Debugging and Profiling on LONI clusters

Le Yan
HPC Consultant
User Services
Goals

- Get familiar with TotalView, the primary debugging tool on LONI clusters
- Get familiar with TAU, the primary profiling tool on LONI clusters
Three Steps of Code Development

- **Debugging**
  - Make sure the code runs and yields correct results

- **Profiling**
  - Analyze performance to identify bottlenecks

- **Optimization**
  - Make the code run faster and/or consume less resources
Three Steps of Code Development

- **Debugging**
  - Make sure the code runs and yields correct results
- **Profiling**
  - Analyze performance to identify bottlenecks
- **Optimization**
  - Make the code run faster and/or consume less resources
Debugging Essentials

- Reproducibility
  - Find the scenario where the error is reproducible
- Reduction
  - Reduce the problem to its essence
- Deduction
  - Form hypotheses on what the problem might be
- Experimentation
  - Filter out invalid hypotheses
Debugging Tools

- Write/Print/Printf
- Compiler flags
- Symbolic debugger
  - GDB - GNU debugger
  - IDB - Intel debugger
- Graphic debugger
  - Totalview
  - DDT
  - Valgrind
  - Eclipse
What A Debugger Can and Cannot Do

- What a debugger can do
  - Tell you where the program crashes
  - Help you to gain a better understanding of the context under which it crashes

- What a debugger cannot do
  - Tell you how to solve the problem
  - Detect a correctness problem
    - Validation is very important
What Is TotalView

• A very powerful debugger
  • Can be used to debug both serial and parallel programs (especially good for parallel programs)
  • Supports multiple languages
  • Supported on most architecture/platforms
  • Both graphic and command line interfaces
  • Numerous features
    • Common functionalities such as controlled execution and breakpoints
    • Array visualization
    • Memory debugging
    • ...
Availability

- TotalView is only available on Queen Bee and Eric for LONI users (due to license restriction)
  - Queen Bee and Eric
    - Key: `+totalview-8.3.0.1`
- Add the key to your `.soft` file and `resoft`
Getting X Window To Work

- Linux
  - `ssh -X -Y username@hostname`
- Mac Os X
  - Use X11
- Windows
  - Install Xming and Putty
  - Enable X11 Forwarding in Putty

https://docs.loni.org/wiki/X11_Forwarding
Compiling Your Program

• Serial program
  • Compile with no optimization and debugging flag turned on:
    • `icc -g -O0 -o myexec myprogram.c`

• Parallel program
  • Use the proper MPI implementation
    • Queen Bee: `+mvapich-1.0-intel10.1-tvdbg`
  • Compile with no optimization and debugging flag turned on:
    • `mpicc -g -O0 -o myexec myprogram.c`
Getting An Interactive Session

• We need to get an interactive debugging session
  
  • `qsub -I -x -V -l walltime=hh:mm:ss,nodes=1:ppn=x -A <alloc> -q checkpt`

  ...

  PBS has allocated the following nodes:
  tezpur333
  tezpur331
  ...

  • "-X" for X window tunneling
  • Run TotalView in the interactive session
Starting Totalview

- **Serial program**
  - `totalview <options> <executable>`
  - **Example:** `totalview myexec`

- **Parallel program**
  - `mpirun_rsh -tv -np <num_procs> <host list> <executable>`
  - **Example:** `mpirun_rsh -tv -np 2 tezpur333 tezpur333 myexec`
  - `mpirun_rsh -tv -np <num_procs> -hostfile <path_to_hostfile> <executable>`
  - **Example:** `mpirun_rsh -tv -np 8 -hostfile $PBS_NODEFILE myexec`
TotalView GUI – Root Window

- Always appears when TotalView is started
- Provides an overview of all processes and threads
TotalView GUI – Root Window

<table>
<thead>
<tr>
<th>Status Code</th>
<th>Description</th>
<th>Status Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blank</td>
<td>Exited</td>
<td>E</td>
<td>Error</td>
</tr>
<tr>
<td>B</td>
<td>At Breakpoint</td>
<td>H</td>
<td>Held</td>
</tr>
<tr>
<td>E</td>
<td>Error</td>
<td>H</td>
<td>Held</td>
</tr>
<tr>
<td>H</td>
<td>Held</td>
<td>K</td>
<td>In kernel</td>
</tr>
<tr>
<td>K</td>
<td>In kernel</td>
<td>M</td>
<td>Mixed</td>
</tr>
<tr>
<td>M</td>
<td>Mixed</td>
<td>R</td>
<td>Running</td>
</tr>
<tr>
<td>R</td>
<td>Running</td>
<td>T</td>
<td>Stopped</td>
</tr>
<tr>
<td>T</td>
<td>Stopped</td>
<td>W</td>
<td>At Watchpoint</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TotalView ID</th>
<th>MPI Rank</th>
<th>Host</th>
<th>Status</th>
<th>Host</th>
<th>Status</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>&lt;local&gt;</td>
<td>T</td>
<td>&lt;local&gt;</td>
<td>T</td>
<td>mpirun_rsh (1 active threads)</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>&lt;local&gt;</td>
<td>T</td>
<td>&lt;local&gt;</td>
<td>T</td>
<td>mpirun_rsh&lt;pi_parallel_debug&gt;.0 (2 active threads)</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>&lt;local&gt;</td>
<td>T</td>
<td>&lt;local&gt;</td>
<td>T</td>
<td>mpirun_rsh&lt;pi_parallel_debug&gt;.1 (2 active threads)</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>&lt;local&gt;</td>
<td>T</td>
<td>&lt;local&gt;</td>
<td>T</td>
<td>mpirun_rsh&lt;pi_parallel_debug&gt;.2 (2 active threads)</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>&lt;local&gt;</td>
<td>T</td>
<td>&lt;local&gt;</td>
<td>T</td>
<td>mpirun_rsh&lt;pi_parallel_debug&gt;.3 (2 active threads)</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>tezpur331</td>
<td>T</td>
<td>tezpur331</td>
<td>T</td>
<td>mpirun_rsh&lt;pi_parallel_debug&gt;.4 (2 active threads)</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>tezpur331</td>
<td>T</td>
<td>tezpur331</td>
<td>T</td>
<td>in __read_nocancel</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>tezpur331</td>
<td>T</td>
<td>tezpur331</td>
<td>T</td>
<td>mpirun_rsh&lt;pi_parallel_debug&gt;.5 (2 active threads)</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>tezpur331</td>
<td>T</td>
<td>tezpur331</td>
<td>T</td>
<td>mpirun_rsh&lt;pi_parallel_debug&gt;.6 (2 active threads)</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>tezpur331</td>
<td>T</td>
<td>tezpur331</td>
<td>T</td>
<td>mpirun_rsh&lt;pi_parallel_debug&gt;.7 (2 active threads)</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>tezpur331</td>
<td>T</td>
<td>tezpur331</td>
<td>T</td>
<td>in __read_nocancel</td>
</tr>
</tbody>
</table>
TotalView GUI – Process Window

- Appears when TotalView is started (if a program is specified)
- For parallel programs each process/thread may have its own process window
TotalView GUI – Process Window

- Stack trace pane
  - Call stack of routines
- Stack frame pane
  - Local variables, registers and function parameters
- Source pane
  - Source code
- Action points, processes, threads pane
TotalView GUI – Variable Window

- Can be opened by double-clicking on a variable name (dive)
- Display detailed information of a variable
- One can also edit the data here
Controlling Execution

- Control commands
  - Go – start/resume execution
  - Halt – stop execution
  - Kill – terminate the job
  - Restart – Restarts a running program
  - Next – run to next source line without stepping in to another subroutine/function
  - Step – run to next source line
  - Out – run to the completion of a function
  - ...

![Image of control commands and debugging interface]
Action Points in TotalView

- **Breakpoints** stop the execution of the processes and threads that reach it
  - Can be conditional or unconditional
- **Process barrier points** synchronize a set of processes or threads
- **Evaluation points** cause a code fragment to be executed when reached
- **Watchpoints** let the programmer monitor a location in memory, and stop execution or evaluate an expression when its value changes
Breakpoints

- The most basic action point
- To add a breakpoint
  - Click on the line number
  - Right click on a source line -> Set breakpoint
Evaluation Points

- Stop and execute a code fragments when reached
  - Useful when testing small patches
- To add a code fragment
  - Tools > Evaluate
Watchpoints

- Two types of watchpoints
  - Unconditional
  - Conditional
    - Example: stop the execution after 50k iterations

- To add a watchpoint
  - Right click on a variable -> Create watchpoint
Diving On An Object

- “Diving” means “showing more details on an object”
- By double clicking, one can dive on
  - Variables
  - Processes/threads
  - Subroutines
  - ...
- Use the 'undive' button to go back
Viewing/Editing Data

- View values and types of variables
  - By hovering mouse over the variable
  - In stack frame
  - In variable window

- Edit variable value and type
  - In stack frame
  - In variable window
Handling Arrays (1)

- **Slicing**
  - Display array subsection by editing the **slice** field in the variable window

- **Form**
  - [upper bound:lower bound:stride]
Handling Arrays (2)

- Filtering
  - Display array subsection by applying a filter (filter field in the variable window)
  - Available filter options
    - Arithmetic comparison to a constant
    - Comparison to NaNs and Infs
    - Conditions can be combined by using logic operators
Handling Arrays (3)

- Visualization
  - Variable window -> Tools -> Visualization
- Statistics
  - Variable window -> Tools -> Statistics
Viewing Dynamic Arrays in C/C++

- Edit “type” in the variable window
- Tell TotalView how to interpret the memory from a starting location
- Example
  - To view an array of 100 integers
    - `int * -> int[100]*`
Bugs in Parallel Programs

- Parallel programs are prone to the usual bugs found in sequential programs, plus
  - Erroneous use of language features
    - Mismatched parameters, missing mandatory calls etc.
  - Defective space decomposition
  - Incorrect/improper synchronization
  - Hidden serialization
  - ...

Debugging Parallel programs with TotalView

• Everything we talked about TotalView still works (well, almost)
  • Exceptions: stepping over a communication call while the other processes are stopped or being held

• Additional features
  • Scope of Control Commands
    • Group/Process/Thread
  • Displaying message queues (MPI programs)
Scope of Control Commands

- For serial programs
  - Not an issue because there is only one execution stream

- For parallel programs, we need to decide the scope to which a control command applies
  - The process window always focuses on one process/thread
  - Need to set the appropriate scope when
    - Giving control commands
    - Setting action points
  - Switch between process/threads
    - “p+/p-” and “t+/t-” button
    - Through the root window
    - Through the process/thread tab
Process/Thread Groups

- Group (control): all processes and threads
- Group (workers): all threads that are executing user code
- Rank X: current process and its threads
- Process (workers): user threads in the current process
- Thread X.Y: current thread
- User defined group
  - Group -> Custom Groups, or
  - Create in call graph
Displaying Message Queues

- Detect
  - Deadlocks
  - Load balancing issues
- To access
  - Tools -> Message Queue Graph
Displaying Call Graph

- Quick view of program state
  - Nodes are functions
  - Edges are calls
  - Look for outliers
- To access
  - Tools -> Call Graph
Not Covered

- Memory debugging
  - Leak detection
  - Heap status
  - Memory usage
  - Memory comparison
  - ...
- Command line interface
- Command line options
References and Additional Resources

- TotalView user manual
  - http://www.totalviewtech.com/support/documentation/totalview/
- LLNL Total View tutorial
  - https://computing.llnl.gov/tutorials/totalview/
- HPCBugBase
  - http://www.hpcbugbase.org/index.php/Main_Page
Three Steps of Code Development

- **Debugging**
  - Make sure the code runs and yields correct results

- **Profiling**
  - Analyze performance to identify bottlenecks

- **Optimization**
  - Make the code run faster and/or consume less resources
Profiling

- Gather performance statistics during execution
  - Inclusive and exclusive time
  - Number of calls
- Reflects performance behavior of program entities
  - Routines
  - Loops
- Implemented through
  - Sampling: OS interrupts or hardware counters
  - Instrumentation: calls to measurement functions
What is TAU

- **Tuning and Analysis Utilities**
  - Developed at University of Oregon

- **Scalable and flexible performance analysis toolkit**
  - Performance profiling and tracing utilities
  - Performance data management and data mining
  - Automatic instrumentation through Program Database Toolkit (PDT)
  - Also provides an instrumentation API
Availability

- Linux clusters
  - Softenv key: +tau-2.18-intel-11.1-mvapich-1.1
- AIX clusters
  - Softenv key: +tau-2.1.6
- Note: PAPI is not available at the moment, so TAU is unable to provide hardware counters
Usage

- Add the softenv key to .soft and resoft
- Compile your code with the TAU compiler scripts
  - `tau_f90.sh` for Fortran, `tau_cc.sh` for C and `tau_cxx.sh` for C++
  - Your code will be instrumented automatically
- Execute the generated executable as usual
  - Profile data files `profile.x.x.x` will be generated
- Analyze the results with paraprof (discussed later)
Paraprof

- Java-based analysis and visualization tool for performance data
- 'pprof' is for text based profile display
- Can work with profile data generated by other profiling tools, e.g. MpiP

Options
- -f <file type>: specify type of performance data
- -m: perform runtime monitoring
- --pack <file>: pack profile data into one file
# Main Data Window

## TAU: ParaProf: /work/lyan1/ClusterTest/tautest/lammps-09Jan09/

<table>
<thead>
<tr>
<th>File</th>
<th>Options</th>
<th>Windows</th>
<th>Help</th>
</tr>
</thead>
</table>

**Metric:** Time  
**Value:** Exclusive

<table>
<thead>
<tr>
<th>Std. Dev.</th>
<th>Mean</th>
</tr>
</thead>
<tbody>
<tr>
<td>node 0</td>
<td></td>
</tr>
<tr>
<td>node 1</td>
<td></td>
</tr>
<tr>
<td>node 2</td>
<td></td>
</tr>
<tr>
<td>node 3</td>
<td></td>
</tr>
<tr>
<td>node 4</td>
<td></td>
</tr>
<tr>
<td>node 5</td>
<td></td>
</tr>
<tr>
<td>node 6</td>
<td></td>
</tr>
<tr>
<td>node 7</td>
<td></td>
</tr>
<tr>
<td>node 8</td>
<td></td>
</tr>
<tr>
<td>node 9</td>
<td></td>
</tr>
<tr>
<td>node 10</td>
<td></td>
</tr>
<tr>
<td>node 11</td>
<td></td>
</tr>
<tr>
<td>node 12</td>
<td></td>
</tr>
<tr>
<td>node 13</td>
<td></td>
</tr>
<tr>
<td>node 14</td>
<td></td>
</tr>
<tr>
<td>node 15</td>
<td></td>
</tr>
</tbody>
</table>

```cpp
void LAMMPS_NS::PairLJCut::compute(int, int) {{pair_lj_cut.cpp} {68,1}--{146,1}}
```

- **Exclusive Time:** 14.066 seconds  
- **Inclusive Time:** 14.092 seconds  
- **Calls:** 50001.0  
- **SubCalls:** 100503.0
Main Data Window: Unstacked Bars
Function Data Window: Histogram
Function Data Window: Bar Chart
3D View
Comparing Multiple Threads
Callpath Profile

![Callpath Profile Diagram](image-url)
Options for TAU Compiler Scripts

- Display available options with 'tau_xxx.sh -help'
- Options
  - `-optVerbose`: display verbose debugging information
  - `-optKeepFiles`: keep intermediate files (instrumented source files)
  - `-optDetectMemory`: trace malloc/free calls
  - ...

LONI High Performance Computing Workshop – Louisiana State University
April 26-28, 2010
Notes for Fortran Programmers

- Use `include 'mpif.h'` instead of `use mpi`
- If free format is used with `.f` files, use the `-optPdtF95OptS=-R free` option
- If more than one module files are used, use the `-optPdtGnuFortranParser` option
- If C preprocessor directives are used, use the `-optPreProcess` option
TAU Environment Variables

• TAU provides many environment variables
  • TAU_MAKEFILE
  • TAU_THROTTLE
  • TAU_OPTIONS
  • PROFILEDIR
  • TRACEDIR
  • ...
TAU_MAKEFILE

- Different TAU makefiles corresponds to different configurations
- There are quite a few
  - The default is “icpc-mpi-pdt”

Makefile.tau-intel-11.1-mvapich-1.1-callpath-icpc-mpi-compensate-pdt
Makefile.tau-intel-11.1-mvapich-1.1-callpath-icpc-mpi-pdt
Makefile.tau-intel-11.1-mvapich-1.1-depthlimit-icpc-mpi-pdt
Makefile.tau-intel-11.1-mvapich-1.1-icpc-mpi-compensate-pdt
Makefile.tau-intel-11.1-mvapich-1.1-icpc-mpi-pdt
Makefile.tau-intel-11.1-mvapich-1.1-icpc-pdt
Makefile.tau-intel-11.1-mvapich-1.1-param-icpc-mpi-pdt
TAU_CALLPATH

- Enables callpath profiling
  - Record callpath for each event
  - Need to set TAU_MAKEFILE to one of those with callpath in their names
- TAU_CALLPATH_DEPTH
  - Level to which callpath is recorded
  - Default is 2
  - Overhead increases with the depth of callpath
Other Environment Variables

- **TAU_THROTTLE**
  - Enable event throttling
  - Purpose: reduce profiling overhead
  - If a function executes more than $TAU\_THROTTLE\_NUMCALLS$ times and has an inclusive time per call of less than $TAU\_THROTTLE\_PERCALLS$ microseconds, then profiling of that function will be disabled after that threshold is reached

- **PROFILEDIR**
  - Controls where the profile files are written to (the default is current directory)

- **TAU_OPTIONS**
  - Override the default instrumentation options
Not covered

- Selective profiling
- Tracing
- Database management
- Phase based profiles
- Track memory and IO
- Instrumentation API
- ...
References

- TAU documentation
  - http://www.cs.uoregon.edu/research/tau/docs.php
- ACTS website for TAU
Debugging Exercise

- Files and instructions
  - http://www.cct.lsu.edu/~lyan1/totalview/lsu.php

- Use the serial program to get familiar with the features of TotalView
  - It is bug-free

- Debug the parallel program
  - Check the result by comparing to that of the serial program