

Introduction to Athena - NAS' New CPU Cluster

Dec. 17, 2025

NASA Advanced Supercomputing Division

Outline

- Hardware Configuration
  - Slingshot 11 Interconnect with Dragonfly Topology
  - AMD Turin processors
  - Comparison with older NAS CPU clusters
- Software Environment
  - Cray Programming Environment
  - Licensed and other 3rd party software
- Porting Your Applications
  - Porting existing executables from older NAS CPU clusters
  - Building new executables on Athena
- Running Jobs
  - Altair PBS Professional 2024 without any NAS local modification

Deviation from setup of older NAS CPU clusters

Athena Hardware Configuration
- Modular Supercomputing Facility



Gray module – Athena; white module - Aitken

Front





Three of the four Athena racks

Back

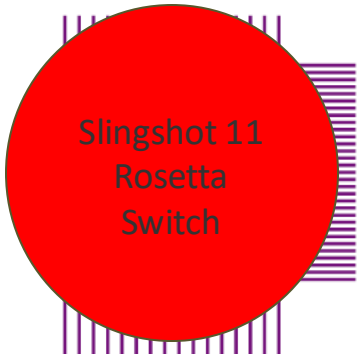


Athena Hardware Configuration - HPE Cray EX4000

- Compute Hardware
  - 4 compute racks x 8 **c**hassis x 8 blade **s**lots x 4 **n**odes = 1,024 AMD Turin compute nodes
  - Compute node hostname using Cray x-name: x100[0-3]**c**[0-7]**s**[0-7]b0**n**[0-3] (b0:node card 0)
  - 4 front-ends, hostname: athfe[01-04]
  - Connect to filesystems and hyperwall via routers
- HPE Cray Slingshot 11 Interconnect with Dragonfly topology
  - Cray optimized HPC Ethernet technology, not InfiniBand (as in Pleiades, Electra, Aitken)
  - High Radix Switches
    - 32 Rosetta Ethernet switches per rack (4 per chassis)
    - Each switch has 64 ports for high-degree connectivity
    - Each port can deliver 200 Gbps (25 GB/s) per direction (same bandwidth as IB 4x HDR used in Aitken)
  - Dragonfly Topology (not Hypercube as in Pleiades, Electra, Aitken)
    - Edge connection (L0) between a switch and a Cassini network interface card (NIC) in a node;
      Each node has two NICs connecting to two different switches, forming two Slingshot interfaces
    - **All-to-all Level 1 (L1)** connection between all switches in each rack
    - **All-to-all Level 2 (L2)** connection between racks (some switches used per rack)
    - **Max distance: 3 switch-to-switch hops regardless the number of racks in the cluster**
      (Milan: 512 nodes, 4 racks, 5D hypercube, 5 switch-to-switch hops max;
      Rome: 2048 nodes, 16 racks, 7D hypercube, 7 switch-to-switch hops max)

Athena Hardware Configuration
- Slingshot 11 Interconnect

16 ports per switch for L0
to 16 compute nodes



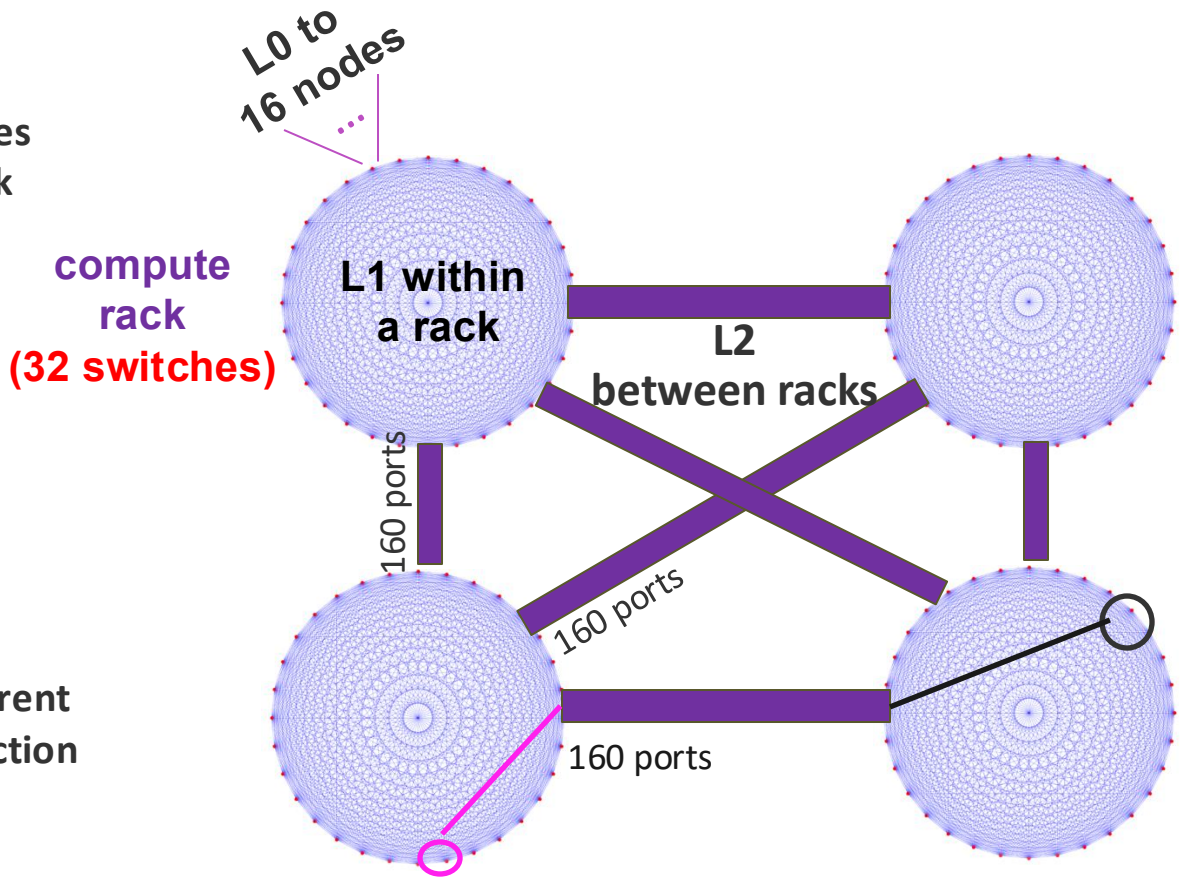
31 ports for L1
to other switches
in the same rack

16 ports for L2
to switches in other racks



L2 arc = A collection of ports from different switches in the same rack for L2 connection

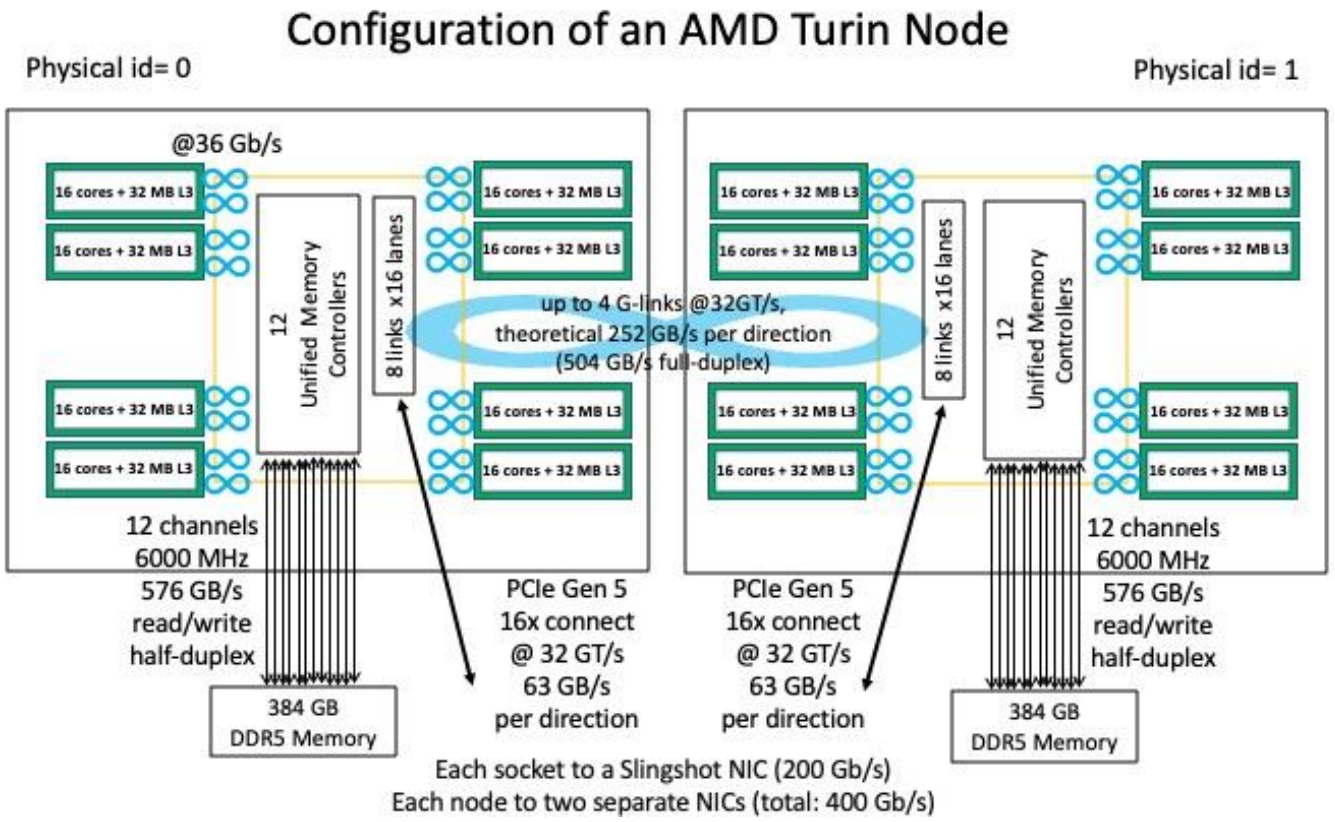
Total available ports for L2 per rack:
16 ports x 32 switches/rack
= 512 ports/rack



Maximum 3 hops

Athena Hardware Configuration
- Compute & Front-end Nodes

- Each node has two 128-core AMD EPYC 9745 processor sockets
  - Newer Zen microarchitecture: Turin (Zen 5c) vs Milan (Zen 3), Rome (Zen 2)
  - Higher density Core Complex: Turin (16 cores per CCX), Milan (8 cores per CCX), Rome (4 cores per CCX)
  - 2x # of cores/socket: Turin (128 cores) vs Milan or Rome (64 cores)
  - Newer Gen PCIe for inter-socket network and connecting to NICs: Turin (5th Gen), Milan/Rome (4th Gen)
  - Faster and larger memory: See next slide



Athena Hardware Configuration - Compute & Front-end Nodes

- Memory
  - 50% more memory per node: Turin (768GB, DDR5) vs Milan/Rome (512 GB, DDR4)
  - 25% less memory per core: Turin (3 GB/core) vs Milan/Rome (4 GB/core)
  - More memory channels, higher frequency, higher memory bandwidth per socket and per core:
    Turin: 12 channels, @6000 MHz, 576 GB/s per socket, 4.5 GB/s per core
    Milan/Rome: 8 channels, @3200 MHz, 204.8 GB/s per socket, 3.2 GB/s per core
  - NUMA nodes per socket: Turin compute nodes (NPS=2); Rome/Milan nodes (NPS=4)
- L1, L2, L3 cache
  - Larger L1 data cache: Turin (48 KB/core) vs Milan/Rome (32 KB/core)
  - Larger L2 cache: Turin (1 MB/core) vs Milan/Rome (512 KB/core)
  - Smaller L3 cache per core: Turin (32 MB for 16 cores), Milan (32 MB for 8 cores), Rome (16 MB for 4 cores)
- CPU core
  - Turin: AVX512 (512 bit data path); 2.4 GHz x 32 DP FLOPs/cycle x 256 cores = 19.66 TFLOPS/node
  - Milan: AVX2 (256 bit data path); 2.45 GHz x 16 DP FLOPs/cycle x 128 cores = 5.02 TFLOPS/node
  - Rome: AVX2 (256 bit data path): 2.25 GHz x 16 DP FLOPs/cycle x 128 cores = 4.61 TFLOPs/node
  - Peak FLOPs per cluster: Turin (1024 nodes, 20.1 PF), Milan (512 nodes, 2.6 PF); Rome (2048 nodes, 9.4 PF)
  - Hyperthreading setting: ON for Turin, Milan, Rome; check with *lscpu | grep "Thread(s) per core"*

Early NAS staff testing on Turin nodes shows a 30 – 50% faster performance (depending on the benchmarks) than Milan nodes on a core-to-core basis

Software Environment

- OS: RedHat Enterprise 9.6 (/etc/os-release) included in TOSS5 (/etc/toss-release)
- Tcl module system, in system startup script for Athena
  - DO *source /opt/cray/pe/modules/default/init/{csh,bash}* -> version 3.2.11.7
  - DON'T *source /usr/share/Modules/init/{csh,tcsh,bash,ksh,…}* or
    source /usr/local/lib/init/global.{cshrc,kshrc,login,profile} -> version 5.3.0
  - Suggestions from early users with slight modification
    - In .cshrc

```
if ( -e /opt/cray/pe/modules/default/init/csh ) then
   source /opt/cray/pe/modules/default/init/csh
else
   source /usr/local/lib/init/global.cshrc
endif
```

    - In .bashrc

```
if [ -f /opt/cray/pe/modules/default/init/bash ]; then
  source /opt/cray/pe/modules/default/init/bash
else
  source /usr/local/lib/global.profile
fi
```

Or

```
source /opt/cray/pe/modules/default/init/bash >& /dev/null || source /usr/local/lib/global.profile
```

Software Environment

- Default module paths
  - Multiple paths under /opt/cray provide access to Cray software packages (see next slide)
  - /nasa/modulefiles/toss5_crayex allows access to licensed or 3rd party packages or some tools built by NAS staff
    - Intel/2023.2.1 (ifort, icx), intel-classic/2023.2.1, intel-oneapi/2023.2.1, aocc/5.0.0
    - IDL, matlab, Tecplot, TotalView
    - Paraview, VisIt
    - SciCon tools: scicon/cli_tools (mbind.x, gm.x, nodes_info, ..), scicon/mpiprof
- Non-default module paths
  - /nasa/modulefiles/toss4 (e.g., comp-intel/xxx)
  - /swbuild/{username,project_name}
- /usr/bin
  - python (3.9), pip (21.3.1), parallel, podman (5.6.1), xclock

Software Environment
- Cray Programming Environment (CPE)

- Currently only CPE 25.09 installed
- CPE 25.09 modules under various /opt/cray paths

Automatically loaded at login

  - Meta-module: PrgEnv-cray, PrgEnv-gnu, PrgEnv-intel, PrgEnv-aocc (case sensitive)
  - Compilers: cce (Cray Compiler Environment), gcc-native, (intel, aocc created by craypkg-gen)
  - Compiler/link driver wrappers: craype includes wrappers **ftn, cc, CC**
  - MPI: cray-mpich (for building), cray-pals (parallel application launch service, runtime mpiexec), cray-mpixlate (for translation of C/C++ built with HPE MPT or OpenMPI)

    Note: **Cray MPICH is optimized for the Cray Slingshot Interconnect**

    cray-mpich/8.1.33 (default) implements MPI-3.1; cray-mpich/9.0.1 implements MPI 4.0.
  - Scientific libraries: cray-libsci (recommended over Intel MKL), cray-fftw
  - I/O: cray-hdf5 (1.14.3.7), cray-hdf5-parallel,

    cray-netcdf (4.9.2.1), cray-netcdf-hdf5parallel, cray-parallel-netcdf
  - Languages: cray-R (4.4.0), cray-python (3.11.7, with pip 23.2.1)
  - Debuggers: gdb4hpc, sanitizers4hpc, valgrind4hpc, atp (Abnormal Termination Processing), cray-stat (Stack Trace Analysis Tool), cray-ccdb (Cray Comparative Debugger)
  - Performance tools: papi, perftools-base, perftools (classic, link-time), perftools-lite, perftools-lite-events, perftools-lite-loops (simplified, compile-time), perftools-preload (easiest, run-time)

Porting Your Applications
- Existing x86_64 Executables

- Existing non-MPI executables built with Intel compiler
  - If it was built with Intel compiler flag –march=core-avx2 for Rome/Milan, it should run on Turin
  - If it was built with Intel compiler flag –xCORE-AVX512 for Skylake/Cascade Lake, it will not run on Turin and the following message is printed:

    Please verify that both the operating system and the processor support Intel(R) X87, CMOV, MMX, FXSAVE, SSE, SSE2, SSE3, SSSE3, SSE4_1, SSE4_2, MOVBE, POPCNT, AVX, F16C, FMA, BMI, LZCNT, AVX2, AVX512F, AVX512DQ, ADX, AVX512CD, AVX512BW and AVX512VL instructions.

- Existing executables built with Intel MKL
  - Intel MKL will detect whether an Intel processor is used and would not support the AVX512 instructions on Turin. Therefore, you don't get AVX512 performance using MKL on Turin.
  - It is recommended that you explore the use of Cray scientific library which supports AVX512
- Existing executables built with non-Cray-MPICH MPI
  - Executables built with Intel MPI may work since it is also based on MPICH.
  - Fortran executables built with HPE/SGI MPT or OpenMPI will NOT run on Athena. Rebuilding your code with Cray MPICH is required.
  - C/C++ built with HPE/SGI MPT or OpenMPI may be able to be translated to Cray MPICH using Cray's mpixlate tool. However, rebuilding with Cray MPICH is recommended.

module load cray-mpich cray-mpixlate cray-pals

mpiexec [mpiexec options] mpixlate –s sgi.225 –t cmpich.12 a.out

Porting Your Applications
- Building New Executables



- Demonstration: building a parallel dgemm MPI Fortran application (requires a BLAS library and an MPI library) on Athena.

Three approaches:

1. Explicitly invoking the Fortran compiler and link to library

   Note: Cray compilers are crayftn, craycc, crayCC; Intel: ifort, icc, icpc, ifx, icx, icpx

2. Using Cray MPICH's mpifort wrapper script

   This is a commonly used approach available from all MPIs (HPE/SGI MPT, MPICH, OpenMPI)

   Note: Cray MPICH mpif77, mpif90 are symlinks to mpifort

   Cray MPICH mpic++ is a symlink to mpicxx

3. Using Cray's ftn compile/link wrapper script

   This is a unique approach only seen with Cray Programming Environment.

   If you have used ftn, cc and CC in other data centers, you can continue to use them on Athena. Otherwise, use method 1 or 2 above.

- Compare the behavior: PrgEnv-cray vs PrgEnv-intel

  When we gain more experience with PrgEnv-aocc or PrgEnv-amd and AMD's scientific libraries, we will share more info with you.

Porting Your Applications

- Method 1: Invoking the compilers directly, not wrapper script

module load	module load PrgEnv-cray	module load PrgEnv-intel
module list	PrgEnv-cray/8.6.0	PrgEnv-intel/8.6.0
	cce/20.0.0 (crayftn, craycc, crayCC)	intel/2023.2.1 (ifort, icx, icpx)
	craype/2.7.35 (ftn, cc, CC)	craype/2.7.35 (ftn, cc, CC)
	cray-libsci/25.09.0	cray-libsci/25.09.0
	cray-mpich/8.1.33	cray-mpich/8.1.33
	a few others related to networks, perftools, etc.	a few others related to networks, perftools, etc.
build command	crayftn -mcmodel=medium par_dgemm.f -lmpifort -lmpi	ifort -mcmodel=medium par_dgemm.f -L${CRAY_LIBSCI_PREFIX}/lib -lsci_intel -lmpifort -lmpi
Error	The compiler cannot open INCLUDE file "mpif.h".	par_dgemm.f(2): error #5102: Cannot open include file 'mpif.h'
Rebuid	crayftn -mcmodel=medium par_dgemm.f -I${CRAY_MPICH_DIR}/include -L${CRAY_MPICH_DIR}/lib -lmpifort -lmpi	ifort -mcmodel=medium par_dgemm.f -L${CRAY_LIBSCI_PREFIX}/lib -lsci_intel -I${CRAY_MPICH_DIR}/include -L${CRAY_MPICH_DIR}/lib -lmpifort -lmpi
Error	none	none
ldd a.out	libsci_cray.so.6 => /opt/cray/pe/cce/20.0.0/cce/x86_64/bin/../../../../../libsci/25.09.0/CRAY/20.0/x86_64/lib/libsci_cray.so.6	libsci_intel.so.6 => /opt/cray/pe/lib64/libsci_intel.so.6
	libmpifort_cray.so.12 => /opt/cray/pe/lib64/libmpifort_cray.so.12	libmpifort_intel.so.12 => /opt/cray/pe/lib64/libmpifort_intel.so.12
	libfl.so.1 => /opt/cray/pe/cce/20.0.0/cce/x86_64/bin/../../../cce/x86_64/lib/libfl.so.1	libifport.so.5 => /nasa/intel/oneapi/compiler/2023.2.1/linux/compiler/lib/intel64/libifport.so.5
	/opt/cray/pe/cce/20.0.0/cce/x86_64/bin/../../../cce/x86_64/lib/libmodules.so.1	libifcoremt.so.5 => /nasa/intel/oneapi/compiler/2023.2.1/linux/compiler/lib/intel64/libifcoremt.so.5
	/opt/cray/pe/cce/20.0.0/cce/x86_64/bin/../../../cce/x86_64/lib/libcraymath.so.1	libimf.so => /nasa/intel/oneapi/compiler/2023.2.1/linux/compiler/lib/intel64/libimf.s
	libf.so.1 => /opt/cray/pe/cce/20.0.0/cce/x86_64/bin/../../../cce/x86_64/lib/libf.so.1	libsvml.so => /nasa/intel/oneapi/compiler/2023.2.1/linux/compiler/lib/intel64/libsvml.so
	libu.so.1 => /opt/cray/pe/cce/20.0.0/cce/x86_64/bin/../../../cce/x86_64/lib/libu.so.1	libintlc.so.5 => /nasa/intel/oneapi/compiler/2023.2.1/linux/compiler/lib/intel64/libintlc.so.5
	libcsup.so.1 => /opt/cray/pe/cce/20.0.0/cce/x86_64/bin/../../../cce/x86_64/lib/libcsup.so.1	libirng.so => /nasa/intel/oneapi/compiler/2023.2.1/linux/compiler/lib/intel64/libirng.so
		libifcore.so.5 => /nasa/intel/oneapi/compiler/2023.2.1/linux/compiler/lib/intel64/libifcore.so.5
	libmpi_cray.so.12 => /opt/cray/pe/lib64/libmpi_cray.so.12	libmpi_intel.so.12 => /opt/cray/pe/lib64/libmpi_intel.so.12
	libfabric.so.1 => /opt/cray/libfabric/12.0/lib64/libfabric.so.1	libfabric.so.1 => /opt/cray/libfabric/12.0/lib64/libfabric.so.1
	libpmi.so.0 => /opt/cray/pe/lib64/libpmi.so.0	libpmi.so.0 => /opt/cray/pe/lib64/libpmi.so.0
	libpmi2.so.0 => /opt/cray/pe/lib64/libpmi2.so.0	libpmi2.so.0 => /opt/cray/pe/lib64/libpmi2.so.0
	libpals.so.0 => /opt/cray/pals/1.7/lib/libpals.so.0	libpals.so.0 => /opt/cray/pals/1.7/lib/libpals.so.0

- crayftn finds libsci_cray automatically but not the MPI libraries. Intel's ifort finds none of these two.
- The cray-xxx modules and CPE integrated intel/2023.2.1 module do not define CPATH (for include) or LD_LIBRARY_PATH (for lib) like we see with Intel modules on Pleiades, Electra, Aitken (i.e, comp-intel/2023.2.1)
- For cray-mpich, **CRAY_MPICH_DIR** and **MPICH_DIR** are available and the same

Porting Your Applications

- Method 2:Building with CPE mpifort, mpicc, mpicxx wrappers

module load	module load PrgEnv-cray	module load PrgEnv-intel
module list	PrgEnv-cray/8.6.0	PrgEnv-intel/8.6.0
	cce/20.0.0 (crayftn, craycc, crayCC)	intel/2023.2.1 (ifort, icx, icpx)
	craype/2.7.35 (ftn, cc, CC)	craype/2.7.35 (ftn, cc, CC)
	cray-libsci/25.09.0	cray-libsci/25.09.0
	cray-mpich/8.1.33	cray-mpich/8.1.33
	a few others related to networks, perftools, etc.	a few others related to networks, perftools, etc.
build command	mpifort -mcmodel=medium par_dgemm.f	mpifort -mcmodel=medium par_dgemm.f
		-L${CRAY_LIBSCI_PREFIX}/lib -lsci_intel
ldd a.out	libsci_cray.so.6 => /opt/cray/pe/cce/20.0.0/cce/x86_64/bin/../../../../../libsci/25.09.0/CRAY/20.0/x86_64/lib/libsci_cray.so.6	libsci_intel.so.6 => /opt/cray/pe/lib64/libsci_intel.so.6
	libmpifort_cray.so.12 => /opt/cray/pe/lib64/libmpifort_cray.so.12	libmpifort_intel.so.12 => /opt/cray/pe/lib64/libmpifort_intel.so.12
	libfi.so.1 => /opt/cray/pe/cce/20.0.0/cce/x86_64/bin/../../../cce/x86_64/lib/libfi.so.1	libifport.so.5 => /nasa/intel/oneapi/compiler/2023.2.1/linux/compiler/lib/intel64/libifport.so.5
	libmodules.so.1 => /opt/cray/pe/cce/20.0.0/cce/x86_64/bin/../../../cce/x86_64/lib/libmodules.so.1	libifcoremt.so.5 => /nasa/intel/oneapi/compiler/2023.2.1/linux/compiler/lib/intel64/libifcoremt.so.5
	libcraymath.so.1 => /opt/cray/pe/cce/20.0.0/cce/x86_64/bin/../../../cce/x86_64/lib/libcraymath.so.1	libimf.so => /nasa/intel/oneapi/compiler/2023.2.1/linux/compiler/lib/intel64/libimf.s
	libf.so.1 => /opt/cray/pe/cce/20.0.0/cce/x86_64/bin/../../../cce/x86_64/lib/libf.so.1	libsvml.so => /nasa/intel/oneapi/compiler/2023.2.1/linux/compiler/lib/intel64/libsvml.so
	libu.so.1 => /opt/cray/pe/cce/20.0.0/cce/x86_64/bin/../../../cce/x86_64/lib/libu.so.1	libintlc.so.5 => /nasa/intel/oneapi/compiler/2023.2.1/linux/compiler/lib/intel64/libintlc.so.5
	libcsup.so.1 => /opt/cray/pe/cce/20.0.0/cce/x86_64/bin/../../../cce/x86_64/lib/libcsup.so.1	libirng.so => /nasa/intel/oneapi/compiler/2023.2.1/linux/compiler/lib/intel64/libirng.so
		libifcore.so.5 => /nasa/intel/oneapi/compiler/2023.2.1/linux/compiler/lib/intel64/libifcore.so.5
	libmpi_cray.so.12 => /opt/cray/pe/lib64/libmpi_cray.so.12	libmpi_intel.so.12 => /opt/cray/pe/lib64/libmpi_intel.so.12
	libfabric.so.1 => /opt/cray/libfabric/12.0/lib64/libfabric.so.1	libfabric.so.1 => /opt/cray/libfabric/12.0/lib64/libfabric.so.1
	libpmi.so.0 => /opt/cray/pe/lib64/libpmi.so.0	libpmi.so.0 => /opt/cray/pe/lib64/libpmi.so.0
	libpmi2.so.0 => /opt/cray/pe/lib64/libpmi2.so.0	libpmi2.so.0 => /opt/cray/pe/lib64/libpmi2.so.0
	libpals.so.0 => /opt/cray/pals/1.7/lib/libpals.so.0	libpals.so.0 => /opt/cray/pals/1.7/lib/libpals.so.0

- With PrgEnv-cray, both the Cray MPI library and libsci_cray are found without explicitly specifying –L and –l options.
- With PrgEnv-intel, mpifort finds the Cray MPI library, but one has to provide the –L and –lsci_intel to mpifort; **$CRAY_LIBSCI_PREFIX** which is same as **$CRAY_LIBSCI_PREFIX_DIR** points to /opt/cray/pe/libsci/25.09.0/**INTEL**/2025.0/x86_64

Porting Your Applications
- More about mpifort, mpicc, mpicxx with PrgEnv-intel

module load	module load PrgEnv-intel	module load PrgEnv-intel	module load PrgEnv-intel
module swap		module swap intel intel-classic	module swap intel intel-oneapi
module list	PrgEnv-intel/8.6.0	PrgEnv-intel	PrgEnv-intel/8.6.0
	intel/2023.2.1	intel-classic/2023.2.1	intel-oneapi/2023.2.1
	cray-mpich/8.1.33	cray-mpich/8.1.33	cray-mpich/8.1.33
ftn -v	ifort	ifort	ifx
cc -v	icx	icc	icx
CC -v	icpx	icpc	icpx
mpifort -v	ifx	ifx	ifx
mpicc -v	icx	icx	icx
mpicxx -v	icpx	icpx	icpx
In order for mpifort/mpicc/mpicxx to be consistent with ftn/cc/CC, need to set the following			
MPICH_FC	need explicitly set it to ifort	need explicitly set it to ifort	
MPICH_CC		need explicitly set it to icc	
MPICH_CXX		need explicitly set it to icpc	

- Intel oneAPI compilers are based on industry-standard LLVM infrastructure
- Intel 2025 compilers no longer provide Intel classic compilers
- Some Intel-classic compiler options and pragmas are not supported by oneAPI

Porting Your Applications
- Method 3: Building with CPE ftn, cc, CC wrappers

module load	module load PrgEnv-cray	module load PrgEnv-intel
module list	PrgEnv-cray/8.6.0	PrgEnv-intel/8.6.0
	cce/20.0.0 (crayftn, craycc, crayCC)	intel/2023.2.1 (ifort, icx, icpx)
	craype/2.7.35 (ftn, cc, CC)	craype/2.7.35 (ftn, cc, CC)
	cray-libsci/25.09.0	cray-libsci/25.09.0
	cray-mpich/8.1.33	cray-mpich/8.1.33
	a few others related to networks, perftools, etc.	a few others related to networks, perftools, etc.
build command	ftn -mcmodel=medium par_dgemm.f	ftn -mcmodel=medium par_dgemm.f
ldd a.out	libsci_cray.so.6 => /opt/cray/pe/lib64/libsci_cray.so.6	libsci_intel.so.6 => /opt/cray/pe/lib64/libsci_intel.so.6
	libmpifort_cray.so.12 => /opt/cray/pe/lib64/libmpifort_cray.so.12	libmpifort_intel.so.12 => /opt/cray/pe/lib64/libmpifort_intel.so.12
	libfi.so.1 => /opt/cray/pe/cce/20.0.0/cce/x86_64/bin/../../../cce/x86_64/lib/libfi.so.1	libifport.so.5 => /nasa/intel/oneapi/compiler/2023.2.1/linux/compiler/lib/intel64/libifport.so.5
	libmodules.so.1 => /opt/cray/pe/cce/20.0.0/cce/x86_64/bin/../../../cce/x86_64/lib/libmodules.so.1	libifcoremt.so.5 => /nasa/intel/oneapi/compiler/2023.2.1/linux/compiler/lib/intel64/libifcoremt.so.5
	libcraymath.so.1 => /opt/cray/pe/cce/20.0.0/cce/x86_64/bin/../../../cce/x86_64/lib/libcraymath.so.1	libimf.so => /nasa/intel/oneapi/compiler/2023.2.1/linux/compiler/lib/intel64/libimf.s
	libf.so.1 => /opt/cray/pe/cce/20.0.0/cce/x86_64/bin/../../../cce/x86_64/lib/libf.so.1	libsvml.so => /nasa/intel/oneapi/compiler/2023.2.1/linux/compiler/lib/intel64/libsvml.so
	libu.so.1 => /opt/cray/pe/cce/20.0.0/cce/x86_64/bin/../../../cce/x86_64/lib/libu.so.1	libintlc.so.5 => /nasa/intel/oneapi/compiler/2023.2.1/linux/compiler/lib/intel64/libintlc.so.5
	libcsup.so.1 => /opt/cray/pe/cce/20.0.0/cce/x86_64/bin/../../../cce/x86_64/lib/libcsup.so.1	libirng.so => /nasa/intel/oneapi/compiler/2023.2.1/linux/compiler/lib/intel64/libirng.so
		libifcore.so.5 => /nasa/intel/oneapi/compiler/2023.2.1/linux/compiler/lib/intel64/libifcore.so.5
	libmpi_cray.so.12 => /opt/cray/pe/lib64/libmpi_cray.so.12	libmpi_intel.so.12 => /opt/cray/pe/lib64/libmpi_intel.so.12
	libfabric.so.1 => /opt/cray/libfabric/12.0/lib64/libfabric.so.1	libfabric.so.1 => /opt/cray/libfabric/12.0/lib64/libfabric.so.1
	libpmi.so.0 => /opt/cray/pe/lib64/libpmi.so.0	libpmi.so.0 => /opt/cray/pe/lib64/libpmi.so.0
	libpmi2.so.0 => /opt/cray/pe/lib64/libpmi2.so.0	libpmi2.so.0 => /opt/cray/pe/lib64/libpmi2.so.0
	libpals.so.0 => /opt/cray/pals/1.7/lib/libpals.so.0	libpals.so.0 => /opt/cray/pals/1.7/lib/libpals.so.0

- You can add other crayftn (when using PrgEnv-cray) or ifort (PrgEnv-intel) compiler flags after ftn
- The ftn, cc, CC driver wrappers from craype simplify the command for compiling, and linking proper **libraries from Cray**, no need to add *–lmpi, lmpifort,* or *–lsci_cray*
- For libnetcdf, ftn finds it but not crayftn
- Loading PrgEnv-intel integrates use of intel compilers and linking **Cray libraries, NOT Intel MKL**

Porting Your Applications
- Building with CPE ftn, cc, CC wrappers + non-CPE-libraries
example: parallel dgemm MPI code with Intel MKL

module load	**module load PrgEnv-intel**
module list	**PrgEnv-intel/8.6.0**
	intel/2023.2.1 (ifort, icx, icpx)
	craype/2.7.35 (ftn, cc, CC)
	cray-libsci/25.09.0
	cray-mpich/8.1.33
	a few others related to networks, perftools, etc.
build command	**ftn -mcmodel=medium par_dgemm.f -qmkl=sequential**
ldd a.out	~~libsci_intel.so.6 => /opt/cray/pe/lib64/libsci_intel.so.6~~
	libmkl_intel_lp64.so.2 => not found
	libmkl_sequential.so.2 => not found
	libmkl_core.so.2 => not found
	libmpifort_intel.so.12 => /opt/cray/pe/lib64/libmpifort_intel.so.12
	libifport.so.5 => /nasa/intel/oneapi/compiler/2023.2.1/linux/compiler/lib/intel64/libifport.so.5
	libifcoremt.so.5 => /nasa/intel/oneapi/compiler/2023.2.1/linux/compiler/lib/intel64/libifcoremt.so.5
	libimf.so => /nasa/intel/oneapi/compiler/2023.2.1/linux/compiler/lib/intel64/libimf.s
	libsvml.so => /nasa/intel/oneapi/compiler/2023.2.1/linux/compiler/lib/intel64/libsvml.so
	libintlc.so.5 => /nasa/intel/oneapi/compiler/2023.2.1/linux/compiler/lib/intel64/libintlc.so.5
	libifcore.so.5 => /nasa/intel/oneapi/compiler/2023.2.1/linux/compiler/lib/intel64/libifcore.so.5
	libmpi_intel.so.12 => /opt/cray/pe/lib64/libmpi_intel.so.12
	libfabric.so.1 => /opt/cray/libfabric/12.0/lib64/libfabric.so.1
	libpmi.so.0 => /opt/cray/pe/lib64/libpmi.so.0
	libpmi2.so.0 => /opt/cray/pe/lib64/libpmi2.so.0
	libpals.so.0 => /opt/cray/pals/1.7/lib/libpals.so.0

During runtime, need to add to LD_LIBRARY_PATH
/nasa/intel/oneapi/mkl/2023.2.0/intel64

/nasa/intel/oneapi/mkl/2023.2.0/lib/intel64/libmkl_intel_lp64.so.2

/nasa/intel/oneapi/mkl/2023.2.0/lib/intel64/libmkl_sequential.so.2

/nasa/intel/oneapi/mkl/2023.2.0/lib/intel64/libmkl_core.so.2

- The module intel/2023.2.1 was created with the craypkg-gen tool in order for Intel compiler to be integrated with the Cray Programming Environment. It does not include the path to MKL library. Add the MKL path to LD_LIBRARY_PATH yourself when using intel/2023.2.1
- When building with ftn for applications that need other packages which are not part of the CPE, you need to add *–I/path-to-include-file –L/path-to-library –lxxx or* load a modulefile which has the include and library paths defined and link with *–lxxx.* This is true with any PrgEnv-xxx.

Optimize Code Targeting Turin Processors



- Cray
  - Load *craype-x86-turin* module (default on Athena)
    - craype wrappers ftn, cc, CC automatically turn on cce flag –hcpu=x86-turin (Fortran), -march=znver5 (C/C++); seen with ftn/cc/CC –craype-verbose output
  - crayftn, craycc, crayCC: need manually adding –hcpu=x86-turin (Fortran) or -march=znver5 (C/C++)
  - Other useful flags:
    - crayftn flag -hmsgs: writes optimization messages to stderr
    - crayftn -hlist=m or craycc –fsave-loopmark flag: produces file xxx.lst with source code loop marks indicating optimization
- Intel
  - Use of *craype-x86-turin* module does not turn on AVX512
    - craype wrappers turn on –march=core-avx2 –mtune=core-avx2 (seen with ftn –craype-verbose), same with *craype-x86-rome* ; without any craype-x86-xxx, default to –msse3
  - "AMD EPYC 9xx5-Series Processors Compiler Options Quick Reference": **–ax**CORE-AVX512 (for automatic cpu dispatch); if –axCORE-AVX512 does not perform, consider –qopt-zmm-usage=high (ifort, icc only) or –march=common-avx512 (icc, icx)
  - Other useful flags:
    - ifort/ifx/icc/icx flag -qopt-report[=n], n=0 to 5 (greatest details): generate optimization report
- Checking if a.out includes instructions with 512-bit zmm registers: *objdump –d a.out | grep zmm*

Warning: The presence of zmm instruction in a.out does not guarantee zmm is used during runtime

Running Jobs on Compute Nodes

- Initial queues, subject to change
  - Before production: testing_free
  - After production: devel, normal, long, wide (subject to approval)
  - Run the command *qstat –fQ queue_name* to find limits of the queue
  - SBU rate: Turin node 12.19; Milan node 4.38; Rome node 4.06
- *:model=tur_ath* in resource request required
- *:mem=xxx* is not needed. You get max ~750 GB/node via PBS
- *–lplace=scatter:excl* should be default; double-check is recommended
  - If a new queue is incorrectly set by NAS with *-lplace=free*, a job requesting *–lselect=2:ncpus=256* will pack 2 chunks of ncpus=256 into the same node (512 logical cores/node); if running with 512 MPI ranks, all of them will run in the same node instead of spreading between 2 nodes
- For MPI jobs, make sure that cray-pals module is loaded
  - When cray-pals is not loaded, you get the error

    "mpiexec: unexpected error - sgicheckppversion returned 0"
- PBS jobs should be submitted from athfe[01-04] to pbs06a

PBS Setting Deviation

- Default setting of mpiprocs in PBS jobs
  - On Pleiades, Electra, Aitken, if not set, it defaults to NCPUS; on Athena, it defaults to 1, and $PBS_NODEFILE has only 1 entry for each node.

    For example, with *–lselect=2:ncpus=4:model=tur_ath*
    - % cat $PBS_NODEFILE

      x1000c0s0b0n0.hsn.ath.nas.nasa.gov

      x1000c0s0b0n1.hsn.ath.nas.nasa.gov
    - *mpiexec a.out* will run with only 2 MPI ranks (one on each node), not 8 ranks
    - *mpiexec –np 8 --cpu-bind verbose a.out* -→ got round-robin placement

      Process 0 on x1000c0s0b0n0 out of 8

      Process 1 on x1000c0s0b0n1 out of 8

      Process 2 on x1000c0s0b0n0 out of 8

      Process 3 on x1000c0s0b0n1 out of 8

      ….

      Process 7 on x1000c0s0b0n1 out of 8
  - To get the first 4 ranks on the first node and last 4 ranks on the second node, do either:
    - setting mpiprocs in PBS resources explicitly, *–lselect=2:ncpus=4:**mpiprocs=4***
    - *mpiexec –np 8 –ppn 4 --cpu-bind verbose a.out*

PBS Setting Deviation

- Default setting of OMP_NUM_THREADS in PBS jobs
  - On Pleiades, Electra, Aitken, defaults to 1; On Athena, defaults to NCPUS
    - if the executable is linked to a multi-threaded library (e.g., –qmkl (default to parallel)), it will run multi-threads unexpectedly
    - A PBS job with –lselect=1:ncpus=256:mpiprocs=256:model=tur_ath

      mpiexec a.out -> will run 256 MPI ranks by 256 OpenMP threads
  - Suggestion: do either one of the following
    - export OMP_NUM_THREADS=1 (bash); setenv OMP_NUM_THREADS 1 (csh)
    - *-lselect=n:ncpus=x:**mpiprocs=y:ompthreads=1**:model=tur_ath*

Other Runtime Optimization

- MPI startup
  - Cray MPICH sets MPICH_OFI_USE_PROVIDER to cxi (Cassini interface) by default
  - To speedup MPI initialization process, you should also set FI_PROVIDER to cxi
- Binding processes/threads
  - Important when running sparse (use less than 256 cores in a Turin node) or MPI/OpenMP hybrid code
  - CRAY MPICH --bind-cpu has many ways to control binding [numa, socket, core, thread,..]
  - Recommend using NAS in-house tool mbind.x for binding
    - Load module scicon/cli_tools to access mbind.x
    - Add *--cpu-bind none* to eliminate interference with Cray's own binding scheme
    - For pure MPI code, set OMP_NUM_THREADS to 1

      mpiexec –np x –ppn y --cpu-bind none mbind.x –cs –v a.out
    - For MPI/OPENMP hybrid code, use one of the following two methods

      mpiexec –np x –ppn y –cpu-bind none mbind.x –cs –t n –v a.out

      or

      export OMP_NUM_THREADS=n (bash)

      setenv OMP_NUM_THREADS n (csh)

      mpiexec –np x –ppn y –cpu-bind none mbind.x –cs –v a.out

Recommendations

- Focus on using Intel compilers + Cray MPICH on Athena without ftn, cc, CC in initial porting to Athena. You can explore using crayftn, craycc, crayCC later
- Set MPICH_FC, MPICH_CC, MPICH_CXX to Intel classic compilers if you want mpifort, mpicc, and mpicxx to use ifort, icc, and icpc
- Use Cray scientific libraries (libsci_xxx) over MKL for AVX512 performance
- Explore the use of AMD aocc or amd Rocm compilers and AMD scientific libraries (AOCL); we will do the same and share our experiences in the future
- Pay attention to setting mpiprocs and OMP_NUM_THREADS in PBS jobs
- Set FI_PROVIDER to cxi to speed up MPI startup
- Use NAS in-house mbind.x tool for process/thread binding

Please share with us how your applications perform on Athena

Report issues via email to support@nas.nasa.gov

Acknowledgement

- NAS/ASRC and HPE Support Teams
- Early Users