Debugging Tools

New User Training
June 16, 2020

Woo-Sun Yang
User Engagement Group
# Debuggers

- **Program errors:**
  - Program crashes, program hangs, program generates incorrect results, ...

- **How to find and fix them?**

<table>
<thead>
<tr>
<th></th>
<th>Using print statements</th>
<th>Using debuggers</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Typical workflow</strong></td>
<td>• Add print statements around suspicious or strategic locations in the source code</td>
<td>• Start your program under a debugger</td>
</tr>
<tr>
<td></td>
<td>• Compile the code</td>
<td>• Set breakpoints in your program</td>
</tr>
<tr>
<td></td>
<td>• Run the program and examine the printed values to get a hint about what or where the problem may be</td>
<td>• Run</td>
</tr>
<tr>
<td></td>
<td>• If no hint is obtained, add different print statements</td>
<td>• When the program stops at the breakpoints, check variables</td>
</tr>
<tr>
<td></td>
<td>• Repeat</td>
<td>• Can add more breakpoints and continue</td>
</tr>
<tr>
<td><strong>Pro and con</strong></td>
<td>• “Easy” – no need to learn about a debugging tool</td>
<td>• Compile only once (in general)</td>
</tr>
<tr>
<td></td>
<td>• Difficult to guess where and what to print</td>
<td>• Control program execution (stop, continue, …)</td>
</tr>
<tr>
<td></td>
<td>• Time consuming and tedious</td>
<td>• Tools and features available to aid to spot problem areas (e.g., visually check for abnormality in variable values by plotting them with the debugger’s visualization tool)</td>
</tr>
<tr>
<td></td>
<td>• Rebuild the code each time the code is modified</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Will likely use multiple batch jobs – inefficient use of allocations</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Not easy to understand what is wrong from the potentially long printed values (e.g., multi-dim arrays)</td>
<td></td>
</tr>
</tbody>
</table>
Parallel Debuggers Available on Cori

● Parallel debuggers with a graphical user interface (GUI)
  ○ DDT (Distributed Debugging Tool) – part of the Arm Forge tool
  ○ TotalView

● Specialized debuggers
  ○ STAT (Stack Trace Analysis Tool)
    ■ Collect stack backtraces from all (MPI) tasks
  ○ ATP (Abnormal Termination Processing)
    ■ STAT invoked when an application fails
  ○ Valgrind
    ■ Suite of debugging and profiling tools
    ■ Best known for its detailed memory debugging tool, ‘memcheck’
    ■ [https://docs.nersc.gov/development/performance-debugging-tools/valgrind/](https://docs.nersc.gov/development/performance-debugging-tools/valgrind/)
  ○ Intel Inspector
    ■ Threading and memory debugging
    ■ [https://docs.nersc.gov/programming/performance-debugging-tools/inspector/](https://docs.nersc.gov/programming/performance-debugging-tools/inspector/)
DDT and TotalView

● GUI-based traditional parallel debuggers
● C, C++, Fortran codes with MPI, OpenMP, pthreads
● Licenses
  ○ DDT: up to 4,096 processes
  ○ TotalView: up to 512 processes
  ○ Shared among users and machines
● For info:
  ○ https://developer.arm.com/docs/101136/latest/arm-forge
  ○ https://docs.nersc.gov/development/performance-debugging-tools/ddt/
  ○ https://docs.nersc.gov/development/performance-debugging-tools/totalview/
How to Build and Run with DDT

- Compile with `-g` for debugging symbols and `-O0` for no optimization (Intel compiler)
  
  ```bash
  $ ftn -g -O0 -o jacobi_mpi jacobi_mpi.f90
  ```

- Start an interactive batch job and run DDT:
  
  ```bash
  $ salloc -N 1 -t 30:00 -q debug -C knl
  $ module load allinea-forge
  $ ddt ./jacobi_mpi
  ```
If You Work Far Away From NERSC

- Running X11 GUIs over network: it responds painfully slowly due to intrinsically high latency and inefficient bandwidth between X11 client and server

- Two solutions
  - Use NoMachine (NX) to improve the speed
    - Works for X11 window applications
    - [https://docs.nersc.gov/connect/nx/](https://docs.nersc.gov/connect/nx/)
  - Use Arm Forge remote client
    - Run on your desktop/laptop
    - Submit a debugging batch job on a NERSC machine and make the job connect to the client ("reverse connect")
    - Display results in real time
    - [https://docs.nersc.gov/programming/performance-debugging-tools/ddt/#reverse-connect-using-remote-client](https://docs.nersc.gov/programming/performance-debugging-tools/ddt/#reverse-connect-using-remote-client) (for setup)
Arm Forge Remote Client Settings

- See [https://docs.nersc.gov/development/performance-debugging-tools/ddt/](https://docs.nersc.gov/development/performance-debugging-tools/ddt/)
DDT Window

**For navigation**

**Processing entity to control**

**Sparklines**

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

**Evaluate expressions**

Parallel stack frame view is helpful in quickly finding out where each process is executing.
Breakpoints, Watchpoints and Tracepoints

● **Breakpoint**
  ○ Stops execution when a selected line (breakpoint) is reached
  ○ Double click on a line to create one; there are other ways, too

● **Watchpoints for variables or expressions**
  ○ Stops when a variable or an expression changes its value

● **Tracepoints**
  ○ When reached, prints what lines of codes is being executed and the listed variables

● **Can add a condition for an action point**
  ○ Useful inside a loop

● **Can be made active or inactive**
Check Variables

- Right click on a variable for a quick summary
- Variable pane
- Evaluate pane
- Display variable values over processes (Compare across processes) or threads (Compare across threads)
- MDA (Multi-Dimensional Array) Viewer
  - Visualization
  - Statistics
  - Quick sanity check, for ex., after halo exchange...
Start a batch job interactively and run your code with TotalView

$ salloc -N 1 -C knl -t 30:00 -q debug
$ module load totalview
$ export OMP_NUM_THREADS=4
$ totalview srun -a \
   -n 8 -c 32 --cpu-bind=cores ./jacobi_mpiomp

Click ‘OK’ in the ‘Startup Parameters - srun’ window

Click ‘Go’ in the main window

Click ‘Yes’ to the question ‘Process srun is a parallel job. Do you want to stop the job now?’
TotalView (cont’d)

Root window

State of MPI tasks and threads; members denoted roughly as ‘rank.thread’

Process window

For navigation

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

Breakpoints, etc.

For selecting MPI task and thread
STAT (Stack Trace Analysis Tool)

- Gathers stack backtraces (sequence of function calls leading up to the current function) from all (MPI) processes
  - Merge them into a single file (*.dot)
  - Results displayed as a single call tree for all processes
  - Can be useful for debugging a hanging application
  - With the info learned from STAT, can investigate further with DDT or TotalView

- Works for MPI, CAF, UPC, and OpenMP
STAT (Stack Trace Analysis Tool) (cont’d)

- STAT commands (after loading the ‘stat’ module)
  - stat-cl: invokes STAT to gather stack backtraces
  - STATview: A GUI to view the results
  - STATGUI: a GUI to run STAT or view results

- For more info:
  - ‘intro_stat’, ‘STAT’, ‘STATview’ and ‘STATGUI’ man pages
  - /opt/cray/pe/stat/default/doc/stat_userguide.pdf
  - https://docs.nersc.gov/development/performance-debugging-tools/stat_atp/
Debug a Hanging App with STAT

- If your code hangs consistently, use STAT to examine whether MPI processes get stuck.

```
$ 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
```

With usual optimization flags, if any

```
- to get source line numbers
STAT samples stack backtraces 10 times
```
Debug a Hanging App with STAT (cont’d)

Check these locations!
ATP (Abnormal Termination Processing)

- ATP invokes STAT when the application fails
  - Output in atpMergedBT.dot and atpMergedBT_line.dot (showing source line numbers), which are to be viewed with STATview
- The atp module is loaded on Cori by default, but ATP is *not* enabled; to enable:
  
  ```
  $ export ATP_ENABLED=1 # sh/bash/ksh
  % setenv ATP_ENABLED 1 # csh/tcsh
  ```

- For more info
  - ‘intro_atp’ man page
  - [https://docs.nersc.gov/development/performance-debugging-tools/stat_atp/](https://docs.nersc.gov/development/performance-debugging-tools/stat_atp/)
Debug a Hanging App with ATP

Submit a hanging job with ATP enabled

```bash
$ ftn -g -o jacobi_mpi jacobi_mpi.f90
$ cat runit
#!/bin/bash
#SBATCH -N 1
#SBATCH -C knl
...
export ATP_ENABLED=1
export FOR_IGNORE_EXCEPTIONS=true
srun -n 4 -c 64 --cpu-bind=cores ./jacobi_mpi

$ sbatch runit
Submitted batch job 31445729
```

# Enable ATP
# Code built with Intel fortran compiler
Debug a Hanging App with ATP (cont’d)

● From a login node, ssh to a MOM node and cancel the srun job

$ ssh cmom02
...
$ sacct -j 31445729

<table>
<thead>
<tr>
<th>JobID</th>
<th>JobName</th>
<th>Partition</th>
<th>Account</th>
<th>AllocCPUS</th>
<th>State</th>
<th>ExitCode</th>
</tr>
</thead>
<tbody>
<tr>
<td>31445729</td>
<td>runit</td>
<td>debug_knl</td>
<td>nstaff</td>
<td>272</td>
<td>RUNNING</td>
<td>0:0</td>
</tr>
<tr>
<td>31445729.0</td>
<td>jacobi_mpi</td>
<td>nstaff</td>
<td>256</td>
<td>RUNNING</td>
<td>0:0</td>
<td></td>
</tr>
<tr>
<td>31445729.1</td>
<td>cti_dlaun+</td>
<td>nstaff</td>
<td>1</td>
<td>RUNNING</td>
<td>0:0</td>
<td></td>
</tr>
</tbody>
</table>

$ scancel -s ABRT 31445729.0
$ logout

● Dot files are generated; view them with STATview

$ ls -l *.dot
-rw-rw---- 1 wyang wyang 1287 Jun 7 15:31 atpMergedBT.dot
-rw-rw---- 1 wyang wyang 1837 Jun 7 15:31 atpMergedBT_line.dot
$ module load stat
$ STATview atpMergedBT_line.dot
Debug a Hanging App with ATP (cont’d)

Check these locations!

Rank 0 was here
Rank 3 was here
Ranks 1 and 2 were here
Arm Tools Tutorial on July 16, 2020!

● ½-day tutorial for Arm tools
  ○ Arm Forge
    ■ DDT - debugger
    ■ MAP – performance profiling
  ○ Performance Reports: performance summary

● Beginning/Intermediate level

● Will teach how to profile Python apps, too

● By Arm engineer

● Info and registration:
Thank You and Welcome to NERSC!