Debugging with Totalview and DDT

Le Yan

User Services
HPC @ LSU
Three Steps of Code Development

• Debugging
  – Make sure the code runs and yields correct results

• Profiling
  – Analyze the code to identify performance 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
  – For hypotheses on what the problem might be

• Experimentation
  – Filter out invalid hypotheses
Debugging Methods

• Write/print/printf
• Compiler flags
  – Array bound check, floating point exception etc.
• Debuggers
  – Command line: gdb
  – Graphic: Totalview, DDT, Valgrind, Eclipse
Validation Is Very Important

• Debuggers can tell you where the program crashes and help you to gain better understanding of the context, but
• They cannot detect a correctness problem
• So, it is always a good idea to have test cases with known solutions against which you can validate your program
TotalView & DDT

• Powerful debuggers
  – Can be used to debug both serial and parallel programs
  – Support multiple languages
    • Both supports CUDA
  – Supported on most architecture/platforms
  – Graphic user interface
    • Totalview also has a command line interface
  – Numerous other features
    • Array visualization
    • Memory debugging
    • …
Availability

• TotalView
  – 8.8.0 on Queen Bee (+totalview-8.8.0)
  – 8.3.0 on Queen Bee, Tezpur, Philip and Eric (+totalview-8.3.0.1)

• DDT
  – 2.6 on all LONI and LSU HPC Linux clusters (+ddt-2.6)
Preparing for a Debugging Session

- Compile the program with debugging turned on and optimization turned off (-O0 –g)
- Add softenv keys and resoft
- Make sure X Windows works
- Submit an interactive job session
Working with Debuggers

• One can start debugging by
  – Starting the debugger with the executable
  – Debugging a core dump
  – Attaching to a running (or hanging) process

• Common debugging operations
  – Setting up action points
  – Controlling the execution
  – Examining the value of variables
  – ...

4/4/2012
Launching a Debugging Session

- Serial program
  - Totalview
    - totalview <executable> -a <program options>
  - DDT
    - ddt -start <executable> <program options>

- Parallel program
  - Totalview
    - mpirun_rsh -tv -np <num_procs> <host list> <executable> <program options>
    - mpirun_rsh -tv -np <num_procs> -hostfile <path_to_hostfile> <executable> <program options>
  - DDT
    - ddt -start -np <num_procs> <executable> <program options>
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>
</tr>
</thead>
<tbody>
<tr>
<td>Blank</td>
<td>Exit</td>
</tr>
<tr>
<td>M</td>
<td>Mixed</td>
</tr>
<tr>
<td>R</td>
<td>Running</td>
</tr>
<tr>
<td>T</td>
<td>Stopped</td>
</tr>
<tr>
<td>W</td>
<td>At watchpoint</td>
</tr>
<tr>
<td>B</td>
<td>At breakpoint</td>
</tr>
<tr>
<td>E</td>
<td>Error</td>
</tr>
<tr>
<td>H</td>
<td>Held</td>
</tr>
<tr>
<td>K</td>
<td>In kernel</td>
</tr>
</tbody>
</table>

Status codes and descriptions:
TotalView GUI – Process Window

- Appears when TotalView is started
- 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
  - Lists of action points
  - Lists of processes
  - List of threads
TotalView GUI – Variable Window

• Can be opened by double-clicking on a variable name
  – Called “dive” in TotalView terminology
• Display detailed information of a variable
• One can also edit the data here
DDT GUI

Group/process/thread control

Source

Stack Frame

Breakpoints/Watchpoints/Call stack

Evaluation
Other Ways of Starting a Debugging Session

• Open a core file
  – Need to select an executable
  – Can only browse variables and evaluate expressions since there is no active process

• Attach to one or more running (or hanging) processes
TotalView: Controlling Execution

- Commonly used commands
  - Go: start/resume execution
  - Halt: stop execution
  - Kill: terminate debugging session
  - Restart: restart a running program
  - Next: run to next source line WITHOUT stepping into another function or subroutine
  - Step: run to next source line
  - Out: run to the completion of a function or subroutine
DDT: Controlling Execution

• Similar commands to TotalView
• A few more commands to move up and down stack frame
  – The “align stack frames” command is useful to bring paused processes to the same place in the program
Action Points

- Break points stop the execution when reached
  - Can be conditional
- Barrier points synchronize a set of processes of threads
- Evaluation points cause a code segment to be executed when reached
- Watch points allow the programmer monitor a location in memory
  - Can stop execution or evaluate an expression when its value changes
TotalView: Breakpoints

- How to set
  - Left click on the line number
  - Right click on a line -> “set breakpoint”
- Will appear in the action point list
TotalView: Evaluation Points

- How to set
  - “Tools” -> “Evaluate”
- Execute a small segment of code at specified location
  - Useful when testing on-the-fly fixes
TotalView: Watch Points

- Monitor a memory location and stop execution when it is overwritten
- How to set
  - Right click on a variable -> “Create watchpoint”
- Can be conditional
  - Example: only watch this memory location after a certain number of iterations
DDT: Breakpoints

- How to set
  - Double click on a line
  - Right click on a line -> “Add breakpoint”
- Will appear in the breakpoint list
DDT: Evaluation and Watch Points

• How to set
  – Right click on variable -> “Add to Evaluations” or “Add to Watches”

• DDT does not provide as many options for evaluation and watch points
“Diving” means “showing more details on an object”

- Variables
- Processes/threads
- Subroutines

Use “undive” to go back
TotalView: 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
TotalView: Handling Arrays (1)

- **Slicing**
  - Display array subsection by editing the slice field in the variable window
- **Form**
  - \([\text{upper bound}:\text{lower bound}:\text{stride}]\)
TotalView: 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
TotalView: Handling Arrays (3)

• Visualization
• Statistics
DDT: Handling Arrays
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

• 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 拥+/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
TotalView: Displaying Call Graph

- Quick view of program state
  - Nodes are functions
  - Edges are calls
  - Look for outliers
- To access
  - Tools -> Call Graph
DDT: Parallel Stack View

- Shows a tree of functions merged from every process in a group of processes
- Can create process groups based on their location
- Very helpful when dealing with a large number of processes
Not Covered

• Memory debugging
  – Leak detection
  – Heap status
  – Memory usage
  – Memory comparison
  – ...

• Command line interface
• Command line options