Debugging Tools on Perlmutter
Outline

- Debug programs graphically with DDT and TotalView - full-fledged debuggers
- CUDA-gdb and Compute Sanitizer (aka CUDA-memcheck) are non-MPI CUDA debuggers provided by NVIDIA
- Can debug parallel programs with gdb4hpc, another text-based GDB-like tool
- Find memory-related bugs with valgrind4hpc and sanitizers4hpc
- Debug crashed or deadlocked programs with STAT and ATP
- [https://docs.nersc.gov/tools/debug/](https://docs.nersc.gov/tools/debug/)
Before we start debugging

- Setup a remote desktop connection
- Compile your program
- Setup your environment
- Allocate compute resources
Setup a remote desktop connection

- NoMachine (https://docs.nersc.gov/connect/nx/)
- Better performance than traditional x11 forwarding over ssh
- DDT and TotalView have their own remote connections that can also be used
Compile your program

- Generate debugging data and disable compiler optimizations
- C
  - `cc -g -O0 -o program program.c`
- Fortran
  - `ftn -g -O0 -o program program.f90`
- CUDA
  - `nvcc -g -O0 -G -o program program.cu`
Setup your environment

- Allow creation of core files
  - `ulimit -c unlimited`

- Abort and create core file on error
  - `export MPICH_ABORT_ON_ERROR=1`
  - `export CUDA_ENABLE_COREDUMP_ON_EXCEPTION=1`

- Use cray-cti module for HPE / Cray tools
  - `module load cray-cti`
  - `export CTI_WLM_IMPL=slurm`
  - [https://github.com/common-tools-interface/cti](https://github.com/common-tools-interface/cti)
Allocating nodes for debugging

- CPU
  - `salloc [options] -q interactive -C cpu`
- GPU
  - `salloc [options] -q interactive -C gpu`
- [https://docs.nersc.gov/jobs/policy/#qos-limits-and-charges](https://docs.nersc.gov/jobs/policy/#qos-limits-and-charges)
Distributed Debugging Tool (DDT)
Debugging programs with DDT

- Supports MPI, OpenMP, OpenACC, CUDA, Python
- Developed by Linaro (previously Allinea, ARM)
- Remote client
  - [https://docs.nersc.gov/tools/debug/ddt/#reverse-connect-using-remote-client](https://docs.nersc.gov/tools/debug/ddt/#reverse-connect-using-remote-client)
- `module load forge`
- `ddt [options] ./program`
- [https://docs.nersc.gov/tools/debug/ddt/](https://docs.nersc.gov/tools/debug/ddt/)
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
Parallel stack frame view is helpful in quickly finding out where each process is executing.

To check the value of a variable, right-click on a variable or check the pane on the right.

Evaluate expressions

Processing entity to control

Sparklines
Debugging programs with TotalView

- Supports MPI, OpenMP, OpenACC, CUDA
- Developed by Perforce
- Remote client
  - https://docs.nersc.gov/tools/debug/totalview/#remote-display-client
- Remote connection
  - https://docs.nersc.gov/tools/debug/totalview/#remote-connections
- module load totalview
- totalview srun -a [options] ./program
- man totalview
- https://docs.nersc.gov/tools/debug/totalview/
- Review a previous training session
  - https://www.nersc.gov/users/training/events/totalview-tutorial-september-29-2022/
For navigation

- GPU focus thread selector (Logical or Physical)
- GPU focus thread ID (negative) and its coords
- Call backtrace
- Shows state of MPI tasks and threads
- A CPU thread assigned a positive thread ID; A CUDA thread assigned a negative ID
- Breakpoints, threads, etc.

To see the value of a variable, right-click on a variable to "dive" on it or just hover mouse over it

For selecting MPI task and thread

Process window

Root window
CUDA for the GNU Debugger (CUDA-gdb)
Debug cuda programs with cuda-gdb

- An extension to GDB that supports cuda programs
- Developed by NVIDIA
- `cuda-gpu [options] ./program [core-file]`
- `(cuda-gdb) help`
- `(cuda-gdb) bt`
- `(cuda-gdb) list`
- `(cuda-gdb) help cuda`
- `cuda-gdb --help`
Compute Sanitizer (aka CUDA-memcheck)
Perform dynamic analysis of CUDA programs with compute-sanitizer

• Use several tools to check program correctness at run-time
• Dynamic instrumentation at compile time
• Developed by NVIDIA

• `srun compute-sanitizer --tool=memcheck [options] ./program`
• memcheck, Detect memory errors
• racecheck, Detect race conditions
• initcheck, Detect use of uninitialized variables
• syncheck, Detect sync errors

• [https://docs.nvidia.com/compute-sanitizer/ComputeSanitizer/index.html](https://docs.nvidia.com/compute-sanitizer/ComputeSanitizer/index.html)
GNU Debugger for HPC (gdb4hpc)
Debug parallel programs with gdb4hpc

• An extension to GDB that supports parallel programming models
• Does not support GPUs
• Developed by Cray / HPE
• module load gdb4hpc
• gdb4hpc
  • (debug all) launch $p{N} ./program
  • (debug all) help viewset
  • (debug all) viewset $p
• man gdb4hpc
• https://docs.nersc.gov/tools/debug/gdb4hpc_ccdb/#parallel-debugging-with-gdb4hpc
dbg all> launch $p{8} ./pcm
Starting application, please wait...
Creating MRNet communication network...
sbcast: error: No compression library available, compression disabled.
sbcast: error: No compression library available, compression disabled.
Waiting for debug servers to attach to MRNet communications network...
Timeout in 400 seconds. Please wait for the attach to complete.
Number of dbgsrvs connected: [0]; Timeout Counter: [1]
Number of dbgsrvs connected: [1]; Timeout Counter: [0]
Number of dbgsrvs connected: [1]; Timeout Counter: [1]
Number of dbgsrvs connected: [1]; Timeout Counter: [2]
Number of dbgsrvs connected: [8]; Timeout Counter: [0]
Finalizing setup...
Launch complete.
p{0..7}: Initial breakpoint, main at /global/u1/j/jscook/hpe-tools/pi_calc_mpi.c:25

dbg all> list
p{0..7}: 25 reqSteps = 31415; /* running about 31 thousand steps */
p{0..7}: 26 mySumBuf = 0;
p{0..7}: 27 mySum = 0;
p{0..7}: 28
p{0..7}: 29 MPI_Init(&argc, &argv);
p{0..7}: 30 MPI_Comm_rank(MPI_COMM_WORLD, &myRank);
p{0..7}: 31 MPI_Comm_size(MPI_COMM_WORLD, &mpiSize);
p{0..7}: 32 if (myRank == 0) {
p{0..7}: 34

dbg all> viewset $p
Name Procs
p p{0..7}

```
dbg all> print $p::myRank
```

```
p{0..7}: 0
```

Breakpoint 1, main at /global/u1/j/jscook/hpe-tools/pi_calc_mpi.c:31
```
dbg all> print $p::myRank
```
```
p{0}: 0
p{1}: 1
p{2}: 2
p{3}: 3
p{4}: 4
p{5}: 5
p{6}: 6
p{7}: 7
```
```
dbg all> list
```
```
p{0..7}: 31    MPI_Comm_size(MPI_COMM_WORLD, &mpiSize);
p{0..7}: 32
```
```
p{0..7}: 33    if (myRank == 0) {
p{0..7}: 34
```
```
p{0..7}: 35        /* sum my share of the series */
p{0..7}: 36        mySum = sumFractions(reqSteps, mpiSize, myRank);
p{0..7}: 37
```
```
p{0..7}: 38    /* add up sums from all nodes */
p{0..7}: 39    for (int srcRank = 1; srcRank < mpiSize; srcRank++){
p{0..7}: 40        MPI_Recv(&mySumBuf, 1, MPI_DOUBLE, srcRank, 0, MPI_COMM_WORLD, &myStatus);
```
```
```
Valgrind for HPC (valgrind4hpc)
Perform dynamic analysis of parallel programs with valgrind4hpc

• Use several tools to check program correctness at run-time
• Dynamic instrumentation at compile time
• Does not support GPUs
• Based on valgrind
• Developed by HPE / Cray
• Aggregated messages/results from all MPI ranks
• Less spurious error messages than valgrind
• module load valgrind4hpc
• valgrind4hpc -n4 --tool=memcheck [launcher-args] [valgrind-args] ./program
• Tools: memcheck, helgrind, exp-sgcheck, drd
• man valgrind4hpc
• man valgrind
• https://docs.nersc.gov/tools/debug/valgrind/
Perform dynamic analysis of parallel programs with sanitizers4hpc

- Use several tools to check program correctness at run-time
- Static instrumentation at compile time
- Aggregates report across all processes
- Based on LLVM Sanitizers
- Supports CCE, GCC
- Supports GPUs with cuda-memcheck
- Developed by HPE / Cray
- module swap PrgEnv-gnu PrgEnv-cray
- module load sanitizers4hpc
- -fsanitize=<sanitizer>
- Sanitizers: Address, Leak, Thread
- sanitizers4hpc -l "-n4" -- ./program
- man sanitizers4hpc
- https://github.com/google/sanitizers
Stack Trace Analysis Tool (STAT)
Debug deadlock programs with STAT

- Attach to a job launcher process
- Gathers and merges stack traces for all processes
- Supports MPI, threads, and cuda (using cuda-gdb)
- `module load cray-stat`
- `srun [options] program &`
- `<program-pid will output>`
- `stat-cl [options] program-pid`
- `stat-view stat-output-file`
- `man stat-cl`
- `man stat-view`
- [https://docs.nersc.gov/tools/debug/stat_atp/#stat](https://docs.nersc.gov/tools/debug/stat_atp/#stat)
$ ftn -g -o jacobi_mpi jacobi_mpi.f90
$ salloc -N 1 -t 30:00 -q debug -C knl
...
$ srun -n 4 -c 64 --cpu-bind=cores ./jacobi_mpi &
[1] 135543
$ module load stat
$ stat-cl -i 135543
...  
Attaching to application...  
Attached!  
Application already paused... ignoring request to pause  
Sampling traces...  
Traces sampled!  
...  
Resuming the application...  
Resumed!  
Merging traces...  
Traces merged!  
Detaching from application...  
Detached!

Results written to /global/cscratch1/sd/wyang/debugging/stat/stat_results/jacobi_mpi.0003

$ ls -l stat_results/jacobi_mpi.0003/*
dot
-rw-rw---- 1 wyang wyang 5201 Jun 7 14:55 stat_results/jacobi_mpi.0003/00_jacobi_mpi.0003.3D.dot
$ STATview stat_results/jacobi_mpi.0003/00_jacobi_mpi.0003.3D.dot
Abnormal Termination Processing (ATP)
Debugging crashed programs with ATP

• Signal handler processes termination signals
• Gathers and merges stack traces for all processes
• Selectively produces core files
• Support MPI, threads, and cuda (using cuda-gdb)
• module load cray-stat
• module load atp
• export ATP_ENABLED=1
• export ATP_GDB_BINARY=$(which gdb) # optional
• -fno-backtrace # GNU Fortran
• srun [options] program
• <termination signal> or <app crashes>
• stat-view dot-file
• man atp
• https://docs.nersc.gov/tools/debug/stat_atp/#atp
Thank You and Welcome to NERSC!