
   user@hostname> module list
   Currently Loaded Modulefiles:
   1) cce/13.0.1
   2) craype/2.7.15
   3) craype-x86-rome
   4) cray-libsci/21.08.1.2
   5) craype-network-ofi
   6) cray-dsmml/0.2.1
   7) perftools-base/22.04.0
   8) xpmem/2.2.40-7.0.1.0_2.4__g1d7a24d.shasta
   ```

2. Change to the directory where the application is located.

   ```
   user@hostname> cd /lus/<USERNAME>/
   ```

3. Create an application.

   See *Example MPI Program Source* for a sample "Hello World" MPI application.

4. Build the application.

   ```
   user@hostname> cc mpi_hello.c -o mpi_hello.x
   ```

9.2 Example MPI Program Source

`mpi_hello.c`

```
/* MPI hello world example */
#include <stdio.h>
#include <mpi.h>
int main(int argc, char **argv)
{
   int rank;
   MPI_Init(&argc, &argv);
   MPI_Comm_rank(MPI_COMM_WORLD, &rank);
   printf("Hello from rank %d\n", rank);
   MPI_Finalize();
   return 0;
}
```

9.3 Build an OpenSHMEM Application

PREREQUISITES

- A UAI is running (see *Create a UAI from a Specific Image in Legacy Mode*).

OBJECTIVE

Compile an OpenSHMEM program with OpenSHMEMX.

OpenSHMEMX is a proprietary OpenSHMEM specification implementation compliant with the OpenSHMEM API Specification Version 1.4. It is designed to be modular to support different transport layers for communication. OpenSHMEMX uses Libfabric for both inter- and intra-node communication. The library manpages are in-development and details on the base environment variables are documented in the `intro_shmem` manpage.

LIMITATIONS

- Only dynamic linking is currently supported.
- OpenSHMEMX is not supported on CentOS.
- Vendor-specific compiler commands such as gcc are not supported.

PROCEDURE

1. Log in to the UAI with the connection string.

   ```
   user@hostname> ssh <USERNAME@UAI_IP_ADDRESS>
   ```

2. Load CPE modules.

   ```
   [user@uai ~]$ module load cray-dsmml
   [user@uai ~]$ module load cray-openshmemx
   ```

3. Verify the correct modules are loaded. Note that module versions are for example purposes only and may vary from those on the machine.

   ```
   [user@uai ~]$ module list
   Currently Loaded Modulefiles:
     1) cce/12.0.3               7) libfabric/1.9.0a1
     2) cray-libsci/21.08.1.2    8) craype-network-slingshot10
     3) cray-mpich/8.1.9         9) cray-dsmml/0.2.1
     4) PrgEnv-cray/8.1.0       10) perftools-base/21.09.0
     5) craype/2.7.10           11) xpmem/2.2.30-7.0.1.2_7.9__g3f90e41.shasta
     6) craype-x86-rome         12) cray-openshmemx/11.3.3
   ```

4. Change to the directory where the application is located.

   ```
   [user@uai ~]$ cd /lus/<USERNAME>
   ```

5. Create an OpenSHMEM application.

   See *Example OpenSHMEM Program Source* for a sample Hello World OpenSHMEM application.

6. Build the application.

   ```
   [user@uai ~]$ cc -dynamic shmem_hello.c -o shmem_hello.x
   ```

9.4 Example OpenSHMEM Program Source

shmem_hello.c

```
/* OpenSHMEM hello world example */
#include <stdio.h>
#include <shmem.h>
int main(void)
{
    shmem_init();
    int me   = shmem_my_pe();
    int npes = shmem_n_pes();
    printf("Hello World from PE #%d of %d\n", me, npes);
    shmem_finalize();
    return 0;
```

```
}
```

9.5 Build a DSMML Application

PREREQUISITES

- A UAI is running (see *Create a UAI from a Specific Image in Legacy Mode*).

OBJECTIVE

Compile a DSMML program with Cray DSMML.

Distributed Symmetric Memory Management Library (DSMML) is a proprietary memory management library. DSMML is a standalone memory management library for maintaining distributed shared symmetric memory heaps for top-level PGAS languages and libraries like Coarray Fortran, UPC, and OpenSHMEM. DSMML allows user libraries to create multiple symmetric heaps and share information with other libraries. Through DSMML, interoperability can be extracted between PGAS programming models.

LIMITATIONS

- Only dynamic linking is currently supported.
- Cray DSMML is not supported on CentOS.
- Vendor-specific compiler commands such as gcc are not supported.

PROCEDURE

1. Log in to the UAI with the connection string.

   ```
   user@hostname> ssh <USERNAME@UAI_IP_ADDRESS> -i ~/.ssh/id_rsa
   ```

2. Load CPE modules.

   ```
   [user@uai ~]$ module load cray-dsmml
   ```

3. Verify the correct modules are loaded. Note that module versions are for example purposes only and may vary from those on the machine.

   ```
   [user@uai ~]$ module list
   Currently Loaded Modulefiles:
     1) cce/12.0.3            7) libfabric/1.9.0a1
     2) cray-libsci/21.08.1.2  8) craype-network-slingshot10
     3) cray-mpich/8.1.9       9) cray-dsmml/0.2.1
     4) PrgEnv-cray/8.1.0     10) perftools-base/21.09.0
     5) craype/2.7.10         11) xpmem/2.2.30-7.0.1.2_7.9__g3f90e41.shasta(default)
     6) craype-x86-rome
   ```

4. Change to directory where the application is located.

   ```
   [user@uai ~]$ cd /lus/<USERNAME>
   ```

5. Create an application.

   See *Example Cray DSMML Program Source* for a sample DSMML application.

6. Build the application.

   ```
   user@ ~]$ cc dsmml_example.c -o dsmml_example.x
   ```

9.6 Example Cray DSMML Program Source

dsmml_example.c

```
/* Example DSMML segment creation operation */
#include <stdio.h>
#include <dsmml.h>
int main(void)
{
   dsmml_init_info_t reg_attrs;
```

```
   dsmml_sheap_seg_info_t valid_seg;
   dsmml_sheap_seg_info_t *seg_info = NULL;

   req_attrs.mype     = 0;
   req_attrs.smp_mype = 0;
   req_attrs.smp_npes = 1;

   valid_seg.id        = 0;
   valid_seg.act_addr  = NULL;
   valid_seg.base_addr = (void *)0x30000000000;
   valid_seg.length    = 1024;
   valid_seg.type      = DSMML_MEM_SYS_DEFAULT;
   valid_seg.pagesize  = DSMML_HPSIZE_DEFAULT;
   valid_seg.mode      = DSMML_MODE_DEFAULT;
   valid_seg.smp_mype  = 0;
   valid_seg.smp_npes  = 1;
   valid_seg.smp_set   = 0;

   dsmml_init(&req_attrs);
   dsmml_create_sheap_seg(&valid_seg);
   dsmml_get_sheap_seg(1, &seg_info);

   fprintf(stderr, "ID %d len %ld act_addr %p\n",
           seg_info->id, seg_info->length,
           seg_info->act_addr);

   dsmml_finalize();
   return 0;
}
```

10 Run an Application

10.1 Run an Application with Slurm in Batch Mode

PREREQUISITES

- Slurm is installed and configured on the system.
- The application is compiled (see *Build An MPI Application* or *Build an OpenSHMEM Application*).

OBJECTIVE

- Run an application (either MPI or OpenSHMEM) in batch mode with Slurm from a User Access Instance (UAI).
- For information on controlling network resources on Slingshot networks, see *Control Network Resources on Slingshot Networks with Slurm*.

PROCEDURE

1. Log in to the UAI with the connection string.

   ```
   user@hostname> ssh <USERNAME@UAI_IP_ADDRESS>
   ```

2. Load CPE modules.

   MPI: MPI modules are loaded by default.

   OpenSHMEM:

   ```
   [user@uai ~]$ module load cray-openshmemx
   ```

   Cray DSMML: Cray DSMML modules are loaded by default.

3. Change to the directory where the application is located.

   ```
   [user@uai ~]$ cd /lus/<USERNAME>/
   ```

4. Create a launch script.

   To launch the application with `sbatch`, add `srun` to the launch script.

   MPI: This example launch script is specific to the *Hello World MPI Application* example code. It runs on four nodes with one PE per node.

   ```
   [user@uai ~]$ cat launch.sh
   #!/bin/sh
   #SBATCH --time=5
   #SBATCH --nodes=4
   #SBATCH --tasks-per-node=1
   ulimit -s unlimited # in case not set by default
   srun ./mpi_hello.x
   ```

   OpenSHMEM: The example batch script below is specific to the *Hello World openSHMEM Application* example code. It runs on two nodes with four PEs per node.

   ```
   [user@uai ~]$ cat launch.sh
   #!/bin/sh
   #SBATCH --time=5
   #SBATCH --nodes=2
   #SBATCH --tasks-per-node=4
   module load cray-openshmemx
   srun -n8 ./shmem_hello.x
   ```

   Cray DSMML: The example batch script is specific to the *Example Cray DSMML Application* code. It runs on a single node with one PE.

   ```
   [user@uai ~]$ cat launch.sh
   #!/bin/sh
   #SBATCH --time=5
   #SBATCH --nodes=1
   #SBATCH --tasks-per-node=1
   ```

```
srun -n1 ./dsmml_example.x
```

5. Assign the required permissions to the `launch.sh` script to ensure it is executable.

```
[user@uai ~]$ chmod u+x launch.sh
```

6. Launch the batch script.

```
[user@uai ~]$ sbatch launch.sh
Submitted batch job 1065736
```

7. Check job output.

   MPI:

```
[user@uai ~]$ cat slurm-1065736.out
Hello from rank 1
Hello from rank 3
Hello from rank 0
Hello from rank 2
```

   Troubleshooting: Add `ldd` to the job script to check that the correct modules are loaded:

```
[user@uai ~]$ ldd ./mpi_hello.x
```

10.2 Run an Application with Slurm in Interactive Mode

PREREQUISITES

- Slurm is installed on the system.
- A User Access Instance (UAI) is running (see *Create a UAI from a Specific Image in Legacy Mode*).
- The application has been compiled (see *Build An MPI Application* or *Build an OpenSHMEM Application*).

OBJECTIVE

- Launch an application with `srun`.
- For information on controlling network resources on Slingshot networks, see *Control Network Resources on Slingshot Networks with Slurm*.

PROCEDURE

1. Log in to the UAI with the connection string.

```
user@hostname> ssh <USERNAME@UAI_IP_ADDRESS>
```

2. Load CPE modules.

   MPI: MPI modules are loaded by default.

   OpenSHMEM:

```
screen [user@uai ~]$ module load cray-openshmemx
```

3. Change to the directory where the application is located.

```
[user@uai ~]$ cd /lus/<USERNAME>
```

4. Execute the application with srun.

   MPI:

```
[user@uai ~]$ srun -N<ranks> --ntasks-per-node=<number_tasks_per_node> ./mpi_hello.x
```

   EXAMPLE: using the *Example MPI Program Source*

```
user@uai ~]$ srun -N4 --ntasks-per-node=1 ./mpi_hello.x
```

   EXAMPLE: using UCX instead of OFI

```
[user@uai ~]$ module swap craype-network-ofi craype-network-ucx
[user@uai ~]$ module swap cray-mpich cray-mpich-ucx
[user@uai ~]$ srun -N4 --ntasks-per-node=1 ./mpi_hello.x
```

OpenSHMEM:

```
[user@uai ~]$ srun -n<ranks> ./shmem_hello.x
```

EXAMPLE: using the *Example OpenSHMEM Program Source*

```
[user@uai ~]$ srun -n8 ./shmem_hello.x
```

10.3 Run an Application with PBS Pro in Batch Mode

PREREQUISITES

- PBS Pro workload manager is installed and configured
- The application is compiled (see *Build An MPI Application*)

OBJECTIVE

- Run an application with PBS Pro in batch mode.
- For information on controlling network resources on Slingshot networks, see *Control Network Resources on Slingshot Networks with PBS and PALS*.

PROCEDURE

1. Change to the directory where the application is located.

   ```
   user@hostname> cd /lus/<USERNAME>
   ```

2. Create a launch script `launch.sh`.

   MPI: This example launch script is specific to the "Hello World" MPI application (see *Example MPI Program Source*) running on four nodes.

   ```
   #!/bin/bash
   #PBS -l walltime=00:00:30
   echo start job $(date)
   module load cray-pals
   echo "mpiexec hostname"
   mpiexec hostname
   echo "mpiexec -n 4 /lus/<USERNAME>/hello_mpi"
   mpiexec -n4 /lus/<USERNAME>/mpi_hello.x
   echo end job $(date)
   exit 0
   ```

3. Assign the required permissions to the `launch.sh` script to verify it is executable.

   ```
   user@hostname> chmod u+x launch.sh
   ```

4. Launch the batch script.

   ```
   user@hostname> qsub -l select=4,place=scatter launch.sh
   ```

5. Check job output.

   ```
   user@hostname> cat launch.sh.o426757
   Hello from rank 3
   Hello from rank 2
   Hello from rank 1
   Hello from rank 0
   ```

10.4 Run an Application with PBS Pro in Interactive Mode

PREREQUISITES

- PBS Pro workload manager is installed and configured
- The application is compiled (see *Build An MPI Application*)

OBJECTIVE

- Run an application with PBS Pro in interactive mode.
- For information on controlling network resources on Slingshot networks, see *Control Network Resources on Slingshot Networks with PBS and PALS*.

PROCEDURE

1. Initiate an interactive session.

```
user@hostname> qsub -I
qsub: waiting for job 4071.pbs-host to start
qsub: job 4071.pbs-host ready
user@hostname>
```

2. Load the `PrgEnv-cray`, `cray-pals`, and `cray-pmi` modules.

```
user@hostname> module load PrgEnv-cray; module load cray-pals; module load cray-pmi
```

3. Get information about `mpiexec`.

```
user@hostname> type mpiexec
mpiexec is /opt/cray/pe/pals/<version>/bin/mpiexec
```

4. Change to the directory where the application is located.

```
user@hostname> cd /lus/<USERNAME>/
```

5. Run the executable MPI program.

```
user@hostname> mpiexec -n4 ./mpi_hello.x
Hello from rank 1
Hello from rank 2
Hello from rank 3
Hello from rank 0
```

EXAMPLE: using UCX instead of OFI

```
user@hostname> module swap craype-network-ofi craype-network-ucx

Inactive Modules:
  1) cray-mpich

user@hostname> module swap cray-mpich cray-mpich-ucx
user@hostname> mpiexec -n4 ./mpi_hello.x
Hello from rank 1
Hello from rank 2
Hello from rank 3
Hello from rank 0
```

11 Debug an Application

11.1 Debug a Hung Application Using gdb4hpc

PREREQUISITES

- Ensure that the targeted application is compiled with `-g` to display debugging symbols. Compiling with `-O0` to disable compiler optimizations is advised but not required.
- Ensure that the targeted application is launched via `srun`.

OBJECTIVE

Debug a hung application using gdb4hpc.

PROCEDURE

1. Load the gdb4hpc module.

   ```
   $ module load gdb4hpc
   ```

2. Launch gdb4hpc.

   ```
   $ gdb4hpc
   ```

3. Attach onto the application with gdb4hpc.

   a. Choose a process set handle. All process sets are represented as a named debugger variable. Debugger variables are prefixed with a $ in the form of $`<name>`. For example, $`app` or $`a`. This can be any variable name of your choosing, whatever is easiest to remember.

   b. Determine the application identifier. For Slurm applications launched with `srun` this is a `jobid.stepid` determined using a mechanism such as `squeue` for the `jobid` and `sstat jobid` for the `stepid`. By default, jobs only have one `stepid` and start at 0. For applications run with `mpiexec`, the pid of the `mpiexec` process can be supplied.

   c. Using the information from steps a and b above, use the `attach` command to attach onto the running application. In the following example, $`a` is the process set handle and `1840118.0` is the application ID.

   ```
   $ dbg all> attach $a 1840118.0
   ```

4. Conduct a traditional parallel debugging session. gdb4hpc commands include:

   - `backtrace` – Displays stack frames. Pass in an argument to limit the number of stack frames displayed (such as `backtrace -5`). See `help backtrace`.
   - `frame` – Displays only the current stack frame.
   - `up` - Moves the current stack frame up. Pass in an argument to move up the specified number of stack frames (`up -3`).
   - `down` - Moves the current stack frame down. Pass in an argument to move down the specified number of stack frames (`down -5`).
   - `watchpoint` - Sets an access or write watchpoint.
   - `assign` - Assigns a debugger convenience variable or application variable (see `help assign` for more details).
   - `gdbmode` - Drops directly into the gdb interpreter (see `help gdbmode` for more details).
   - `kill` - Kills the application with `SIGKILL` and ends the debug session.
   - `release` - Detaches and resumes the application. Ends the debug session.
   - `quit` - Exits gdb4hpc and releases the application.

5. Fix any pinpointed bugs, recompile, and verify that the fix works.

TIP: While gdb4hpc is running, use the `help` command to get more information about command usage. For example, to find information about the `launch` command, enter:

```
$ help launch
```

11.2 Debug Hung Applications with STAT

PREREQUISITES

- The STAT module must be loaded. To load the system default version:

  ```
  module load cray-stat
  ```

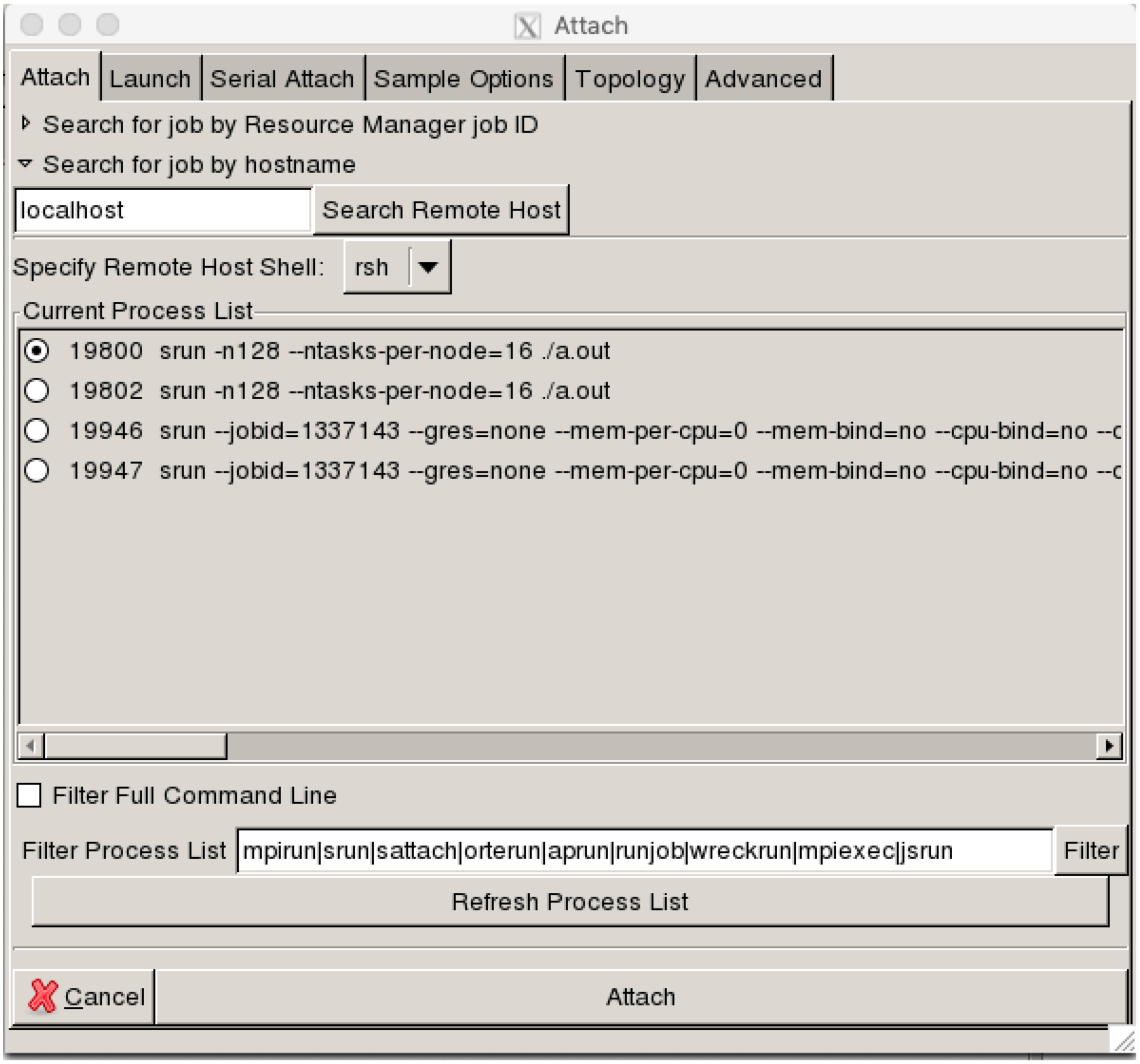
OBJECTIVE

Debug a hung application using the Cray Stack Trace Analysis Tool (STAT).

PROCEDURE

1. Attach onto the hung application with STAT using the workload manager job launcher, using either `stat-gui` or `stat-cl`. In the example, 19800 is the pid of the `srun` process.

   ```
   $ stat-cl 19800
   ```



localhost

Specify Remote Host Shell: rsh

```
(●) 19800  srun -n128 --ntasks-per-node=16 ./a.out
( ) 19802  srun -n128 --ntasks-per-node=16 ./a.out
( ) 19946  srun --jobid=1337143 --gres=none --mem-per-cpu=0 --mem-bind=no --cpu-bind=no --c
( ) 19947  srun --jobid=1337143 --gres=none --mem-per-cpu=0 --mem-bind=no --cpu-bind=no --c
```

[ ] Filter Full Command Line

Filter Process List: mpirun|srun|sattach|orterun|aprun|runjob|wreckrun|mpiexec|jsrun

Figure 1: Attach A Hung Application In STAT

   STAT launches its daemons and gathers a stack trace from each process and merges them into a prefix tree.

2. Analyze the merged backtrace using `stat-view` or `stat-gui`.

3. Choose additional debugging steps based on the nature of the hang. The following are available `stat-gui` tools:

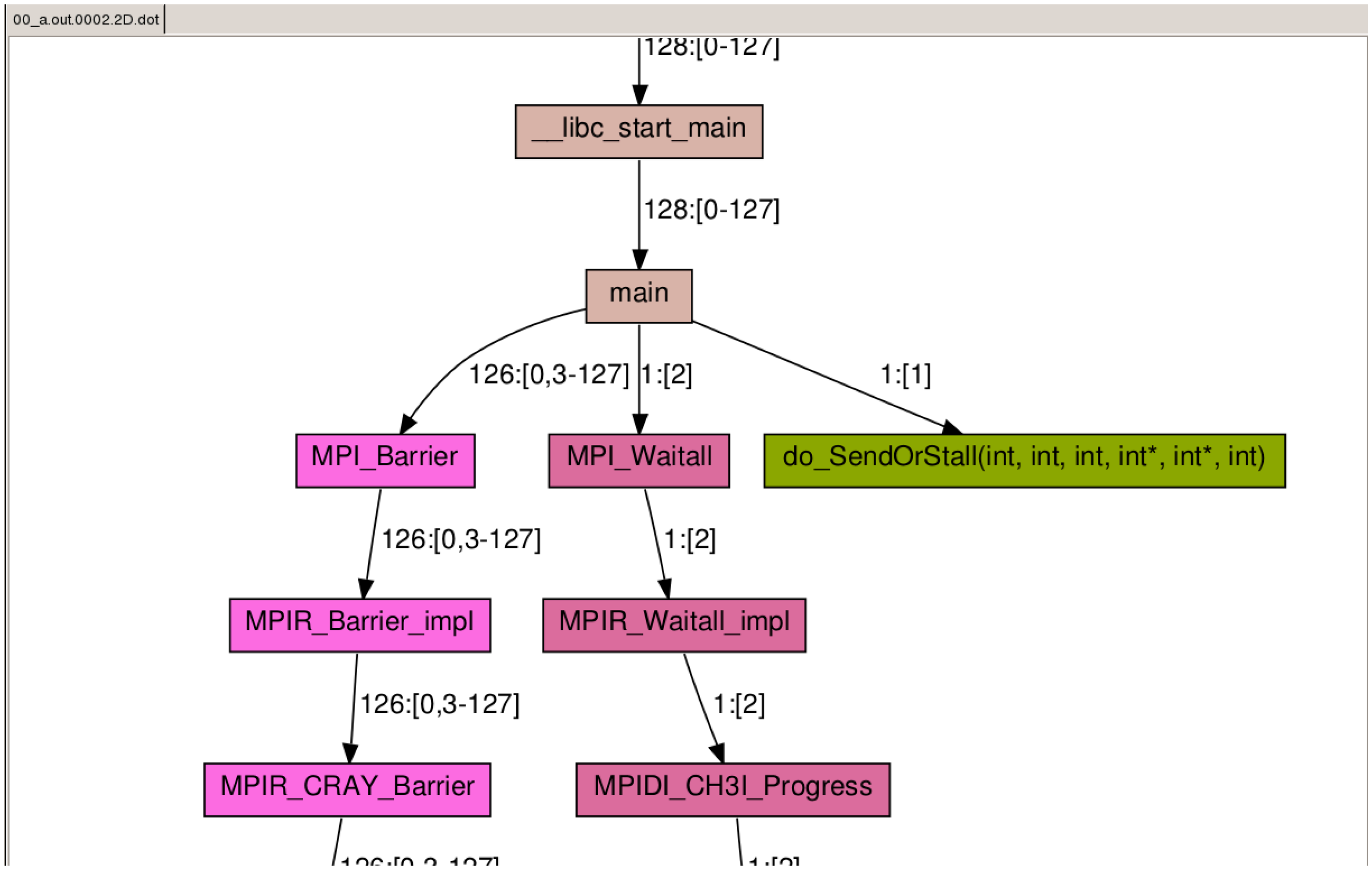


Figure 2: STAT Prefix Tree

- Narrow down to the trace steps exhibiting the bug by clicking **Shortest Path** (or **Longest Path**).
- Adjust sample size to look at the function level, or down to the function and line level, by clicking **Sample**.
- Identify stack traces visited by the least or most numbers of tasks to identify outliers by clicking **Least Task** or **Most Task**.- Step though the temporal order of the stack trace by clicking **Back TO** or **Forward TO**. Right-click on tasks that have made the least progress to **View Source** code.
- Gather X number of stack traces over time by clicking **Sample Multiple**.
- Choose a subset of equivalent classes to feed to a debugger by clicking **Eq C**.

4.Narrow down the search space to a specific function and use a traditional debugger like gbd4hpc or valgrind4hpc (depending on the bug's nature) to find the bug.

11.3 Debug Crashed Applications with ATP

PREQUISITES

- ATP module must be loaded (default).
- Target application is dynamically linked, or statically linked against the ATP support library.
- Target application is compiled with `-g` to keep debugging symbols.

OBJECTIVE

Debug a crashed application using the Cray Abnormal Termination Processing (ATP) debugger.

ATP is the first-line tool to diagnosis crashing applications.

When a parallel application crashes, ATP can produce a merged stack trace tree, providing an overview of the entire job state. ATP also selectively produces core files from crashing processes (or ranks). If further debugger support is required, the user may opt to rerun the job under the Cray parallel debugger, gdb4hpc.

PROCEDURE

1. Load the ATP module if not already loaded:

   ```
   $ module load atp
   ```

2. Set `ATP_ENABLED=1`.

3. (Optional) Set environmental variables.

   With the exception of `ATP_ENABLED`, ATP does not usually need other environment variables. But if necessary, runtime and output behavior can be modified with the following:

   - `ATP_CONSOLE_OUTPUT`: Default enabled. If enabled, ATP produces an overview of the crashed program and writes it to a standard error. This overview provides rank information, the signal that caused the crash, and crash location and assertion, if available.
   - `ATP_HOLD_TIME`: Default 0 minutes. If set to a nonzero value, ATP pauses for the specified number of minutes after detecting a crash. The job is held in a stopped state so a debugger like GDB4hpc can attach for further debugging.
   - `ATP_MAX_ANALYSIS_TIME`: Default 300 seconds. After sending a crash analysis request to the ATP backend process, the ATP frontend process waits the given number of seconds for crash analysis to occur. If this timeout expires, ATP assumes that the backend process was unsuccessful and continues job termination.
   - `ATP_MAX_CORES`: Default 20 core files. After crash analysis completion, ATP selects a subset of ranks from which to dump core files. The maximum number of such files is limited by this variable. If set to 0, core file dumping is disabled.
   - `ATP_CORE_FILE_DIRECTORY`: Default current directory. ATP writes core files from the selected subset of ranks to the given directory.

4. Run the application.

5. Examine the ATP output.

   When handling a crash, ATP prints the following message:

   ```
   Application is crashing. ATP analysis proceeding...
   ```

   It proceeds to list each process and the reason for its failure, if crashed.

   ```
   Processes died with the following statuses:
    <0 > Reason: '<RUNNING>' Address: 0x7ffff7bab697 Assertion: ''
   ```

```
 <1 2 3 > Reason: 'SIGSEGV /SEGV_MAPERR' Address: 0x0 Assertion: ''
```

In this example, process 0 did not crash and was still running. Processes 1, 2, and 3 experienced segfaults.

6. Gather the merged backtrace and core files.

   After displaying a summary of the job status, ATP writes selected core files and a graph visualization of the complete stack trace tree.

   ```
   Producing core dumps for ranks 3 1 2
    3 cores written in /cray/css/users/adangelo/stash/atp/tests
    View application merged backtrace tree with: stat-view atpMergedBT.dot
    You may need to: module load cray-stat
   ```

   By default, ATP writes the files `atpMergedBT.dot` and `atpMergedBT_line.dot` in the current working directory. `atpMergedBT.dot` is a function-level view of the stack trace tree, and `atpMergedBT_line.dot` is a source-line level view of the stack trace tree. Core files can be analyzed using the GNU Debugger, gdb.

7. Examine those files with `stat-view` or `gdb4hpc`.

   ```
   $ module load cray-stat
   $ stat-view atpMergedBT_line.dot
   ```

8. Fix any pinpointed bugs, recompile, and verify that the fixes work.

11.4 Debug Crashing Applications with gdb4hpc

PREREQUISITES

- The targeted application is compiled with `-g -O0` to keep debugging symbols and disable compiler optimizations.

OBJECTIVE

Debug a crashing application using gdb4hpc.

PROCEDURE

1. Load the gdb4hpc.

   ```
   $ module load gdb4hpc
   ```

2. Launch the application under gdb4hpc control.

   ```
   $ launch $a{<number of inferiors/ranks>} [<optional_launch_args>] <path_to_executable>
   ```

3. After gdb4hpc launches `srun`:

   a. Determine WLM settings.

   b. Determine the process set handle. Choose an easy to remember/type name, such as $app or $a. Array syntax notation implies the number of processing elements/threads, equivalent to the Slurm `srun -n` option.

   c. Determine any additional arguments. See `help launch` for available launcher arguments.

   d. After launch is complete, initial entry point is displayed. Note the process set notation of the application `{0..1023}`, each representing a process element.

4. Conduct a traditional parallel debugging session. gdb4hpc commands include:

   - `backtrace` – Displays stack frames. Pass in an argument to limit the number of stack frames displayed (such as `backtrace -5`). See `help backtrace`.
   - `frame` – Displays only the current stack frame.
   - `up` - Moves up the current stack frame. Pass in an argument to move up the specified number of stack frames (`up -3`).
   - `down` - Moves the current stack frame down. Pass in an argument to move down the specified number of stack frames (`down -5`).
   - `watchpoint` - Sets an access or write watchpoint.
   - `assign` - Assigns a debugger convenience variable or application variable.
   - `gdbmode` - Drops directly into the gdb interpreter.
   - `kill` - Kills the application with `SIGKILL` and ends the debug session.
   - `release` - Detaches and resumes the application. Ends the debug session.

- `quit` - Exits gdb4hpc and releases the application.

5. Fix any pinpointed bugs, recompile, and verify that the fix works.

11.5 Debug Applications with valgrind4hpc to Find Common Errors

PREREQUISITES

- Ensure target application is dynamically linked.
- Ensure the targeted application is compiled with `-g` to keep debugging symbols.

OBJECTIVE

Find common issues like memory leaks using the valgrind4hpc debugger.

Valgrind4hpc is a Valgrind-based debugging tool which detects memory leaks and errors in parallel applications. Valgrind4hpc aggregates any duplicate messages across ranks to help provide an understandable picture of program behavior. Valgrind4hpc manages starting and redirecting output from many copies of Valgrind, as well as deduplicating and filtering Valgrind messages.

PROCEDURE

1. Load the valgrind4hpc module (if not already loaded):

   ```
   $ module load valgrind4phc
   ```

2. Run the `memcheck` tool to look for memory leaks.

   ```
   $ valgrind4hpc -n1024 --launcher-args="--exclusive -ntasks-per-node=32" \
   $PWD/build_cray/apps/transpose_matrix -- -c -M 31 -n 1000
   ```

   Use these common valgrind4hpc arguments:

   - `-n`, `--num-ranks=<n>` - Number of job ranks to pass to the workload manager (e.g., Slurm).
   - `-l`, `--launcher-args="<args>"` - Additional workload manager arguments, such as rank distribution settings.
   - `-o`, `--outputfile=<file>` - Redirects all Valgrind4hpc error output to file.
   - `-v`, `--valgrind-args="<args>"` - Arguments to pass to the Valgrind instance.
     - For example, `--valgrind-args="--track- origins=yes --leak-check=full"` will track the exact origin of every memory leak, at the cost of performance.

3. Examine the Valgrind4hpc output.

   Valgrind4hpc detects a potential memory error, such as an uninitialized read/write or a memory leak, and displays an error block containing the affected ranks, and a backtrace of where the error occurred. Note that with errors stemming from an invalid use of system library routines, the backtrace will mention internal library functions.

4. Fix any pinpointed bugs, recompile, and verify that the fixes worked.

11.6 Debug Applications with Sanitizers4hpc to Find Common Errors

PREREQUISITES

- Ensure target application is built with support for the desired LLVM or GPU.
- Ensure Sanitizer is compiled with `-g` to keep debugging symbols.

OBJECTIVE

Find memory access or leak issues at runtime using information from LLVM Sanitizers.

Sanitizers4hpc is an aggregation tool to collect and analyze LLVM Sanitizers output at scale. Currently, the AddressSanitizer, LeakSanitizer, and ThreadSanitizer tools are supported. Preliminary support for AMD's GPU Sanitizer library is also available. Sanitizers4hpc manages the launch of your job through the system's workload manager.

PROCEDURE

1. Start the tool by supplying any `Sanitizers4hpc` options, followed by a double dash `--` and your workload manager's job launch arguments. The target binary and its arguments are listed after the second double dash `--`.

   For example, to run the binary `a.out` with four ranks, run

```
$ sanitizers4hpc --launcher-args="-n4" -- ./a.out binary_argument
```

2. After encountering a memory error, the linked LLVM Sanitizer produces error reports for each affected rank. Sanitizers4hpc processes these error reports and aggregates them for easier analysis.

   - In the following example run, the application encounters an invalid read off the end of a buffer on all ranks. Sanitizers4hpc prints a single error report, noting that while it occurs on all four ranks, the AddressSanitizer library reports it at the same place in the source file.

     ```
     RANKS: <0-3>
     AddressSanitizer: heap-buffer-overflow on address
     READ of size 4 at 0x61d000002680 thread T0
     ...
     #1 0x328dc3 in main /source.c:37:22
     ...
     SUMMARY: AddressSanitizer: heap-buffer-overflow /source.c:52:15 in main
     ```

For more information on different supported Sanitizers, as well as running with GPU Sanitizers and troubleshooting job launch, load the `sanitizers4hpc` module and run `man sanitizers4hpc` to view the manual page.

11.7 Debug Two Versions of the Same Application Side-By-Side with CCDB

PREREQUISITES

- There must be two versions of the same application to compare.

OBJECTIVE

Debug an application by comparing it to a similar, yet different and working application, to see differences in data at every stage of execution.

LIMITATIONS

Applications must be similar enough to step through execution steps in parallel, but different enough to see data changes through those execution steps.

PROCEDURE

1. Load the CCDB module.

   ```
   $ module load cray-ccdb
   ```

2. Launch CCDB.

   ```
   $ ccdb
   ```

   The CCDB window pops up.

3. Fill out Launch Specifications for both applications.

   a. If resources have already been allocated and a qsub session started, enter: `Application`, `Launcher Args`, and `Number of PEs`.

   b. If resources have not been allocated, also enter a `Batch Type` in each `Launch Specification`.

4. Double click the source file for each test application to view the source code.

5. Generate an Assertion Script.

   a. Left click on any line number.

   b. Click **Build Assert**.

      The Assertion Script Dialog window opens.

   c. Enter the following information for both `App0` and `App1`: `Name of the Source File`, `Line number`, `Variable`, and `Decomposition`.

   d. Click the **Add Assert** button.

   e. Select **Save Script** to save the Assertion script.

   f. Select **Start** to run the assertion script.

6. Open an Assertion Script.

   a. From Menu bar, select **View**.

   b. Hover over **Assertion Scripts** in the **View** dropdown list.

   c. Select an Assertion Script.

   The CCDB Assertion Script dialog opens with the assertion loaded.

7. Alternately, use CCDB controls to step through the two applications to see where each breakdown or diverge.

8. Click the red **FAILURE** boxes to bring up failure details.

   TIP: For more information on using CCDB, click **?** in the current window or **Help** from the main CCDB window menu bar.

9. After narrowing down the search space, use a traditional debugger like gbd4hpc or valgrind4hpc (depending on the bug's nature) to find the bug. After fixing it, run the old and new versions side-by-side in CCDB to verify that the bug was fixed.

12 Profile an Application

12.1 Identify Application Bottlenecks

PREREQUISITES

- CPE must be installed.

OBJECTIVE

This procedure instruments applications, runs them, and creates detailed output highlighting application bottlenecks.

PROCEDURE

1. Load the `perftools-base` module if it is not already loaded.

   ```
   $ module load perftools-base
   ```

2. Load the `perftools-lite` instrumentation module.

   ```
   $ module load perftools-lite
   ```

3. Compile and link the program.

   ```
   $ make program
   ```

4. Run the program.

   ```
   $ srun a.out
   ```

   Upon completing execution, `perftools-lite` produces the following output:

   - A text report to `stdout`, profiling the program's behavior, identifying where the program spends its execution time, and offering recommendations for further analysis and possible optimizations.
   - An experiment data directory, containing files which can be used to examine the program's behavior more closely using Cray Apprentice2 or `pat_report`.
   - A report file, `data-directory/rpt-files/RUNTIME.rpt`, containing the same information written to `stdout`.

5. Review the profiling reports written to `stdout`. To get additional information without re-running, use the `pat_report` utility on the experiment directory (such as `my_app.mpi+68976-16s`) produced from a profiling run to generate new text reports. For example:

   ```
   $ pat_report -O calltree+src my_app.mpi+68976-16s
   ```

TIP: For additional help, run `pat_help` from the command line. Also, refer to the ***HPE Performance Analysis Tools User Guide (S-8014)***.

13 Manage the Environment Lifecycle

13.1 List UAS Information

OBJECTIVE

Use the `cray uas` command to gather information about the User Access Service's version, images, and running User Access Instances (UAIs). List descriptive information about the User Access Service.

13.1.1 List UAS Version with `cray uas mgr-info list`

```
ncn-m001# cray uas mgr-info list
service_name = "cray-uas-mgr",
version = "0.11.3"
```

13.1.2 List Available UAS Images with `cray uas images list`

```
ncn-m001# cray uas images list
default_image = "registry.local/cray/cray-uas-sles15sp1-slurm:latest"
image_list = [ "registry.local/cray/cray-uas-sles15sp1-slurm:latest",
"registry.local/cray/cray-uas-sles15sp1:latest",]
```

13.1.3 List UAI Information for Current User with `cray uas list`

```
ncn-m001# cray uas list
[[results]]
username = "user"
uai_host = "ncn-m001"
uai_status = "Running: Ready"
uai_connect_string = "ssh user@203.0.113.0 -i ~/.ssh/id_rsa"
uai_img = "registry.local/cray/cray-uas-sles15sp1-slurm:latest"
uai_age = "11m"
uai_name = "uai-user-be3a6770"
```

13.1.4 List UAIs on a Specific Host Node

```
ncn-m001# cray uas uais list --host ncn-m001
[[results]]
username = "user"
uai_host = "ncn-m001"
uai_status = "Running: Ready"
uai_connect_string = "ssh user@203.0.113.0 -i ~/.ssh/id_rsa"
uai_img = "registry.local/cray/cray-uas-sles15sp1-slurm:latest"
uai_age = "2h56m"
uai_name = "uai-user-f3b8eee0"
[[results]]
username = "user"
uai_host = "ncn-m001"
uai_status = "Running: Ready"
uai_connect_string = "ssh user@203.0.113.0 -i ~/.ssh/id_rsa"
uai_img = "registry.local/cray/cray-uas-sles15sp1-slurm:latest"
uai_age = "1d5h"
uai_name = "uai-user-f8671d33"
```

ncn-m ## Delete a UAI

PREREQUISITES

- A UAI is up and running.

OBJECTIVE

The `cray uas` command allows users to manage UAIs. This procedure deletes one of the user's UAIs. To delete all UAIs on the system, see ***List and Delete UAIs*** in the ***HPE Cray EX System Administration Guide S-8001***.

LIMITATIONS

- Currently, the user must SSH to the system as `root`.

PROCEDURE

1. Log in to an NCN as `root`.
2. List existing UAIs.

```
ncn-m001# cray uas list

username = "user"
uai_host = "ncn-m001"
uai_status = "Running: Ready"
uai_connect_string = "ssh user@203.0.113.0 -i ~/.ssh/id_rsa"
uai_img = "registry.local/cray/cray-uas-sles15sp1-slurm:latest"
uai_age = "0m"
uai_name = "uai-user-be3a6770"

username = "user"
uai_host = "ncn-s001"
uai_status = "Running: Ready"
uai_connect_string = "ssh user@203.0.113.0 -i ~/.ssh/id_rsa"
uai_img = "registry.local/cray/cray-uas-sles15sp1-slurm:latest"
uai_age = "11m"
uai_name = "uai-user-f488eef6"
```

3. Delete a UAI.

```
ncn-m001# cray uas delete --uai-list <UAI_NAME>
results = [ "Successfully deleted uai-user-be3a6770",]
```

When a UAI is deleted, WLM jobs are not cancelled or cleaned up.

14 Troubleshoot UAS Issues

This section provides examples of some commands that can be used to troubleshoot UAS related issues.

14.1 Troubleshoot Connection Issues

```
packet_write_wait: Connection to 203.0.113.0 port 30841: Broken pipe
```

If an error message related to broken pipes returns, enable keep-alives on the client side. The admin should update the `/etc/ssh/sshd_config` and `/etc/ssh/ssh_config` files to add the following:

```
TCPKeepAlive yes
ServerAliveInterval 120
ServerAliveCountMax 720
```

14.2 Invalid Credentials

```
ncn-m001 # cray auth login --username <USER> --password <WRONGPASSWORD>
Usage: cray auth login [OPTIONS]
Try "cray auth login --help" for help.

Error: Invalid Credentials
```

To resolve this issue:

- Log in to Keycloak and verify the user exists.
- Make sure the username and password are correct.

14.3 Retrieve UAS Logs

The system administrator can execute the following commands to retrieve UAS and the remote execution service logs:

```
ncn-m001# kubectl logs -n services -c cray-uas-mgr -l "app=cray-uas-mgr"
```

14.4 Ensure that Slurm is Running and Configured Correctly

Check if Slurm is running:

```
[user@uai ~] $ sinfo
```

The system returns a message similar to the following if Slurm is not running:

```
slurm_load_partitions: Unable to contact slurm controller (connect failure)
```

If this error is returned, it is likely that Slurm is not running. The system administrator can use the following commands to debug the issue:

```
ncn-m001# kubectl logs -n user -l app=slurmdb -c slurmdb  --tail=-1
ncn-m001# kubectl logs -n user -l app=slurmdbd -c slurmdbd  --tail=-1
ncn-m001# kubectl logs -n user -l app=slurmctld -c slurmctld  --tail=-1
```

14.5 Troubleshoot Default Images Issues when Using the CLI

If the image name provided while creating a new UAI is not registered for use by the system, the system returns an error message similar to the following:

```
ncn-m001# cray uas create --publickey ~/.ssh/id_rsa.pub  --imagename fred
Usage: cray uas create [OPTIONS]
Try "cray uas create --help" for help.

Error: Bad Request: Invalid image (fred). Valid images:
['dtr.dev.cray.com:443/cray/cray-uas-sles15sp1:latest'].
Default: dtr.dev.cray.com:443/cray/cray-uas-sles15sp1:latest
```

Retry creating the UAI using the list of images and the name of the default image provided in the error message.

14.6 Verify that the User Access Instances (UAIs) are Running

The system administrator can use the `kubectl` command to check the status of the UAI.

```
ncn-m001# kubectl get pod -n user -l uas=managed -o wide
NAME                  READY  STATUS             RESTART SAGE  IP      NODE   NOMINATED       READINESS
                                                                             NODE            GATES
uai-user-603-85d-zk6  0/1    ContainerCreating  0       109s  <none>  sms-2  <none>          <none>
uai-user-d7f-6db-7h5  0/1    ContainerCreating  0       116s  <none>  sms-2  <none>          <none>
uai-user-f6b-5dc-grb  0/1    ContainerCreating  0       113s  <none>  sms-2  <none>          <none>
```

If UAS pods are stuck in the `Pending` state, the admin needs to ensure the Kubernetes cluster has nodes available for running UAIs. Check that nodes are labeled with `uas=True` and are in the `Ready` state.

```
ncn-m001# kubectl get nodes -l uas
NAME        STATUS     ROLES    AGE   VERSION
ncn-m001     Ready    master    11d   v1.13.3
```

If none of the nodes are found or if the nodes listed are marked as `NotReady`, the UAI pods will not be scheduled and will not start.

14.7 Troubleshoot kubectl Certificate Issues

While `kubectl` is supported in a UAI, a `kubeconfig` file to access a Kubernetes cluster is not provided. To use `kubectl` to interface with a Kubernetes cluster, the user must supply their own `kubeconfig`.

```
[user@uai ~]# kubectl get nodes
The connection to the server localhost:8080 was refused - did you specify the right host or port?
```

For instructions to copy certificates into a UAI, see *Copy Application Data from an External Workstation to a UAI*.

Specify the location of the Kubernetes certificate with `KUBECONFIG`.

```
[user@uai ~]# KUBECONFIG=/tmp/<CONFIG> kubectl get nodes
NAME STATUS ROLES AGE VERSION
ncn-m001 Ready master 16d v1.13.3
ncn-m002 Ready master 16d v1.13.3
```

Users must specify `KUBECONFIG` with every `kubectl` command or specify the `kubeconfig` file location for the life of the UAI. To do this, either set the `KUBECONFIG` environment variable or set the `--kubeconfig` flag.

14.8 Troubleshoot X11 Issues

The system may return the following error if the user attempts to use an application that requires an X window (such as `xeyes`):

```
$ ssh <UAI_USERNAME@UAI_IP_ADDRESS>

   ______ ____   ___ __  __   __  __ ___     ____
  / ____// __ \ /   |\ \/ /  / / / //   |   /  _/
 / /    / /_/ // /| | \  /  / / / // /| |   / /
/ /___ / _, _// ___ | / /  / /_/ // ___ | _/ /
\____//_/ |_|/_/  |_|/_/   \____//_/  |_|/___/
[user@uai ~]$ xeyes
Error: Can't open display:
```

To resolve this issue, pass the `-X` option with the `ssh` command as show below:

```
$ ssh <UAI_USERNAME@UAI_IP_ADDRESS> -X

   ______ ____   ___ __  __   __  __ ___     ____
  / ____// __ \ /   |\ \/ /  / / / //   |   /  _/
 / /    / /_/ // /| | \  /  / / / // /| |   / /
/ /___ / _, _// ___ | / /  / /_/ // ___ | _/ /
\____//_/ |_|/_/  |_|/_/   \____//_/  |_|/___/
/usr/bin/xauth:  file /home/users/user/.Xauthority does not exist
[user@uai ~]$ echo $DISPLAY
203.0.113.0
```

The warning stating "Xauthority does not exist" will disappear with subsequent logins.

14.9 Troubleshoot SSH Host Key Issues

If strict host key checking enabled is enabled on the user's client, the below error may appear when connecting to a UAI over `ssh`.

```
WARNING: REMOTE HOST IDENTIFICATION HAS CHANGED
```

This can occur in a few circumstances, but is most likely to occur after the UAI container is restarted. If this occurs, remove the offending `ssh` hostkey from the local `known_hosts` file and try to connect again. The error message from `ssh` will contain the correct path to the `known_hosts` file and the line number of the problematic key.

14.10 Hard limits on UAI Creation

Each Kubernetes worker node has limits on how many pods it can run. Nodes are installed by default with a hard limit of 110 pods per node, but the number of pods may be further limited by memory and CPU utilization constraints. For a standard node the maximum number of UAIs per node is 110; if other pods are co-scheduled on the node, the number will be reduced.

Determine the hard limit on Kubernetes pods with `kubectrl describe node` and look for the `Capacity` section.

```
## kubectl describe node <NODE_NAME> -o yaml
...
capacity:
    cpu: "16"
    ephemeral-storage: 1921298528Ki
    hugepages-1Gi: "0"
    hugepages-2Mi: "0"
    memory: 181009640Ki
    pods: "110"
...
```

When UAIs are created, some UAIs might be left in the "Pending" state. The Kubernetes scheduler is unable to schedule them to a node, due to CPU, memory, or pod limit constraints. Use `kubectl describe pod` to check why the pod is pending. For example, this pod is "Pending" because the node has reached the hard limit of 110 pods.

```
## kubectl describe pod UAI-POD
Warning  Failed
Scheduling  21s (x20 over 4m31s)  default-scheduler  0/4 nodes are available:
1 Insufficient pods, 3 node(s) didn't match node selector.
```

15 Troubleshoot PBS Pro Issues

15.1 Check PBS Server Status

From a non-compute node (NCN):

```
ncn-m001# kubectl exec <PBS-SERVER-POD> -- ps -ef | grep pbs
root           1       0  0 Sep18 ?        00:00:00 /bin/bash -x ./pbs-init.sh server
root         848       1  0 Sep18 ?        00:00:00 /opt/pbs/sbin/pbs_comm
root         854       1  0 Sep18 ?        00:00:00 /opt/pbs/sbin/pbs_sched
root         992       1  0 Sep18 ?        00:00:05 /opt/pbs/sbin/pbs_ds_monitor monitor
pbsdata     1015       1  0 Sep18 ?        00:00:00 /opt/pbs/pgsql/bin/postgres -D
/var/spool/pbs/datastore -p 15007
pbsdata     1024    1015  0 Sep18 ?        00:00:00 postgres: logger process
pbsdata     1026    1015  0 Sep18 ?        00:00:00 postgres: checkpointer process
pbsdata     1027    1015  0 Sep18 ?        00:00:00 postgres: writer process
pbsdata     1028    1015  0 Sep18 ?        00:00:00 postgres: wal writer process
pbsdata     1029    1015  0 Sep18 ?        00:00:00 postgres: autovacuum launcher process
pbsdata     1030    1015  0 Sep18 ?        00:00:01 postgres: stats collector process
pbsdata     1035    1015  0 Sep18 ?        00:00:00 postgres: pbsdata pbs_datastore
10.2.200.1(35324) idle
root        1036       1  0 Sep18 ?        00:00:04 /opt/pbs/sbin/pbs_server.bin
```

From a compute node:

```
nid000001:~ # ps -ef | grep pbs
root     19163     1  0 06:51 ?        00:00:00 /opt/pbs/sbin/pbs_mom
root     21572 21500  0 08:28 pts/2    00:00:00 grep pbs

nid000001:~ # pbsnodes -a
nid000001
     Mom = nid000001.local
     ntype = PBS
     state = free
...
```

15.2 Check PBS Pro Version

From an NCN:

```
ncn-m001# kubectl exec -n user <PBS-SERVER-POD> -- /opt/pbs/sbin/pbs_server --version
pbs_version = 19.2.2.20190502140845
```

From a User Access Instance (UAI):

```
[user@uai ~]$ pbsnodes --version
pbs_version = 19.2.2.20190502140845
```

From a compute node:

```
nid000001# pbs_mom --version
pbs_version = 19.2.2.20190502140845
```

15.3 Retrieve PBS Logs

If there is a problem with the PBS server pod, run `kubectl describe pod -n user -lapp=pbs`.

Retrieve the logs from NCNs:

```
ncn-m001# kubectl logs -n user <PBS-SERVER-POD> cat /var/spool/pbs/sched_logs/<LOG_FILE>
```

```
ncn-m001# kubectl logs -n user <PBS-SERVER-POD> cat /var/spool/pbs/server_logs/<LOG_FILE>
```

View PBS MoM logs on compute nodes:

```
nid000001# view /var/spool/pbs/mom_logs/<LOG_FILE>
```

15.4 Return Nodes to `state=free`

From the NCN, get a list of nodes with pbsnodes and run the following as root:

```
ncn-m001# kubectl exec -n user <SERVER-POD-NAME> -- /opt/pbs/bin/qmgr \
-c "set node <NID_NAME> state=free"
```

15.5 Run the WLM Diagnostics Script

The WLM diagnostics script, `/opt/cray/tests/sms-smoke/user/cray-uas-mgr/wlm-diagnostics.sh`, can help determine several common issues. It must run with `/usr/bin/craytest` to use `check_pod_status` tool located in `/opt/cray/tests/sms-resources/bin`.

```
ncn-m001# /usr/bin/craytest /opt/cray/tests/sms-smoke/user/cray-uas-mgr/wlm-diagnostics.sh
```

Test output is located in `/tmp/cray/tests/<name_of_system>/sms-smoke/user/.summary`.

16 Troubleshoot Slurm Issues

Troubleshooting for systems using Slurm as the workload manager.

16.1 Check Slurm Status

Check `slurmctld` status:

```
[user@uai ~]$ scontrol ping
```

Check `slurmdbd` status:

```
[user@uai ~]$ sacct
```

Check compute node status:

```
[user@uai ~]$ sinfo
```

For more detail:

```
[user@uai ~]$ sinfo --list-reasons
```

16.2 Check Slurm Version

Check the Slurm version installed:

```
[user@uai ~]$ rpm -qi slurm
```

16.3 Retrieve Slurm Logs

From compute nodes:

```
nid000001# journalctl -u slurmd
```

Retrieve slurmctld logs:

```
ncn-m001# kubectl logs -n user --timestamps --tail=-1 -c slurmctld -lapp=slurmctld
```

Retrieve `slurmdbd` logs:

```
ncn-m001# kubectl logs -n user --timestamps --tail=-1 -c slurmdbd -lapp=slurmdbd
```

Retrieve accounting logs:

```
ncn-m001# kubectl logs -n user --timestamps --tail=-1 -lapp=slurmdb
```

16.4 Update Firmware to Resolve Slurm Issues

If Slurm is not able to use all nodes, it is likely due to an older version of firmware existing on the HPE Cray EX system.

Refer to the ***Firmware Update Service (FUS)*** section of the ***HPE Cray EX System Administration Guide S-8001*** for more information.