Techniques for Debugging HPC Applications

NIKOLAY PISKUN, DIRECTOR OF CONTINUING ENGINEERING, TOTALVIEW PRODUCTS

AUGUST 7 2019, ATRESC 2019
Agenda

- What is debugging and why TotalView?
- Overview of TotalView and TotalView’s new UI
- Advanced C++ and Data debugging
- MPI and OpenMP parallel debugging
- Reverse debugging
- Memory debugging
- GPU debugging
- Python/C++ debugging
- Using TotalView on ANL
- TotalView resources and documentation
- Questions/Comments
What is Debugging and Why do you need TotalView?
What is Debugging?

• Debugging is the process of finding and resolving defects or problems within a computer program or a system.
  • Algorithm correctness
  • Data correctness
  • Scaling/Porting correctness
TotalView debugger enables you to do:

• Interactive debugging
  • Live control of an executing program

• Remote debugging
  • Debug a program running on another computer

• Post-mortem debugging (core files and reverse debugging)
  • Debugging a program after it has crashed or exited

• Memory debugging
  • Find memory management problems (leaks, corruption ...)
  • Comparing results between executions

• Batch debugging (tvscript, CI environments)
  • Unattended debugging
TotalView for HPC and for All

- Leading debug environment for HPC users
  - Active development for 30+ years
  - Thread specific breakpoints
  - Control individual thread execution
  - View complex data types easily
  - From Mac to Top500 Supercomputers

- Track memory leaks in running applications
- Supports C/C++ and Fortran on Linux/Unix/Mac
- Support debugging mixed Python/C++
- Integrated Reverse debugging
- Batch non-interactive debugging.

- Allowing YOU to have
  - Predictable development schedules
  - Less time spent debugging
TotalView’s GUI
TotalView’s Classic UI

- Original powerful design
- Better tested for high-scale MPI jobs
- Assembler support
- Better supported for Remote Display Client

To use:
- Set UI preference
- Or command line argument
  ```
  totalview -oldUI
  ```
TotalView’s New UI (default)

- Provides a modern, dockable interface
- Easier to use, better workflows
- An architecture to grow
- To use:
  - Set UI preference
  - Or command line argument
    totalview -newUI

- New UI gaps:
  - Missing array slicing and striding, view across, data visualization
  - No very high-scale support
Intro & Starting Up
Start a Debugging Session

What would you like to debug?

- A new program
- A new parallel program
- A running program (attach)
- A core file

Help Manage Sessions Cancel
Start New Process – Arguments

Program Session

- Session Name: demoMpi
- File Name: /home/clemens/training/lab/programs/demompi
- Arguments: Hello World
- Debug on Host: localhost (local)

Press Start Session to start the debugging session.
Start New Process – Enable ReplayEngine
Start New Process – Memory Debugging

- Reverse Debugging
  - Step and debug in reverse from any point during execution
  - Enable reverse debugging with ReplayEngine

- Memory Debugging
  - Dynamically track memory allocations, catch memory errors and view memory analysis reports
  - Enable memory debugging
    - Suppress memory error notifications

- CUDA Debugging
  - Detect global memory addressing violations and unaligned memory accesses for CUDA based programs
  - Enable CUDA memory checking

Press Start Session to start the debugging session.
CUDA memory checking

Program Session

Reverse Debugging

Step and debug in reverse from any point during execution.
- Enable reverse debugging with ReplayEngine

Memory Debugging

Dynamically track memory allocations, catch memory errors and view memory analysis reports.
- Enable memory debugging
  - Suppress memory error notifications

CUDA Debugging

Detect global memory addressing violations and misaligned memory accesses for CUDA based programs.
- Enable CUDA memory checking

Press Start Session to start the debugging session.
Set environment variables

Enter environment variables to add to your program's environment:

```
EXE_HOME_DIR=/home/denouen/prog
```

Enter the environment variables that the process needs to run.

Press Start Session to start the debugging session.
Standard I/O redirection
Attach to Process

Start a Debugging Session

What would you like to debug?

- My last session: demoKipi
- A new program
- A new parallel program
- A running program (attach)
- A core file

Help  Manage Sessions  Cancel
Attach to Process
Attach to Process – Enable Replay Engine
Open a Core File
Open a Core File

![Core File Session](image)

Press Start Session to start the debugging session.
Process Control & Navigation
Interface Concepts

- State of all processes being debugged
- Process and Thread status
- Instant navigation access
- Sort and aggregate by status
Process Window Overview

Provides detailed state of one process, or a single thread within a process

A single point of control for the process and other related processes
Stepping Commands

Based on PC location
Action Points

Breakpoints
Barrier Points
Conditional Breakpoints
Evaluation Points
Watchpoints
Conditional Breakpoint

![Image of debug tool with conditional breakpoint](image-url)
Evaluation Point – Test Fixes on the Fly!

- Test small source code patches
- Call functions
- Set variables
- Test conditions
- C/C++ or Fortran
- Some limitations:
  - Can’t use C++ constructors
Watchpoints

Watchpoints are set on a fixed memory region

Use Tools > Watchpoint from a Variable Window or
From source pane with contextual menu

When the contents of watched memory change, the watchpoint is triggered and TotalView stops the program.

Watchpoints are not set on a variable. You need to be aware of the variable scope.
Watchpoints can be conditional or unconditional
Uses Hardware Watchpoints with various limitations based on architecture
Advanced C++ and Data Debugging
Advanced C++ and Data Debugging

- TotalView supports debugging the latest C++11/14 features including:
  - lambdas, transformations for smart pointers, auto types, R-Value references, range-based loops, strongly-typed enums, initializer lists, user defined literals

- TotalView transforms many of the C++ and STL containers such as:
  - array, forward_list, tuple, map, set, vector and others.
Array Slicing, Striding and Filtering (classic UI)

- **Slicing** – reduce display to a portion of the array
  - \([\text{lower}\_\text{bound}:\text{upper}\_\text{bound}]\)
  - \([5:10]\)

- **Striding** – Skip over elements
  - \([::\text{stride}]\)
  - \([::5], [5:10:-1]\)

- **Filtering**
  - Comparison: ==, !=, <, <=, >, >=
  - Range of values: \([>\] \text{low-value} : [<\] \text{high-value}\)
  - IEEE values: \$\text{nan}, \$\text{inf}, \$\text{denorm}\)
Array Statistics

- Easily display a set of statistics for the filtered portion of your array
Visualizing Array Data

- Visualizer creates graphic images of your program’s array data.
- Visualize one or two dimensional arrays
- View data manually through the Window > Visualize command on the Data Window
- Visualize data programmatically using the $visualize function
Dive in All

- Dive in All
  - Use Dive in All to easily see each member of a data structure from an array of structures
Looking at Variables Across Processes

- TotalView allows you to look at the value of a variable in all MPI processes
  - Right Click on the variable
  - Select the View > View Across
- TotalView creates an array indexed by process
  - You can filter and visualize
  - Use for viewing distributed arrays as well.
  - You can also View Across Threads
Multi-Thread and Multi-Process Parallel Debugging
In the Parallel Program Session select:

Select:
- MPI preference
- number of tasks
- number of nodes
- starter arguments

... then save all this in Session
Stepping Commands
Message Queue Graph

- Hangs & Deadlocks
- Pending Messages
  - Receives
  - Sends
  - Unexpected
- Inspect
  - Individual entries
- Patterns
Find Deadlocks and Performance Sinks

- Filtering
  - Choose messages to track
  - Choose MPI Communicators
- Cycle detection
Reverse Debugging
Reverse Debugging

Replay Engine – The right way to debug

- Step forward over functions
- Step forward into functions
- Advance forward out of current Function, after the call
- Advance forward to selected line
- Run forward
- Advance forward to “live” session
- Step backward over functions
- Step backward into functions
- Advance backward out of current Function, to before the call
- Advance backward to selected line
- Run backward
ReplayEngine

- Captures execution history
  - Records all external input to program
  - Records internal sources of non-determinism
- Replays execution history
  - Examine any part of the execution history
  - Step back as easily as forward
  - Jump to points of interest
- An add-on product to TotalView
  - Support for
    - Linux/x86
    - Linux x86-64
Memory Debugging
Memory Debugging

- TotalView’s memory debugging technology allows you to
  - Easily find memory leaks and other memory errors
  - Detect malloc/free new/delete API misuse
  - Dangling pointer detection
  - Detect buffer overruns
  - Paint memory blocks on allocation and deallocation
- Memory debugging results can be easily shared as
  - HTML reports or raw memory debugging files.
  - Compare memory results between runs to verify elimination of leaks
  - Supports parallel applications
  - Low overhead and does not require recompilation or instrumentation
Strategies for Parallel Memory Debugging

- Run the application and see if memory events are detected
- View memory usage across the MPI job
  - Compare memory footprint of the processes
    - Are there any outliers? Are they expected?
- Gather heap information in all processes of the MPI job
  - Select and examine individually
    - Look at the allocation pattern. Does it make sense?
    - Look for leaks
  - Compare with the ‘diff’ mechanism
    - Are there any major differences? Are they expected?
GPU Debugging
GPU debugging with TotalView

• NVIDIA CUDA support
  – Multiple platforms: X86-64, PowerLE, ARM64
  – Multiple cards: from Jetson to Turing

• Features and capabilities include
  – Support for dynamic parallelism
  – Support for MPI based clusters and multi-card configurations
  – Flexible Display and Navigation on the CUDA device
    • Physical (device, SM, Warp, Lane)
    • Logical (Grid, Block) tuples
  – CUDA device window reveals what is running where
  – Support for CUDA Core debugging
  – Leverages CUDA memcheck
  – Support for OpenACC
GPU Debugging Model Improvements

- First in class Unified Source debugging
- Improves and streamlines debugging CUDA applications
  - Set breakpoints in CPU and GPU kernel code before it is launched on the GPU
  - Compare variables in CPU and GPU code together
CUDA Debugging Demo
Extending Debugging Capabilities: How to Debug (AI) Mixed Python/C++ Code
Debugging multiple languages

- Debugging one language is difficult enough
  - Especially with many threads/processes

- The language intersection is tougher
  - Data comparison
  - Glue code

- Issues are:
  - Type mismatches
  - Extraneous stack frames
Python debugging with TotalView (New GUI only)

• What TotalView provides:
  • Easy Python debugging session setup
  • Fully integrated Python and C/C++ call stack
    • "Glue" layers between the languages removed
  • Easily examine and compare variables in Python and C++
  • Utilize reverse debugging and memory debugging

• What TotalView does not provide (yet):
  • Setting breakpoints and stepping within Python code
#!/usr/bin/python

def callFact():
    import tv_python_example as tp
    a = 3
    b = 10
    c = a+b
    ch = "local string"
    ......
    return tp.fact(a)
if __name__ == '__main__':
    b = 2
    result = callFact()
    print result
Python without special debugger support

No viewing of Python data and code
Showing C code with mixed data

• Glue code filtered out
• Python data and code available for viewing
Remote Display Debugging
Remote Display Client (RDC)

• Offers users the ability to easily set up and operate a TotalView debug session that is running on another system
• Consists of two components
  • Client – runs on local machine
  • Server – runs on any system supported by TotalView and “invisibly” manages the secure connection between host and client
• Free to install on as many clients as needed
• Remote Display Client is available for:
  • Linux x86, x86-64
  • Windows
  • Mac OS X
Remote Display Client
Remote Display Client (Argonne NL)
Summary

• Use of modern debugger **saves** you time.

• TotalView can help you because:
  • It’s **cross-platform** (the only debugger you ever need)
  • Allow you to debug accelerators (GPU) and CPU in **one session**
  • Allow you to debug **multiple languages** (C++/Python/Fortran)
Using TotalView for Parallel Debugging on ANL
Starting a MPI job – method 1

For HPC we have two methods to start the debugger

The ‘classic’ method

- `totalview -args mpiexec -np 512 ./myMPIprog myarg1 myarg2`
- This will start up TotalView on the parallel starter (mpiexec, srun, runjob, etc) and when you hit ‘Go’ the job will start up and the processes will be automatically attached. At that point you will see your source and can set breakpoints.

- Some points to consider...
  - You don’t see your source at first, since we’re ‘debugging’ the mpi starter
  - Some MPI’s don’t support the process acquisition method (most do, but might be stripped of symbols we need when packaging)
  - In general more scalable than the next method...
Starting a MPI job – method 2

The ‘indirect’ method

- Simply ‘totalview’ or ‘totalview myMPIprog’ and then you can choose a parallel system, number of tasks, nodes, and arguments to the program.

- With this method the program source is available immediately
- Less dependent on MPI starter symbols
- May not be as scalable as some ‘indirect’ methods launch a debug server per process
Using TotalView at Argonne

- TotalView available on Theta, Vesta, Mira, Cooley
  - Installed at: /soft/debuggers/totalview-2019-08-01/toolworks/totalview.2019T.2.7/bin/totalview
  - module load totalview
- Download and install RDC from https://www.roguewave.com/products-services/features/remote-display-client
- Connect to Theta
- Get allocation first
  - On Theta: qsub -A ATPESC2019 -n <N> -q debug-flat-quad -l
  - Module load totalview
  - totalview -args aprun -np <N> ....
TotalView Resources and Documentation
TotalView Resources & Documentation

• TotalView documentation:
  • User Guides: Debugging, Memory Debugging and Reverse Debugging
  • Reference Guides: Using the CLI, Transformations, Running TotalView

• TotalView online HTML doc:

• Other Resources (Blogs, videos, white papers, etc):
  • [https://www.roguewave.com/resources?tagid=18](https://www.roguewave.com/resources?tagid=18)

• New UI resources:
  • Reference CodeDynamics Help
    [https://www.roguewave.com/help-support/documentation/codedynamics](https://www.roguewave.com/help-support/documentation/codedynamics)

• New UI videos:
  • [https://www.roguewave.com/products-services/codedynamics/videos](https://www.roguewave.com/products-services/codedynamics/videos)

• Python Debugging blog:
Questions/Comments

• Any questions or comments?
  • Don’t hesitate to reach out to me directly with any problems or suggestions!
  • Email: nikolay.piskun@roguewave.com

• Thank you for your time today!
THANK YOU