Parallel Debugging Techniques
& Introduction to Totalview

Le Yan
Louisiana Optical Network Initiative
Outline

• Overview of parallel debugging
  – Challenges
  – Tools
  – Strategies

• Get familiar with TotalView through hands-on exercises
Outline

• Overview of parallel debugging
  – Challenges
  – Tools
  – Strategies

• Get familiar with TotalView through hands-on exercises
Bugs in Parallel Programming

• Parallel programs are prone to the usual bugs found in sequential programs
  – Improper pointer usage
  – Stepping over array bounds
  – Infinite loops
  – ...
• Plus...
Common Types of Bugs in Parallel Programming

• Erroneous use of language features
  – Mismatched parameters, missing mandatory calls etc.
• Defective space decomposition
• Incorrect/improper synchronization
• Hidden serialization
• ......
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

Terrence Parr, *Learn The Essentials of Debugging*
Challenges in Parallel Debugging

• Reproducibility
  – Many problems cannot be easily reproduced

• Reduction
  – Smallest scale might still be too large and complex to handle

• Deduction
  – Need to consider concurrent and interdependent program instances

• Experimentation
  – Cyclic debugging might be very expensive
Bugs: A Little Example

- What is the potential problem with large core count?

```fortran
... integer*4 :: i,ista,iend
integer*4 :: chunksize=1024*1024
...
call MPI_Comm_Rank(MPI_COMM_WORLD, &
    myrank,error)
...
ista=myrank*chunksize+1
iend=(myrank+1)*chunksize
do i = ista,iend
    ...
enddo
...
Bugs: A Little Example

- A bug that shows up only when running with more than 4096 cores

```fortran
... integer*4 :: i,ista,iend
integer*4 :: chunksize=1024*1024
...
call MPI_Comm_Rank(MPI_COMM_WORLD, &
     myrank,error)
...
ista=myrank*chunksize+1
iend=(myrank+1)*chunksize
do i = ista,iend
   ... Integer overflow if
   myrank ≥ 4096
   ... enddo
...```

Integer overflow if myrank ≥ 4096
Debugging with write/printf

• Very easy to use and most portable, but...
  – Need to edit, recompile and rerun when additional information is desired
  – May change program behavior
  – Only capable of displaying a subset of the program’s state
  – Output size grows rapidly with increasing core count and harder to comprehend

• Not recommended
Compilers Can Help

• Most compilers can (at runtime)
  – Check array bounds
  – Trap floating operation errors
  – Provide traceback information

• Relatively scalable, but...
  – Overhead added
  – Limited capability
  – Non-interactive
Parallel Debuggers

• Capable of what serials debuggers can do
  – Control program execution
  – Set action points
  – View/edit values of variables

• More importantly
  – Control program execution at various levels
    • Group/process/thread
  – Display communication status between processes
An Ideal Parallel Debugger

- Should allow easy process/thread control and navigation
- Should support multiple high performance computing platforms
- Should not limit the number of processes being debugged and should allow it to vary at runtime
How Parallel Debuggers Work

• Frontend
  – GUI
  – Debugger engine
• Debugger Agents
  – Control application processes
  – Send data back to the debugger engine to analyze
Debugging at Very Large Scale

• The debugger itself becomes a large parallel application

• Bottlenecks
  – Debugger framework startup cost
  – Communication between frontend and agents
  – Access to shared resources, e.g. file system
Validation Is Crucial

- Have a solid validation procedure to check the correctness
- Test smaller components before putting them together
General Parallel Debugging Strategy

• Incremental debugging
  – Downscale if possible
    • Participating processes, problem size and/or number of iterations
    • Example: run with one single thread to detect scope errors in OpenMP programs
  – Add more instances to reveal other issues
    • Example: run MPI programs on more than one node to detect problems introduced by network delays
Strategy at Large Scale

• Again, downscale if possible
• Reduce the number of processes to which the debugger is attached
  – Reduces overhead
  – Reduces the required number of license seats as well
• Focus on one or a small number of processes/threads
  – Analyze call path and message queues to find problematic processes
  – Control the execution of as few processes/threads as possible while keeping others running
    • Provides the context where the error occurs
Trends in Debugging Technology

• Lightweight trace analysis tools
  – Help to identify processes/threads that have similar behavior and reduce the search space
  – Complementary to full feature debuggers
  – Example: Stack Trace Analysis Tool (STAT)

• Replay/Reverse execution
  – ReplayEngine now available from TotalView

• Post-mortem statistical analysis
  – Detect anomalies by analyzing profile dissimilarity of multiple runs
Outline

• Overview of parallel debugging
  – Challenges
  – Tools
  – Strategies

• Get familiar with TotalView through hands-on exercises
What Is TotalView

• A powerful debugger for both serial and parallel programs
  – Support Fortran, C/C++ and Assembler
  – Supported on most platforms
  – Both graphic and command line interface

• Features
  – Common debugging functions such as execution control and breakpoints
  – Memory debugging
  – Reverse debugging
  – Batch mode debugging
  – Remote debugging client
  – ...

7/6/2010 Scaling to Petascale Virtual Summer School
Three Ways to Start TotalView

- Start with core dumps
- Start by attaching to one or more running processes
- Start the executable within TotalView
User Interface - Root Window

<table>
<thead>
<tr>
<th>Status Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blank</td>
<td>Exited</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>
<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>
</tbody>
</table>

TotalView ID

MPI Rank

Host name

Status

7/6/2010 Scaling to Petascale Virtual Summer School
User Interface – 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
  - Manage action points, processes and threads
## Control Commands

<table>
<thead>
<tr>
<th>TotalView</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Go</td>
<td>Start/resume execution</td>
</tr>
<tr>
<td>Halt</td>
<td>Stop execution</td>
</tr>
<tr>
<td>Kill</td>
<td>Terminate the job</td>
</tr>
<tr>
<td>Restart</td>
<td>Restarts a running program</td>
</tr>
<tr>
<td>Next</td>
<td>Run to the next source line without stepping into another function</td>
</tr>
<tr>
<td>Step</td>
<td>Run to next source line</td>
</tr>
<tr>
<td>Out</td>
<td>Run to the completion of current function</td>
</tr>
<tr>
<td>Run to</td>
<td>Run to the indicated location</td>
</tr>
</tbody>
</table>
Controlling Execution

• The process window always focuses on one process/thread
• Switch between processes/threads
  – p+/p-, t+/t-, double click in root window, process/thread tab
• Need to set the appropriate scope when
  – Giving control commands
  – Setting action points
Process/Thread Groups

- Scope of commands and action points
  - 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
Types of Action Points

• **Breakpoints** stop the execution of the processes and threads that reach it

• **Evaluation points**: stop and execute a code fragment when reached
  – Useful when testing small patches

• **Process barrier points** synchronize a set of processes or threads

• **Watchpoints** monitor a location in memory and stop execution when its value changes
  – Unconditional
  – Conditional
Setting Action Points

• Breakpoints
  – Right click on a source line -> Set breakpoint
  – Click on the line number

• Watch points
  – Right click on a variable -> Create watchpoint

• Barrier points
  – Right click on a source line -> Set barrier

• Edit action point property
  – Right click on a action point in the Action Points tab -> Properties
Viewing/Editing Data

- View values and types of variables
  - At one process/thread
  - Across all processes/threads
- Edit variable value and type
- Array Data
  - Slicing
  - Filtering
  - Visualization
  - Statistics
Viewing Dynamic Arrays in C/C++

- Edit “type” in the variable window
- Tell TotalView how to access the memory from a starting location
- Example
  - To view an array of 100 integers
    - Change “Int *” to “int[100]*”
MPI Message Queues

• Detect
  – Deadlocks
  – Load balancing issues

• Tools -> Message Queue Graph
  – More options available
Call Graph

• Tools -> Call graph
• Quick view of program state
  – Nodes: functions
  – Edges: calls
• Look for outliers
Attaching to/Detaching from Processes

• You can
  – Attach to one or more running processes after launching TotalView
  – Launch the program within TotalView and detach from/reattach to any subset of processes later on
Memory Debugging

• Features
  – Memory usage report
  – Error detection
    • Memory leak
    • Dangling pointer
    • Memory corruption
  – Event notification
    •Deallocation/reallocation
  – Memory comparison between processes
Memory Debugging - Usage

• Need to link to the TotalView heap library to monitor heap status
  – The name of the library is platform dependent
• To access memory debugging functions
  – Prior to 8.7
    • Tools -> Memory debugging
  – Since 8.7
    • Debug -> Open MemoryScape
References

• TotalView user manual

• LLNL TotalView tutorial
  – https://computing.llnl.gov/tutorials/totalview

• NCSA Cyberinfrastructure Tutor
  – “Debugging Serial and Parallel Codes” course

• HPCBugBase
  – http://hpcbugbase.org/index.php/Main_Page
Hands-on Exercise

• Debug MPI and OpenMP programs that solve a simple problem to get familiar with
  – Basic functionalities of parallel debuggers
    • TotalView: BigRed, Kraken, Steele and Queen Bee
    • DDT: BigRed, Kraken, Ranger and Lonestar
  – Some common types of bugs in parallel programming

• Programs and instructions can be found at http://www.cct.lsu.edu/~lyan1/summerschool10
Problem

- A 1-D periodic array with \( N \) elements
- Initial value
  - C: \( \text{cell}(x) = x \% 10 \)
  - Fortran: \( \text{cell}(x) = \text{mod}(x-1,10) \)
- In each iteration, all elements are updated with the value of two adjacent elements:
  - \( \text{cell}(x)_{i+1} = [\text{cell}(x-1)_i + \text{cell}(x+1)_i] \% 10 \)

Execute \( N_{\text{iter}} \) iterations
The final outputs are the global maximum and average
Sequential Program

• Use an integer array to hold current values
• Use another integer array to hold the calculated values
• Swap the pointers at the end of each iteration
• The result is used to check the correctness of the parallel programs
  – Chances are that we will not have such a luxury for large jobs
MPI Program

• Divide the array among \( n \) processes
• Each process works on its local array
• Exchange boundary data with neighbor processes at the end of each iteration
• Ring topology
OpenMP Program

- Each thread works on its own part of the global array
- All threads have access to the entire array, so no data exchange is necessary