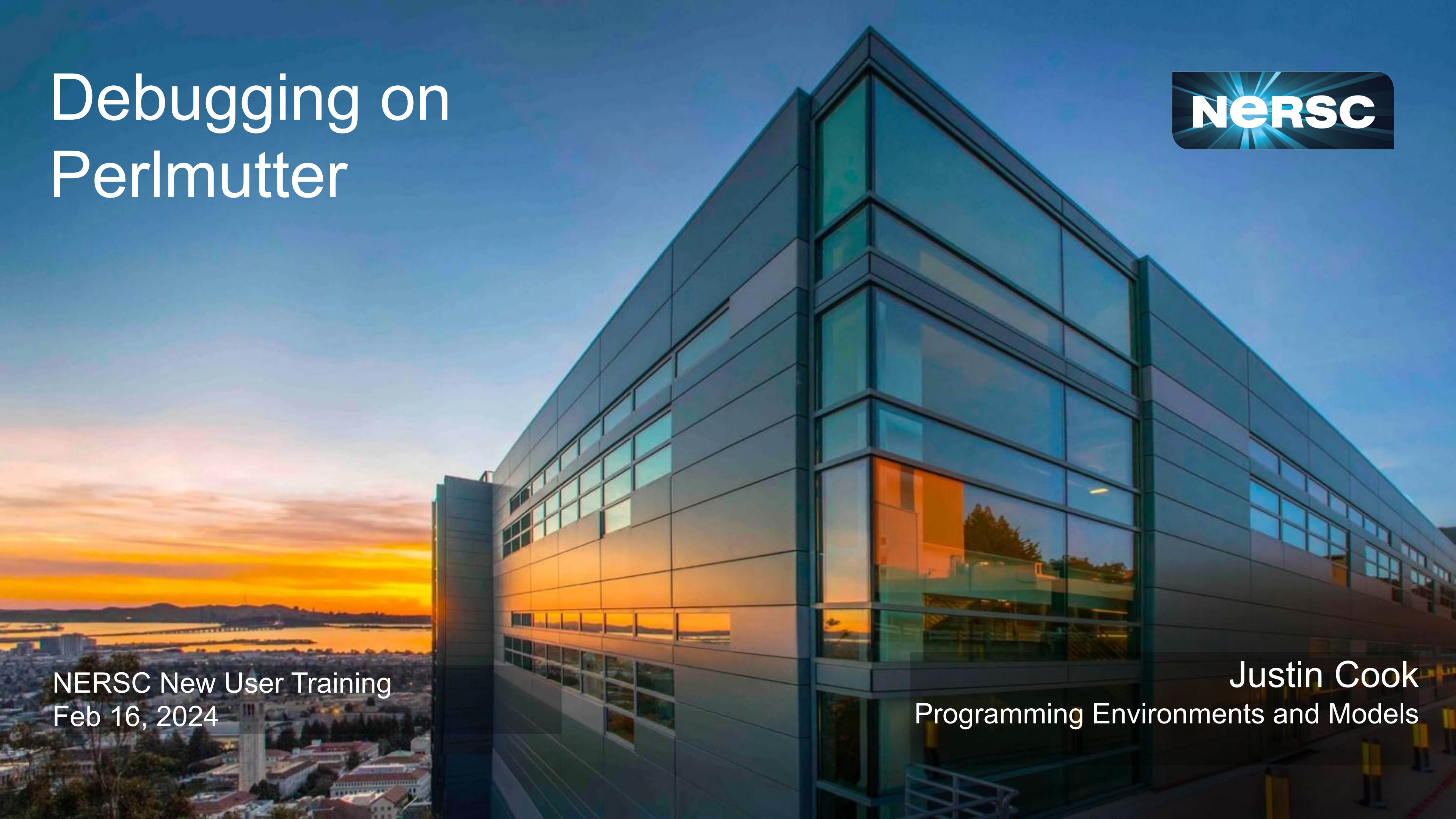


Debugging on Perlmutter

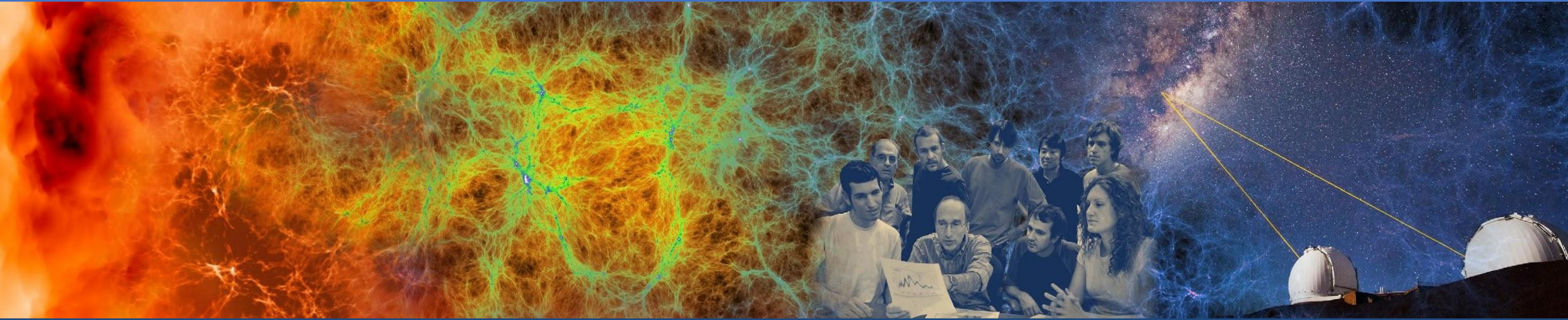
NERSC New User Training
Feb 16, 2024

Justin Cook
Programming Environments and Models

Debugging on Perlmutter

- Traditional parallel programming debuggers
  - DDT
  - TotalView
- Task-based debuggers with parallel programming extensions
  - atp + stat
  - gdb
  - valgrind
  - llvm sanitizers
- More options available:
  - https://docs.nersc.gov/tools/debug/

Best Practices



Setup a remote desktop connection

- Traditional x11 forwarding over ssh is slow
  - https://en.wikipedia.org/wiki/X_Window_System#Remote_desktop
- We recommend using NoMachine to improve the performance of this workflow
  - Download: https://www.nomachine.com/
  - Setup: https://docs.nersc.gov/connect/nx/
- Alternatively, the GUI debuggers have remote clients that can be used
- Check out sshproxy
  - https://docs.nersc.gov/connect/mfa/#sshproxy

Allow the creation of core files

- Core dump - Wikipedia
  - File containing the state of memory when a program crashed
  - Common input for some debuggers

```
$ ulimit -c unlimited
$ export MPICH_ABORT_ON_ERROR=1
$ export CUDA_ENABLE_COREDUMP_ON_EXCEPTION=1
```

Compile your program

- Generate debugging data and disable compiler optimizations
- C, Fortran
  - ‘g’ for adding debugging symbols
  - ‘O0’ (oh-zero) to disable optimizations
- CUDA
  - ‘G’ for device debugging
  - ‘cudart shared’ for memory debugging

```
$ cc -g -O0 -o program program.c
$ ftn -g -O0 -o program program.f90
$ nvcc -g -O0 -G -o program program.cu
```

Allocating nodes for debugging

- Quality of Service (QoS)
  - Interactive: high priority, 4 hours max, 4 nodes max
- Constraints:
  - CPU: Allocate only CPU nodes
  - GPU: Allocate only GPU nodes
- Account
  - Add '_g' to project name for GPU Nodes

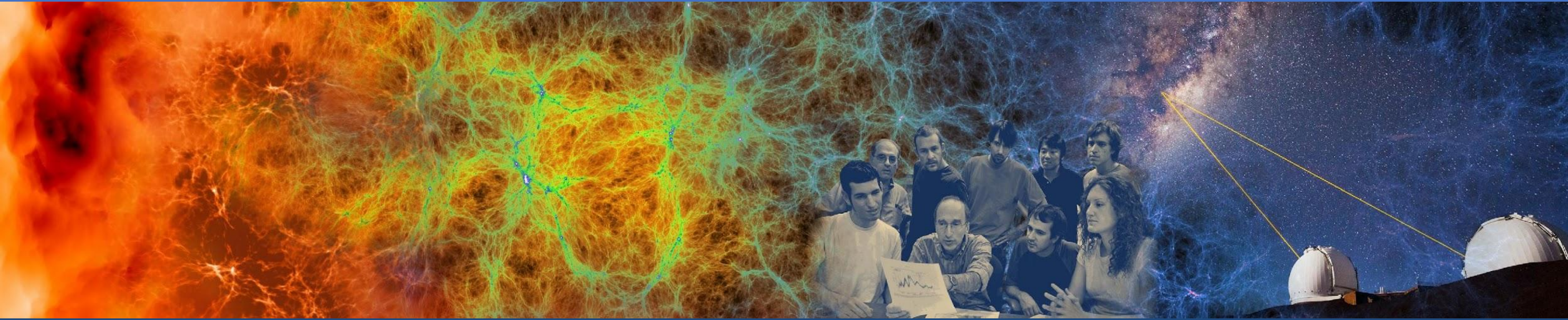
```
$ salloc --nodes=1 --qos=interactive --constraint=cpu --acount=mxxxx
$ salloc --nodes=1 --qos=interactive --constraint=gpu --acount=mxxxx_g --gpus=4
```

A note on HPE / Cray tools

- Make use of a Common Tools Interface (cti)
- https://cpe.ext.hpe.com/docs/debugging-tools/index.html#cti-common-tools-interface
- May require some additional setup

```
$ module load cray-cti
$ module load <cray product>
$ export CTI_WLM_IMPL=slurm
```

Getting started with DDT



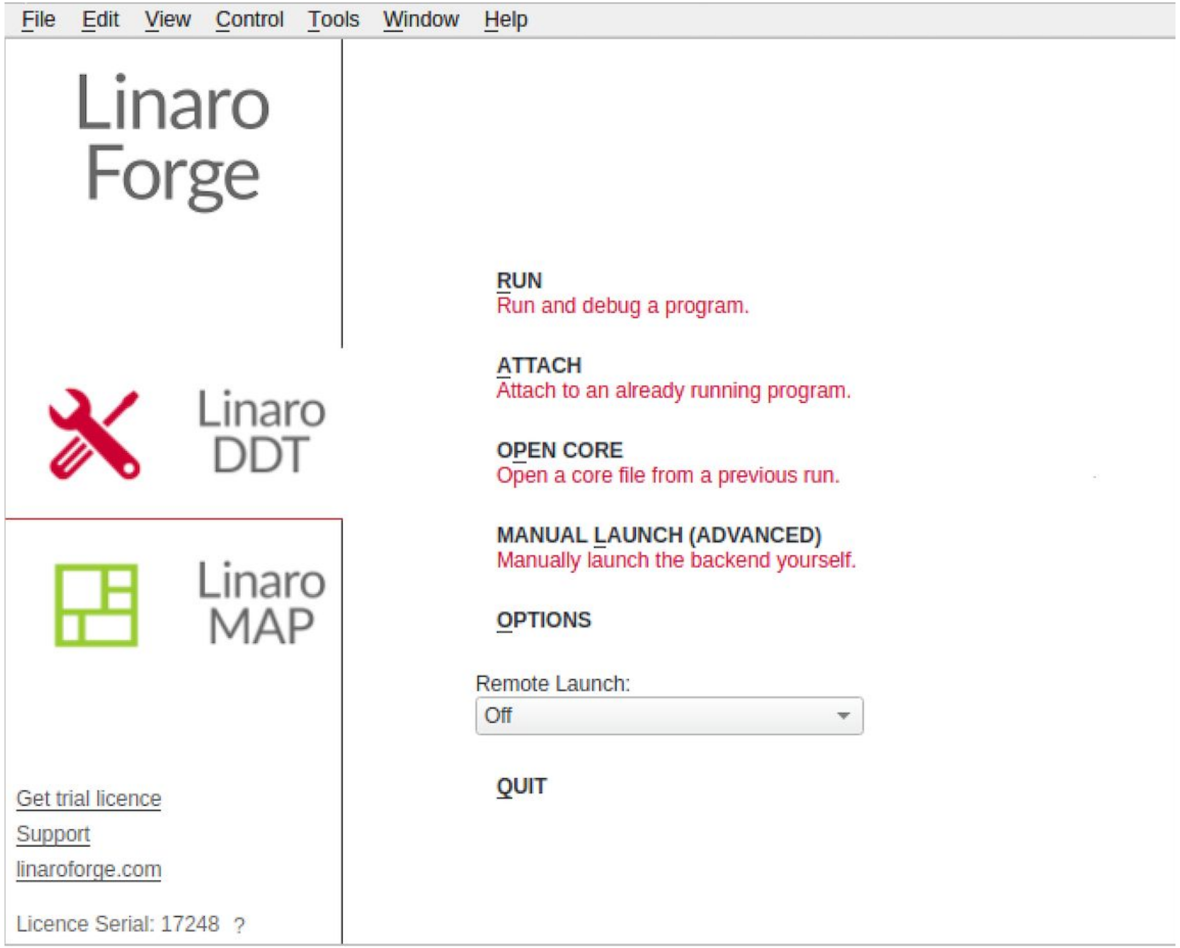
Debugging with DDT (Distributed Debugging Tool)

- Developed by Linaro
  - Linaro Forge Suite (https://www.linaroforge.com/)
- Supports core languages: C/C++, Fortran, Python
- Supports several parallel programming models:
  - MPI, OpenACC, OpenMP, CUDA, ROCM, CAF, UPC
- Documentation
  - Linaro: https://docs.linaroforge.com/latest/html/forge/index.html
  - NERSC: https://docs.nersc.gov/tools/debug/ddt

Usage

- Load the forge module
- Compile your program
- Allocate your compute nodes
- Run ddt
  - Run your program
  - Attach to an already running program

```
$ module load forge
$ ftn -g -O0 -o program program.f90
$ salloc -N1 -q interactive -C cpu [options]
$ ddt [./program]
```



RUN
Run and debug a program.

ATTACH
Attach to an already running program.

OPEN CORE
Open a core file from a previous run.

MANUAL LAUNCH (ADVANCED)
Manually launch the backend yourself.

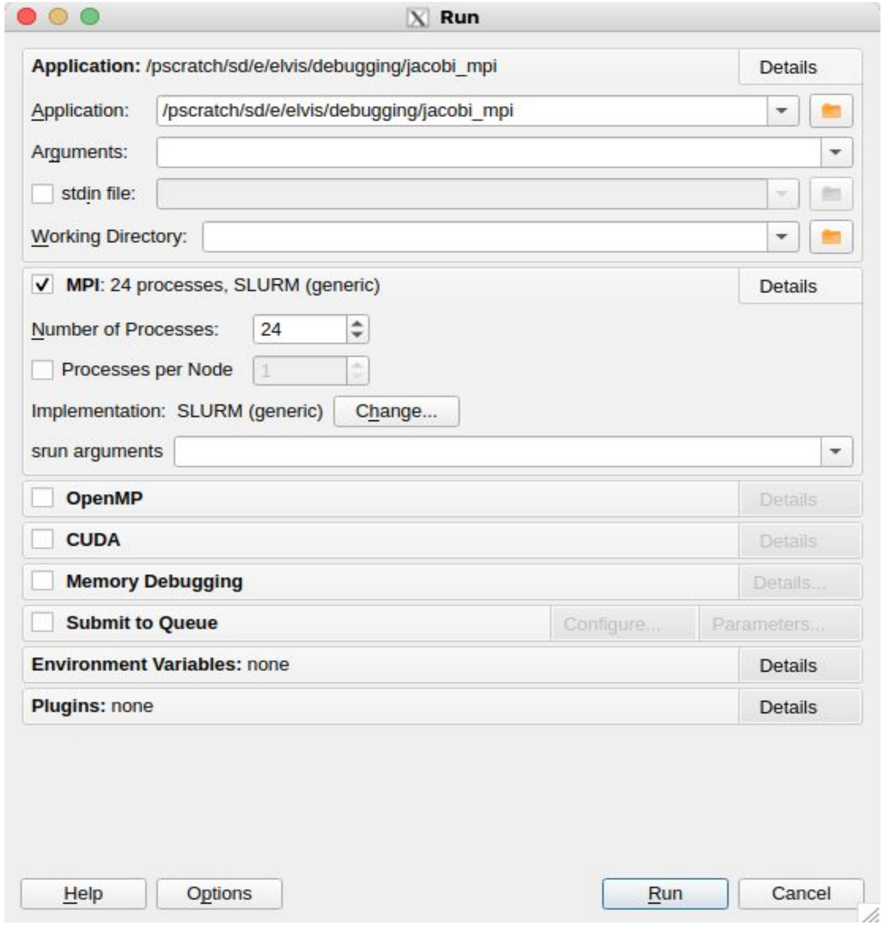
OPTIONS

Remote Launch: Off

QUIT

Get trial licence
Support
linaroforge.com

Licence Serial: 17248 ?



Run

Application: /pscratch/sd/e/elvis/debugging/jacobi_mpi

Details

Application: /pscratch/sd/e/elvis/debugging/jacobi_mpi

Arguments:

☐ stdin file:

Working Directory:

✓ **MPI**: 24 processes, SLURM (generic)

Details

Number of Processes: 24

☐ Processes per Node 1

Implementation: SLURM (generic) Change...

srun arguments

☐ **OpenMP** Details

☐ **CUDA** Details

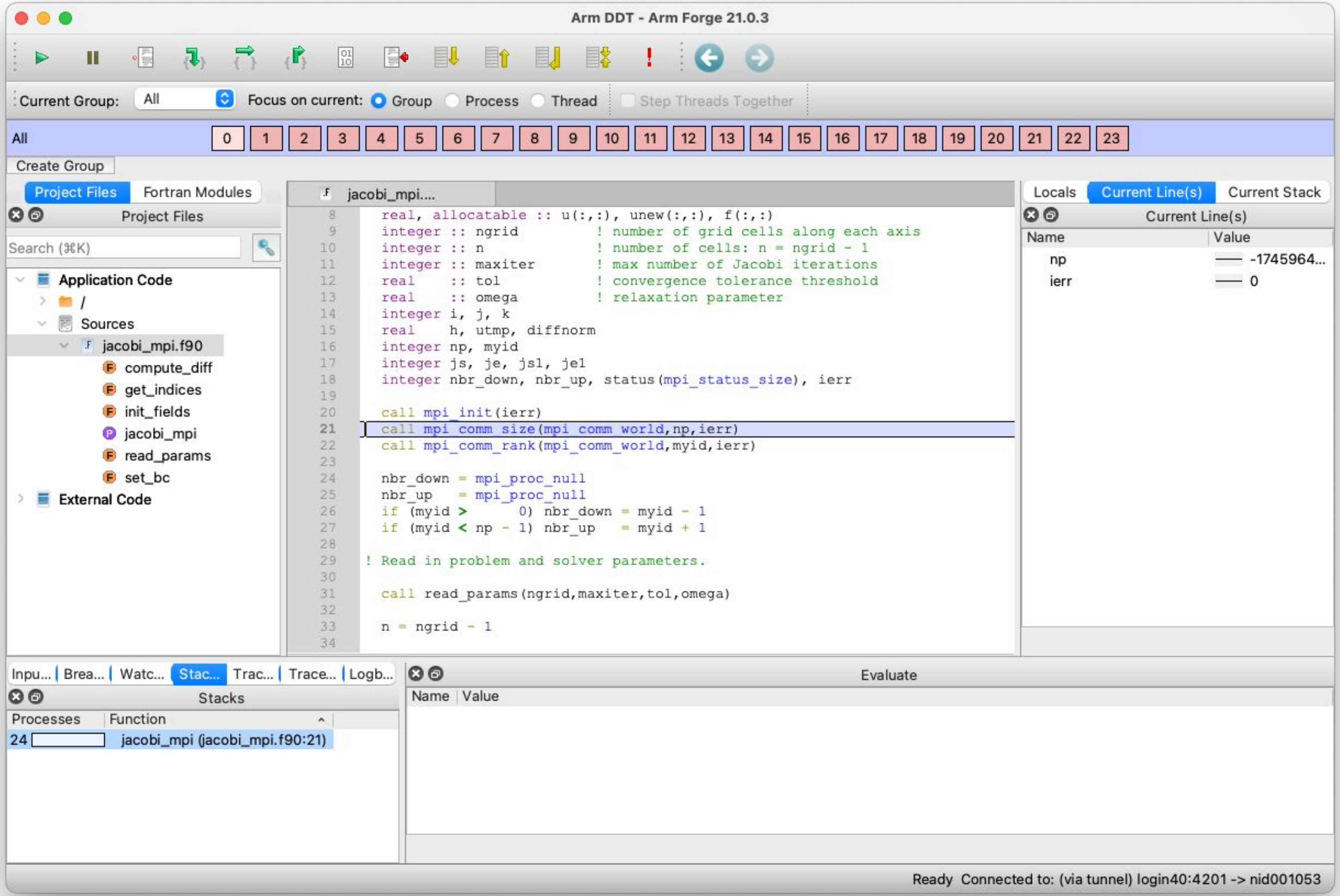
☐ **Memory Debugging** Details...

☐ **Submit to Queue** Configure... Parameters...

Environment Variables: none Details

Plugins: none Details

Help Options Run Cancel



```
  real, allocatable :: u(:,:), unew(:,:), f(:,:)
  integer :: ngrid         ! number of grid cells along each axis
  integer :: n             ! number of cells: n = ngrid - 1
  integer :: maxiter       ! max number of Jacobi iterations
  real    :: tol           ! convergence tolerance threshold
  real    :: omega         ! relaxation parameter
  integer i, j, k
  real    h, utmp, diffnorm
  integer np, myid
  integer js, je, jsl, jel
  integer nbr_down, nbr_up, status(mpi_status_size), ierr

  call mpi_init(ierr)
  call mpi_comm_size(mpi_comm_world,np,ierr)
  call mpi_comm_rank(mpi_comm_world,myid,ierr)

  nbr_down = mpi_proc_null
  nbr_up   = mpi_proc_null
  if (myid >      0) nbr_down = myid - 1
  if (myid < np - 1) nbr_up   = myid + 1

! Read in problem and solver parameters.

  call read_params(ngrid,maxiter,tol,omega)

  n = ngrid - 1
```

Current Line(s)

Name	Value
np	-1745964...
ierr	0

Stacks

Processes	Function
24	jacobi_mpi (jacobi_mpi.f90:21)

Evaluate

Name	Value

Ready Connected to: (via tunnel) login40:4201 -> nid001053

NERSC

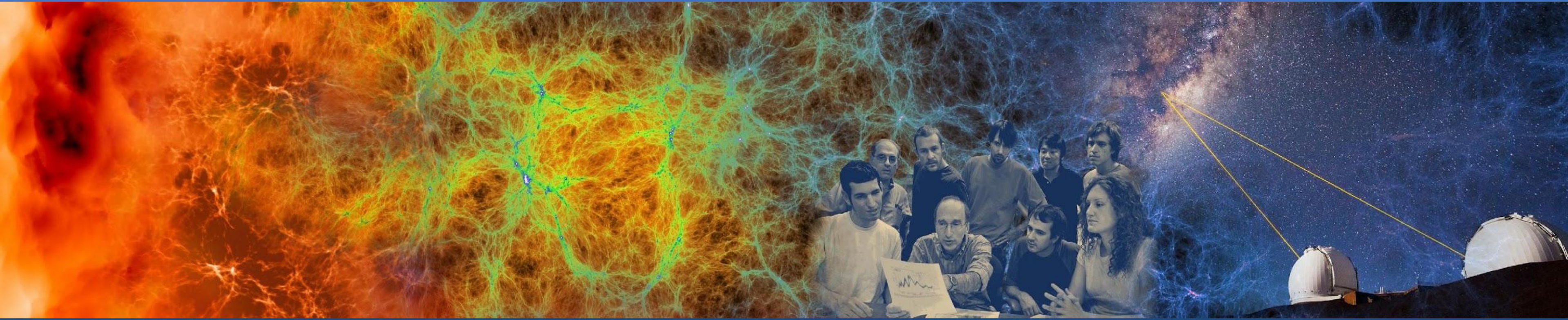
BERKELEY LAB Bringing Science Solutions to the World

U.S. DEPARTMENT OF ENERGY Office of Science

Reverse connection using the remote client

- Download the remote client
  - https://www.linaroforge.com/downloadForge/
- Configure your remote launcher settings
- Connect to Perlmutter
- Setup your reverse connection
  - https://docs.nersc.gov/tools/debug/ddt/#reverse-connect-using-remote-client

Getting started with TotalView



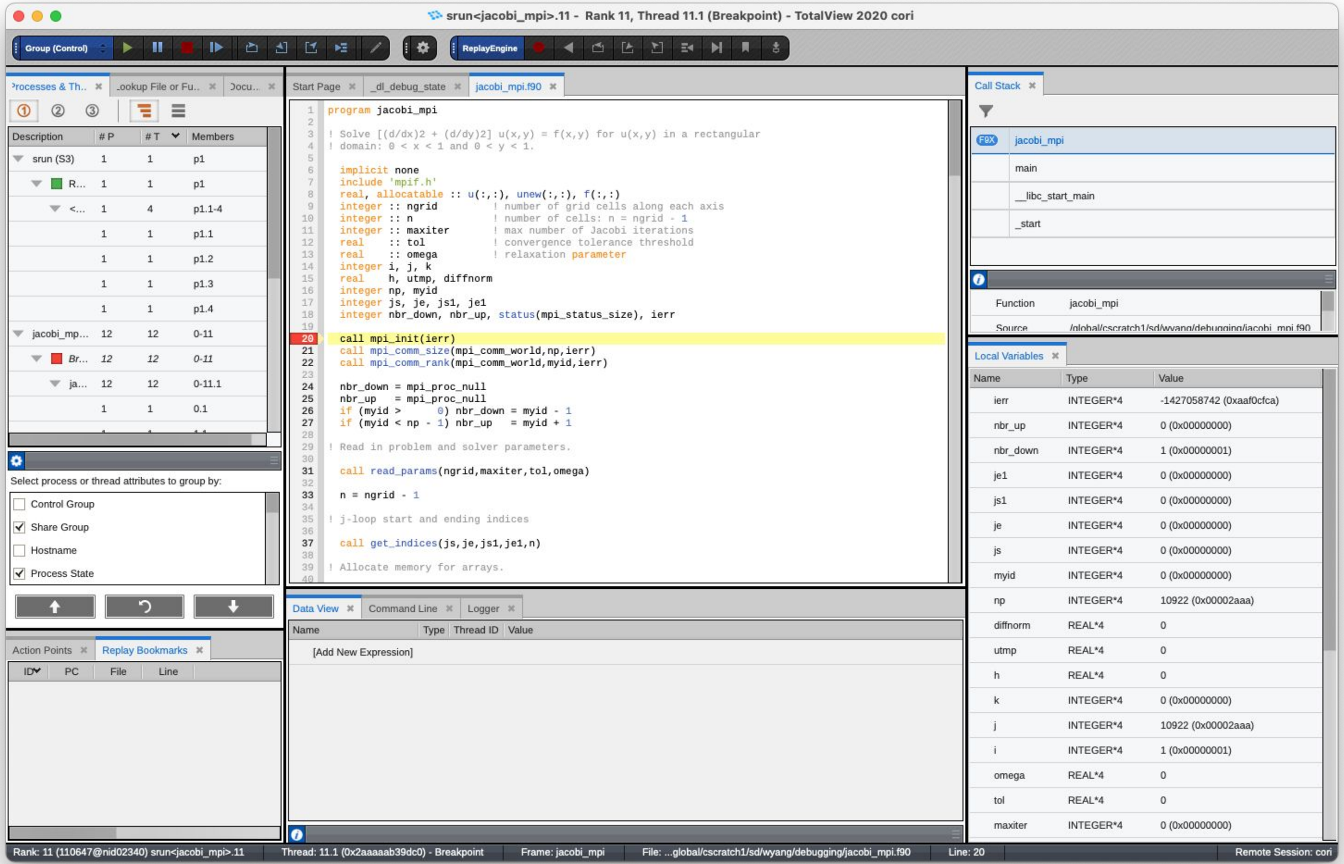
Debugging with TotalView

- Developed by Perforce
  - TotalView (https://totalview.io/)
- Supports core languages: C/C++, Fortran, Python
- Supports several parallel programming models
  - MPI, OpenMP, CUDA, ROCM
- Documentation
  - Perforce: https://help.totalview.io/
  - NERSC: https://docs.nersc.gov/tools/debug/totalview/

Usage

- Load the totalview module
- Compile your program
- Allocate your compute nodes
- Run totalview
  - Run your program
  - Attach to an already running program

```
$ module load totalview
$ ftn -g -O0 -o program program.f90
$ salloc -N1 -q interactive -C cpu [options]
$ totalview [srun -a <srun args> ./program]
```


srun<jacobi_mpi>.11 - Rank 11, Thread 11.1 (Breakpoint) - TotalView 2020 cori

Description	# P	# T	Members
srun (S3)	1	1	p1
🟩 R...	1	1	p1
<...	1	4	p1.1-4
	1	1	p1.1
	1	1	p1.2
	1	1	p1.3
	1	1	p1.4
jacobi_mp...	12	12	0-11
🟥 *Br...*	*12*	*12*	*0-11*
ja...	12	12	0-11.1
	1	1	0.1

Select process or thread attributes to group by:

- [ ] Control Group
- [x] Share Group
- [ ] Hostname
- [x] Process State

```
program jacobi_mpi

! Solve [(d/dx)2 + (d/dy)2] u(x,y) = f(x,y) for u(x,y) in a rectangular
! domain: 0 < x < 1 and 0 < y < 1.

  implicit none
  include 'mpif.h'
  real, allocatable :: u(:,:), unew(:,:), f(:,:)
  integer :: ngrid          ! number of grid cells along each axis
  integer :: n              ! number of cells: n = ngrid - 1
  integer :: maxiter        ! max number of Jacobi iterations
  real    :: tol            ! convergence tolerance threshold
  real    :: omega          ! relaxation parameter
  integer i, j, k
  real    h, utmp, diffnorm
  integer np, myid
  integer js, je, js1, je1
  integer nbr_down, nbr_up, status(mpi_status_size), ierr

  call mpi_init(ierr)
  call mpi_comm_size(mpi_comm_world,np,ierr)
  call mpi_comm_rank(mpi_comm_world,myid,ierr)

  nbr_down = mpi_proc_null
  nbr_up   = mpi_proc_null
  if (myid >      0) nbr_down = myid - 1
  if (myid < np - 1) nbr_up   = myid + 1

! Read in problem and solver parameters.

  call read_params(ngrid,maxiter,tol,omega)

  n = ngrid - 1

! j-loop start and ending indices

  call get_indices(js,je,js1,je1,n)

! Allocate memory for arrays.
```

Call Stack:

- jacobi_mpi
- main
- __libc_start_main
- _start

Function: jacobi_mpi
Source: /global/cscratch1/sd/wyang/debugging/jacobi_mpi.f90

Local Variables

Name	Type	Value
ierr	INTEGER*4	-1427058742 (0xaaf0cfca)
nbr_up	INTEGER*4	0 (0x00000000)
nbr_down	INTEGER*4	1 (0x00000001)
je1	INTEGER*4	0 (0x00000000)
js1	INTEGER*4	0 (0x00000000)
je	INTEGER*4	0 (0x00000000)
js	INTEGER*4	0 (0x00000000)
myid	INTEGER*4	0 (0x00000000)
np	INTEGER*4	10922 (0x00002aaa)
diffnorm	REAL*4	0
utmp	REAL*4	0
h	REAL*4	0
k	INTEGER*4	0 (0x00000000)
j	INTEGER*4	10922 (0x00002aaa)
i	INTEGER*4	1 (0x00000001)
omega	REAL*4	0
tol	REAL*4	0
maxiter	INTEGER*4	0 (0x00000000)

Name	Type	Thread ID	Value
[Add New Expression]			

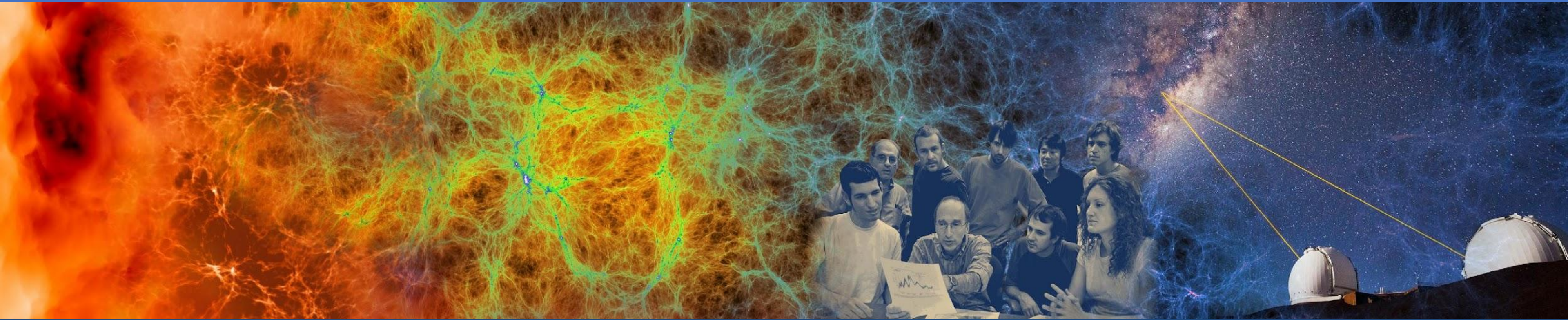
ID	PC	File	Line

Rank: 11 (110647@nid02340) srun<jacobi_mpi>.11
Thread: 11.1 (0x2aaaaab39dc0) - Breakpoint
Frame: jacobi_mpi
File: ...global/cscratch1/sd/wyang/debugging/jacobi_mpi.f90
Line: 20
Remote Session: cori

Reverse connection using the remote client

- Download the remote client
  - https://totalview.io/downloads
- Setup your remote connection
  - https://docs.nersc.gov/tools/debug/totalview/#remote-connections
- Connect to Perlmutter
- Start a remote connection
  - https://docs.nersc.gov/tools/debug/totalview/#starting-a-job-with-totalview

Notes on gdb



Debugging with gdb (GNU Debugger)

- Text-based, open source software
  - https://www.sourceware.org/gdb/
- Supports core languages: C/C++, Fortran
- Traditionally a serial program debugger

```
$ man gdb
$ gdb --help
$ gdb
(gdb) help
(gdb) help run
(gdb) help attach
```

cuda-gdb

- Open source software developed by nvidia
  - https://docs.nvidia.com/cuda/cuda-gdb/index.html
- Supports CUDA only

```
$ module load cudatoolkit
$ cuda-gdb --help
$ cuda-gdb
(cuda-gdb) help
(cuda-gdb) help cuda
```

gdb4hpc

- Developed by HPE
  - https://cpe.ext.hpe.com/docs/debugging-tools/index.html#gdb4hpc
  - https://docs.nersc.gov/tools/debug/gdb4hpc_ccdb
- Support more in-line with parallel debuggers like DDT (kokkos + raja)
- Comparative debugging support

```
$ module load gdb4hpc
$ man gdb4hpc
$ salloc [options]
$ gdb4hpc --help
$ gdb4hpc
dbg all> help
dbg all> help launch
```

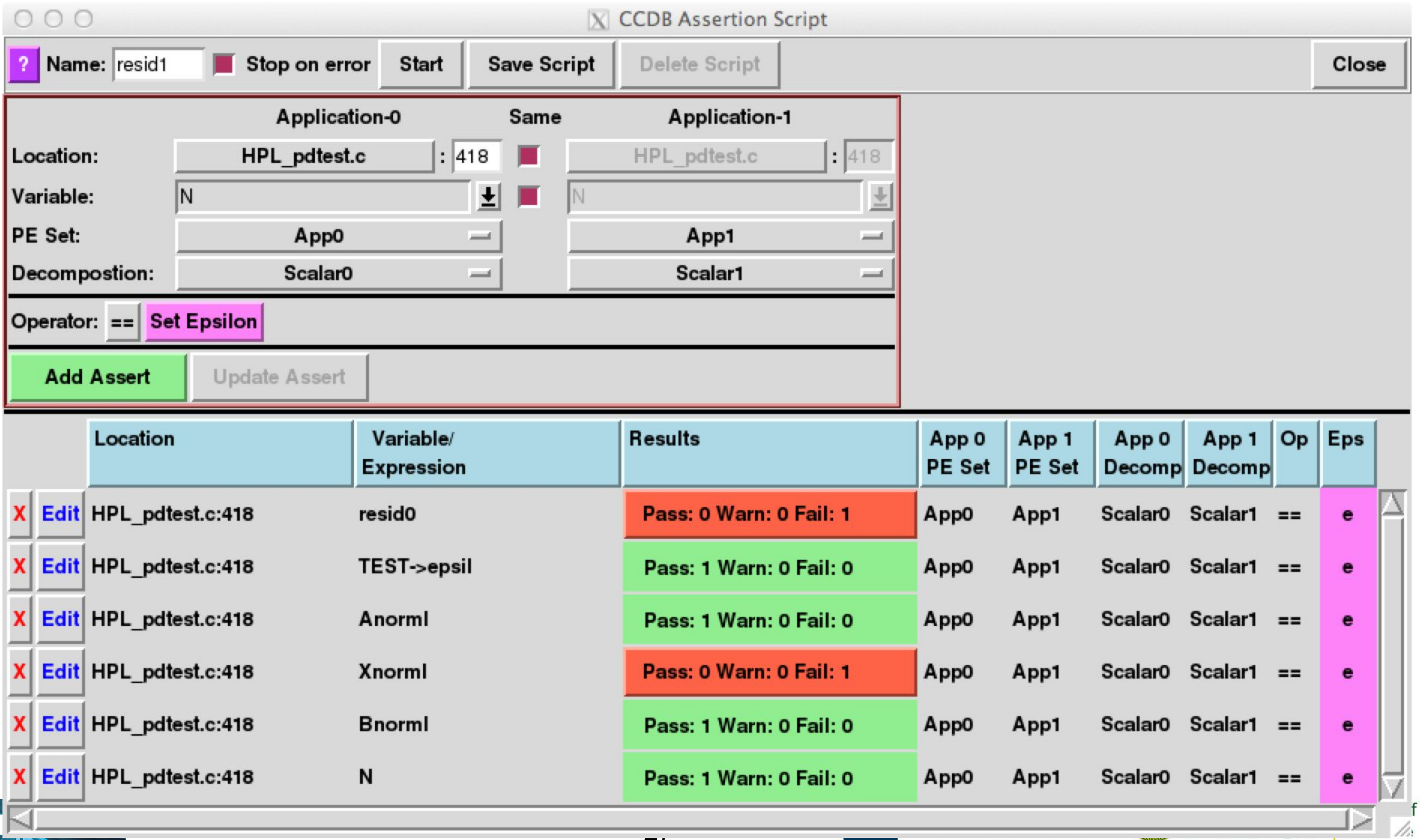
Using gdb4hpc

```
dbg all> launch $pset{8} ./hello_mpi    # Launch 'hello_mpi' using 8 tasks named '$pset'

dbg all> viewset $pset                  # Display the PE set thus defined
Name         Procs
pset         pset{0..7}

dbg all> bt                             # Show where I am - the backtrace
pset{0..7}: #0  0x00000000200009c5 in main at /global/cscratch/sd/elvis/hello_mpi.c:8

dbg all> break hello_mpi.c:11           # Set a breakpoint at line 11 of hello_mpi.c
dbg all> continue                       # Run

dbg all> print myRank                   # Print the value of 'myRank' for all processes
pset[0]: 0
...
pset[7]: 7
dbg all> print $pset{3}::myRank         # Print the value of 'myRank' for rank 3 only
pset[3]: 3
```

ccdb (Cray Comparative Debugger)

- Developed by HPE
  - https://cpe.ext.hpe.com/docs/debugging-tools/index.html#ccdb-cray-comparative-debugger
  - https://docs.nersc.gov/tools/debug/gdb4hpc_ccdb
- Combines gdb4hpc with a GUI to compare two programs in a debugging sessions

```
$ module load cray-ccdb
$ man ccdb
$ ccdb --help
```



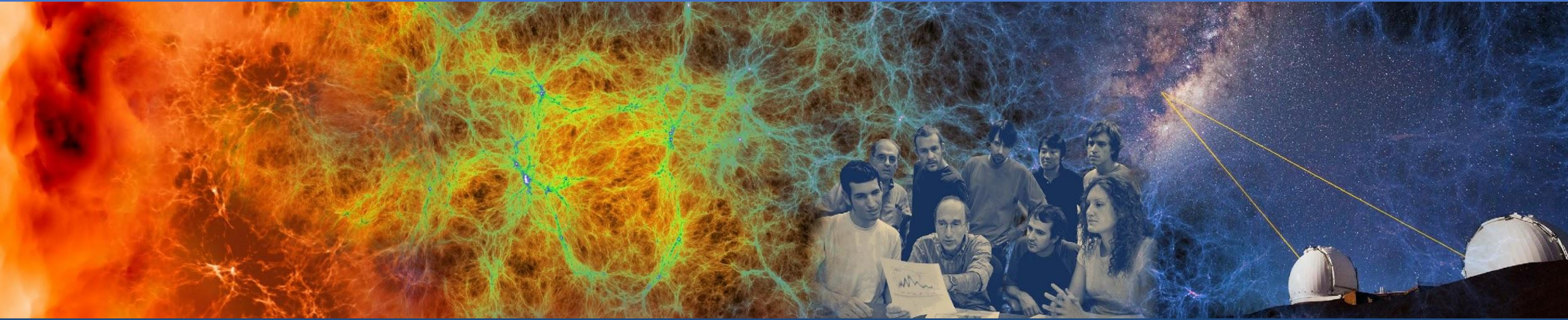
Name: resid1 Stop on error

	Application-0		Same	Application-1	
Location:	HPL_pdtest.c	418	■	HPL_pdtest.c	418
Variable:	N		■	N	
PE Set:	App0			App1	
Decompostion:	Scalar0			Scalar1	

Operator: ==

		Location	Variable/ Expression	Results	App 0 PE Set	App 1 PE Set	App 0 Decomp	App 1 Decomp	Op	Eps
X	Edit	HPL_pdtest.c:418	resid0	Pass: 0 Warn: 0 Fail: 1	App0	App1	Scalar0	Scalar1	==	e
X	Edit	HPL_pdtest.c:418	TEST->epsil	Pass: 1 Warn: 0 Fail: 0	App0	App1	Scalar0	Scalar1	==	e
X	Edit	HPL_pdtest.c:418	Anorml	Pass: 1 Warn: 0 Fail: 0	App0	App1	Scalar0	Scalar1	==	e
X	Edit	HPL_pdtest.c:418	Xnorml	Pass: 0 Warn: 0 Fail: 1	App0	App1	Scalar0	Scalar1	==	e
X	Edit	HPL_pdtest.c:418	Bnorml	Pass: 1 Warn: 0 Fail: 0	App0	App1	Scalar0	Scalar1	==	e
X	Edit	HPL_pdtest.c:418	N	Pass: 1 Warn: 0 Fail: 0	App0	App1	Scalar0	Scalar1	==	e

Notes on STAT and ATP

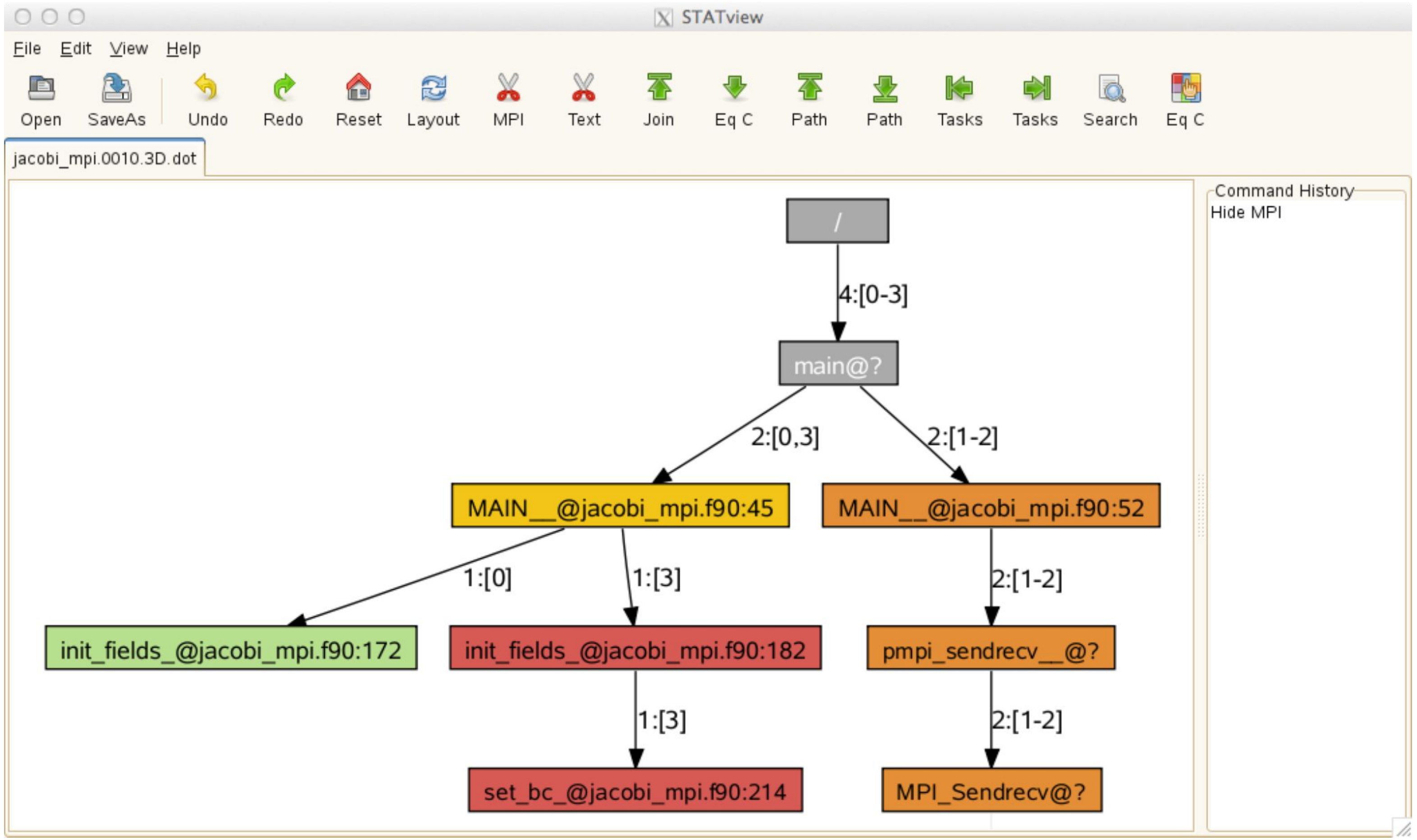


Debugging with STAT (Stack Trace Analysis Tool)

- Developed by HPE/Cray
  - https://cpe.ext.hpe.com/docs/debugging-tools/index.html#stat-stack-trace-analysis-tool
  - https://docs.nersc.gov/tools/debug/stat_atp/#stat
- Attaches to a job launcher process
- Gathers and merges stack traces from all processes
- Supports MPI, threads, and cuda (using cuda-gdb)

Using STAT

```
$ module load cray-cti
$ module load cray-stat
$ export CTI_WLM_IMPL=slurm
$ srun [options] ./program
$ man intro_stat
$ stat-cl [options] <srun pid>  # text-based
$ stat-gui [options] <srun pid>
```



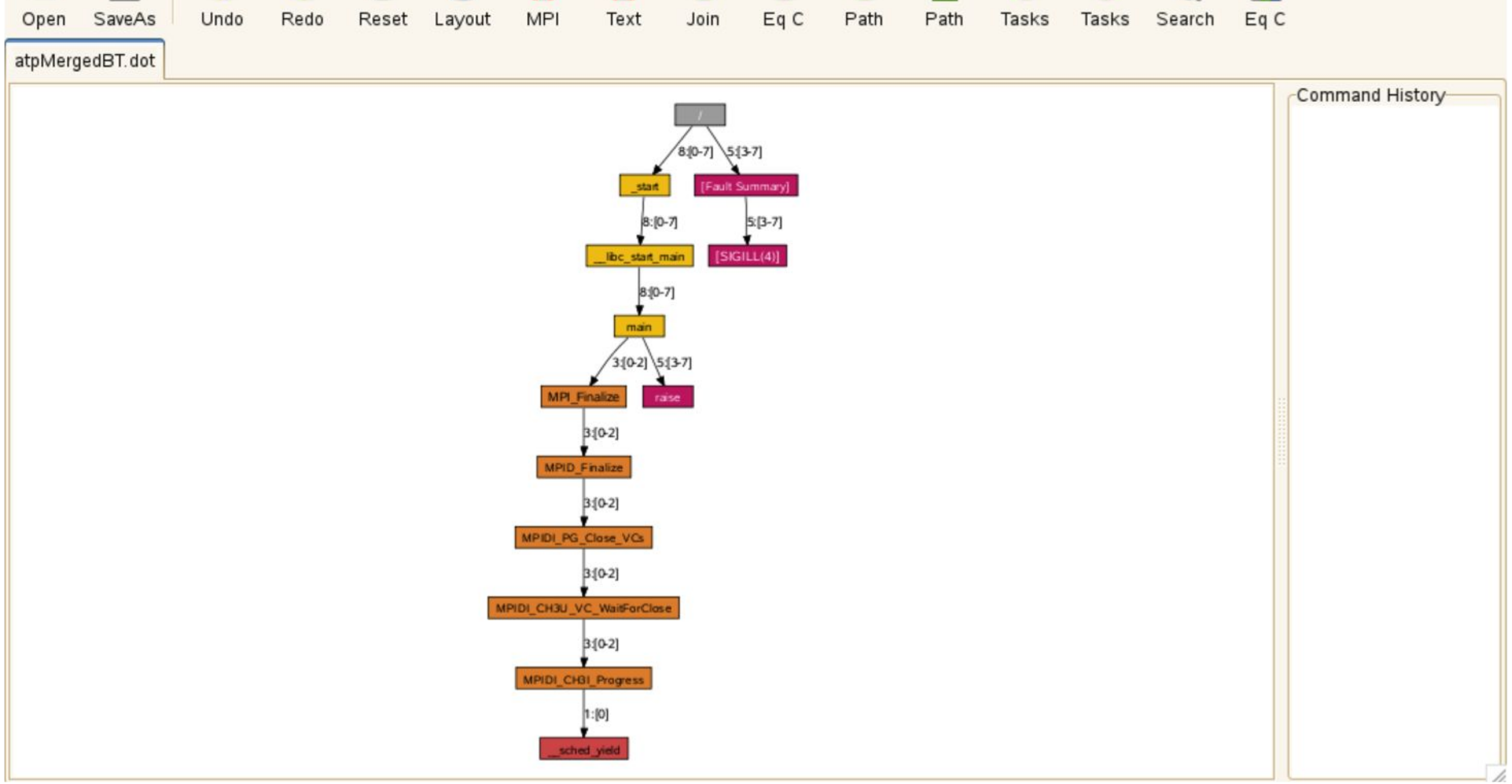
Command History
Hide MPI

Debugging with ATP (Abnormal Termination Processing)

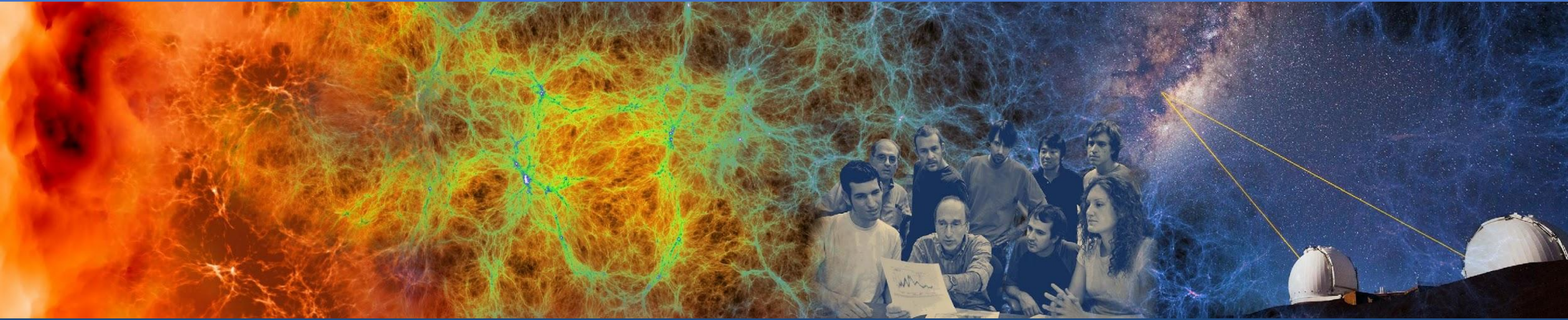
- Developed by HPE/Cray
  - https://cpe.ext.hpe.com/docs/debugging-tools/index.html#atp-abnormal-termination-processing
- Signal handler that processes termination signals from your program
- Uses stat to create and view merged stack traces
- Selectively produces core files
- Supports MPI, threads, and cuda (using cuda-gdb)
- Requires compile with ‘fno-backtrace’ if using the GNU Fortran compiler

Using ATP

```
$ module load cray-cti
$ module load cray-stat
$ module load atp
$ man intro_atp
$ export CTI_WLM_IMPL=slurm
$ export ATP_ENABLED=1
$ export ATP_GDB_BINARY=$(which gdb) #optional
$ export FOR_IGNORE_EXCEPTIONS=true # Intel Fortran
$ srun [options] ./program
<termination signal>
Application 3169879 is crashing. ATP analysis proceeding...
$ stat-view atpMergedBT.dot
```


Open
SaveAs
Undo
Redo
Reset
Layout
MPI
Text
Join
Eq C
Path
Path
Tasks
Tasks
Search
Eq C
atpMergedBT.dot
Command History
/
8:[0-7]
5:[3-7]
_start
[Fault Summary]
8:[0-7]
5:[3-7]
__libc_start_main
[SIGILL(4)]
8:[0-7]
main
3:[0-2]
5:[3-7]
MPI_Finalize
raise
3:[0-2]
MPID_Finalize
3:[0-2]
MPIDI_PG_Close_VCs
3:[0-2]
MPIDI_CH3U_VC_WaitForClose
3:[0-2]
MPIDI_CH3I_Progress
1:[0]
__sched_yield

Notes on valgrind and llvm-sanitizers



Debugging with valgrind

- Text-based, open source software
  - https://valgrind.org/
- Uses several tools to check for program correctness at run-time using dynamic recompilation
- Tools: Memcheck, Cachegrind, Callgrind
  - https://valgrind.org/info/tools.html

valgrind4hpc

- Developed by HPE
  - https://cpe.ext.hpe.com/docs/debugging-tools/index.html#valgrind4hpc
- Based on valgrind
- Aggregates results across all processes

Using valgrind4hpc

```
$ module load cray-cti
$ module load valgrind4hpc
$ export CTI_WLM_IMPL=slurm
$ man valgrind4hpc
$ valgrind4hpc --help
$ # valgrind4hpc [options] program [args]
$ valgrind4hpc -n4 --launcher-args="-N2" ./program
```

Debugging with llvm-sanitizers

- Text-based, open source software
  - https://clang.llvm.org/docs/index.html
- Uses several tools to check program correctness at run-time by instrumenting the source code
- Tools: Address, Leak, Thread

sanitizers4hpc

- Developed by HPE
  - https://cpe.ext.hpe.com/docs/debugging-tools/index.html#sanitizers4hpc
- Based on llvm-sanitizers
- Aggregates results across all processes
- Supports CCE, GCC
- Supports CUDA with compute-sanitizer
  - https://docs.nvidia.com/compute-sanitizer/ComputeSanitizer/index.html

Using sanitizers4hpc

```
$ module load cray-cti
$ module load sanitizers4hpc
$ export CTI_WLM_IMPL=slurm
$ cc -fsanitize=address -o program program.c
$ sanitizers4hpc [launcher args] ./program
```

Thank You and Welcome to NERSC!