Reverse Debugging with The TotalView Debugger

Chris Gottbrath
Product Manager

Outline

- **Introduction**
  - Why is Troubleshooting so Frustrating?
  - Why Reverse Debugging?
  - Record and Replay
- **TotalView Debugger**
  - Usability and Interface Concepts
  - Visualization and Scalability
  - Status and Projects
- **The Reverse Debugging Add-on Module**
  - Architecture
  - Usage Model
  - Future Directions
- **Conclusions**
  - Feedback on Reverse Debugging
  - Early Experience Program

TOTALVIEW TECHNOLOGIES

5/2/2008

Thoughts about Troubleshooting

- **Why is it so unpleasant?**
- **Separation between error and result**
  - Crashing, running slow, giving bad results just a symptom
  - What is the root cause?
- **Time's arrow**
  - Working backwards from effect to cause
  - Detective work, based on clues and deduction
- **Cyclic process**
  - Run it again and watch how it got into that state
  - Is the problem easy to reproduce?
  - Is it easy to get to the point you want to be in the program?
- **Where is the effort and attention?**
  - Reproducing the error
    - Tedious
    - Distracting
  - Actually solving the error
    - Satisfying

Why Reverse Debugging?

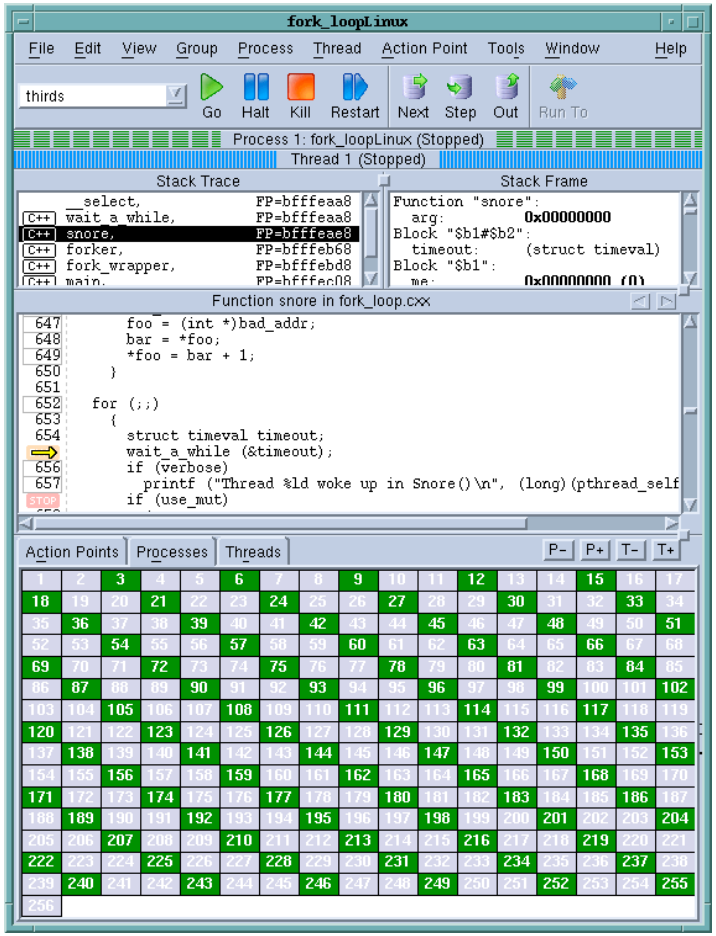
- **That's the direction you need to go**
  - From the effect
  - To the cause
- **Drastically reduces the effort**
  - No need to run over again from the beginning
    - Carefully providing input
    - Controlling concurrency
- **Lets you focus**
  - Less distraction
  - No chance that you'll miss something and not be able to get it back
    - Race conditions

Record and Replay

- **Full history of process available for review**
- **Capture execution history**
  - Record all external input
    - Reading from files, network
    - Time()
  - Record “other” sources of non-determinism
    - Thread context switches
- **Replay execution**
  - Simulate the execution along exactly the same trajectory
  - Can ‘reset’ to any point along the path already taken
- **Everything is managed by the tool**
  - The user just says where they want to go
    - Back one line
  - There is some overhead
    - Lots of clever tricks to reduce overhead

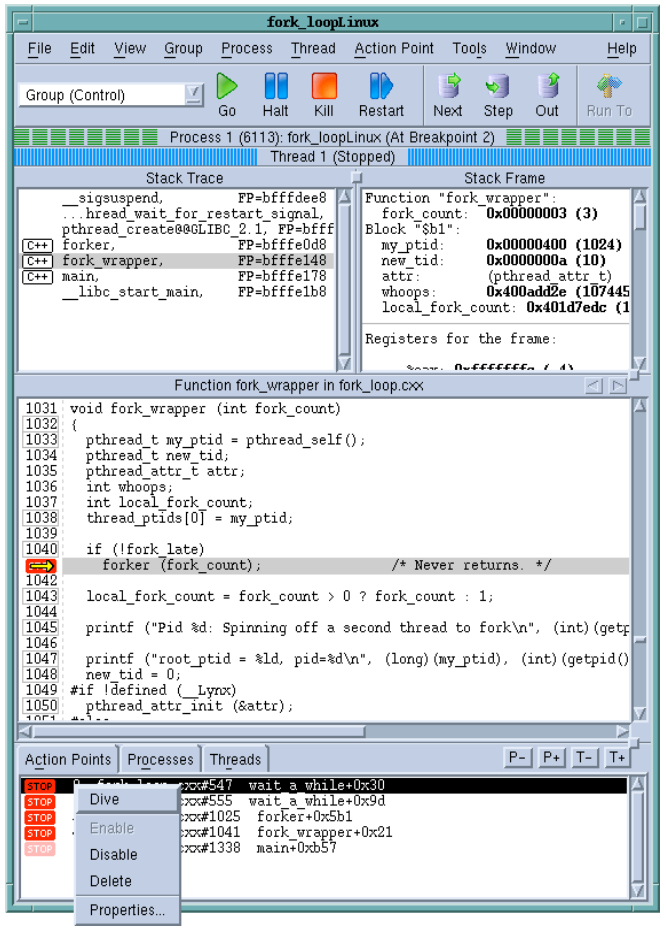
TotalView Source Code Debugger

- **C, C++, Fortran 77, Fortran90, UPC**
  - Complex language features
- **Wide Compiler and Platform Support**
  - Supports Cray XT3/4
    - Catamount and CNL
    - Catamount licenses can be migrated (no charge), contact TotalView Technologies
  - Supports Cray X1 and X2
  - Supports many other platforms
- **Parallel & Multi-threaded Debugging**
  - MPI, UPC
  - OpenMP
- **Memory Debugging Capabilities**
  - Integrated into the debugger
- **Powerful and Easy GUI**
- **Visualization**
- **CLI for Scripting**



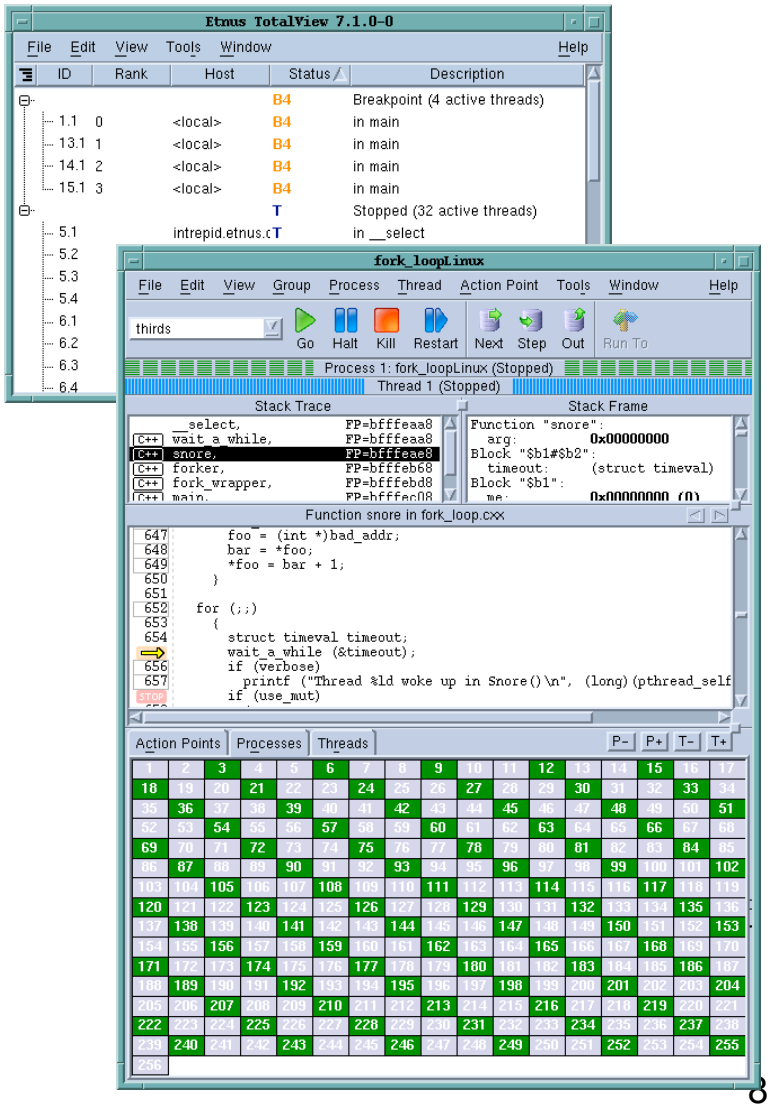
Usability and Productivity

- **Most used operations should be clear**
  - Conform to basic expectations
  - Consistency
- **Attention to users and use cases**
  - Support to overcome unexpected challenges
    - If it doesn't work then the user is not going to be productive
  - Focus on things that improve productivity
    - Collaboration
    - Visualization
- **Depth of functionality**
  - Extensibility, script-ability
  - Scalability
    - Time to perform operations
    - Presentation of information
    - Many input parameters beyond 'N'



Interface Concepts

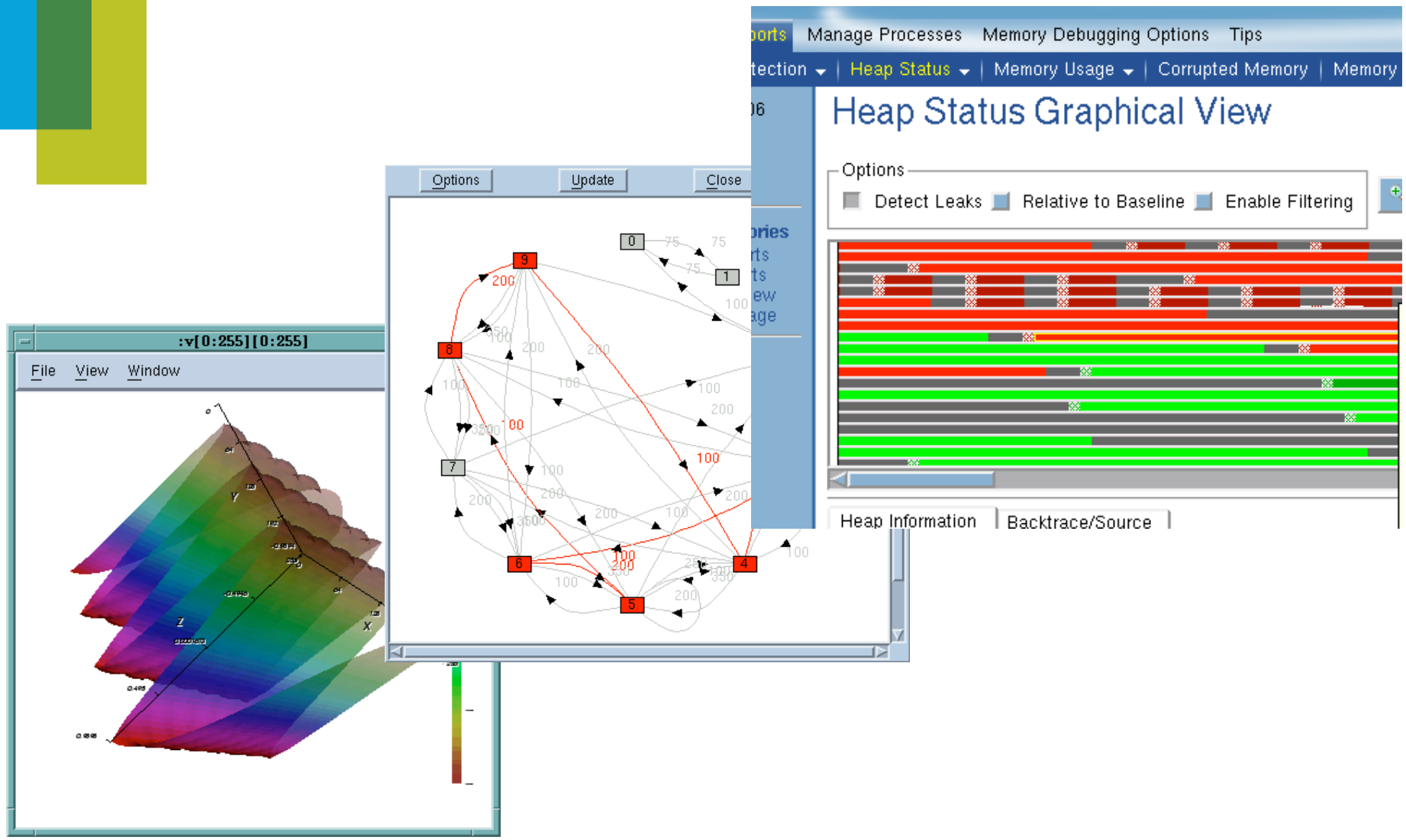
- **Root Window**
  - State of all processes being debugged
- **Process Window**
  - Detailed state of a single process
    - Thread within a process
  - Point of control
    - Control the process and possibly other related processes



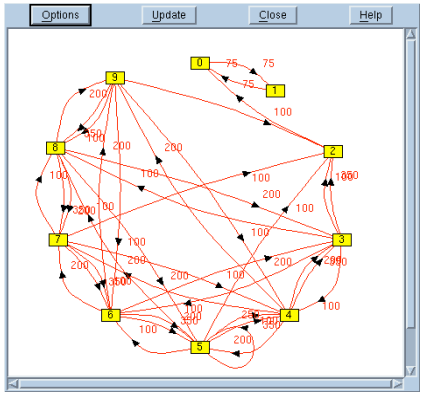
Visualizing Program Data and State

TOTALVIEW TECHNOLOGIES

5/2/2008



The Scalability of TotalView Debugger



- **Techniques for using TotalView at scale, e.g.**
  - Subset attach, message queue display, cycle detection, call graph, view data across processes and threads, sorting on the root window, etc.
- **Current scalability (tested and verified)**
  - Debug one to thousands of processes regularly
  - Almost all operations at 2,000 take less than a few seconds
  - Higher scale, depending on the system and application
    - Cray XT: up to 6,000 processes
    - Linux cluster: up to 6,000 processes
    - Blue Gene: up to 32,000 processes
- **Actively working on performance and scalability**
  - Improvements come from rigorous profiling and timing
- **Systematic measurement of TotalView's performance**
  - Taking an objective, systematic approach to characterizing and measuring TotalView's performance at scale
  - Performance regression testing at scale
- **Scalability Partnership Program**
  - Ask TotalView Technologies

Recent and Current

- **TotalView 8.4 and MemoryScape 2.2**
  - IPv6 support
  - Lightweight memory corefile debugging
    - For very large scale memory debugging
  - Scalability improvements
- **Cray XT**
  - Recent Scalability improvements
    - Parallel Launch
    - Variable Viewing
  - CNL memory support (new)
  - Subset attach on Cray XT CNL (coming soon)
- **Other platform news**
  - BlueGene / P platform support
    - Currently available
    - Threads and libraries should be ready for test soon
  - Linux-Cell platform support
    - Currently available for testing

Product Plans

- **TotalView Debugger**
  - Fall release
    - More scalability improvements
    - Long distance remote debugging
      - Easier session setup
      - Faster graphics performance
    - Batch debugging script framework
- **Reverse Debugging Add-on**
  - Fall release
    - Step backwards from crashes
    - Support for x86 and x86-64 Linux

Reverse Debugging Add-on Module

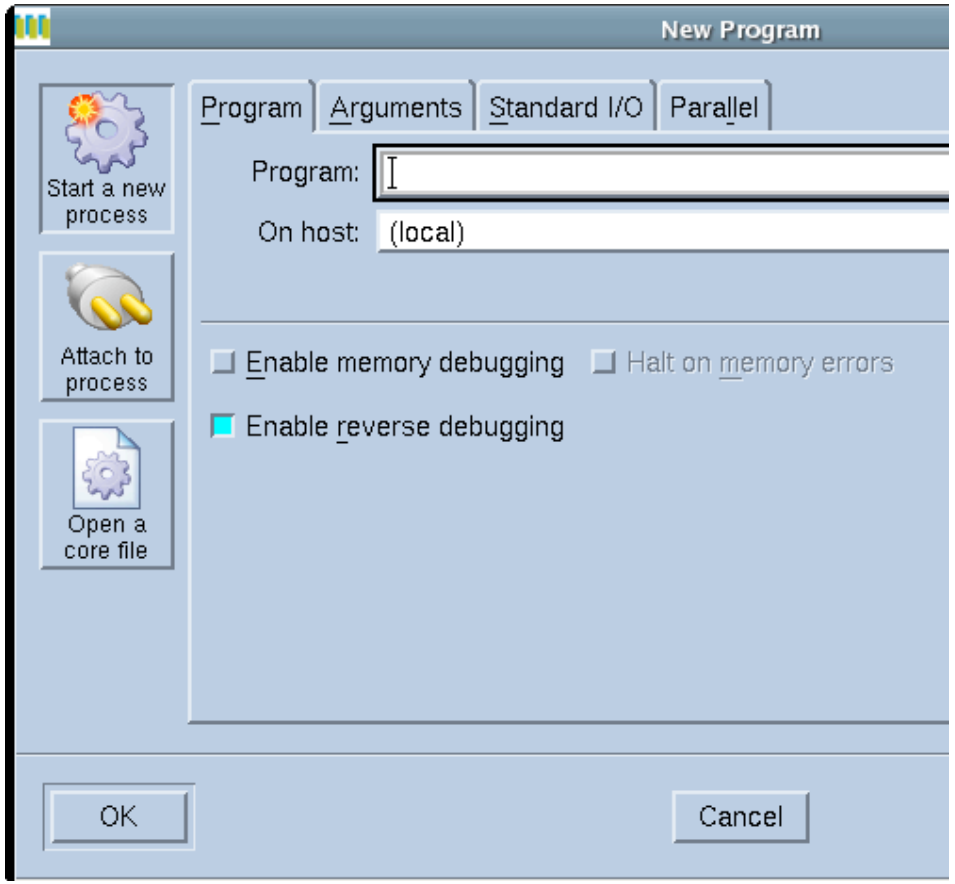
- **What is it?**
  - A separately licensed add-on to TotalView Debugger
  - Based on “record and deterministic replay”
  - Provides users with the ability to review any part of the program execution from the beginning of the run to the current time
  - Similar to adding a rewind button on a DVR
  - There is some overhead
  - Initial support for Linux-x86 32-bit and 64-bit only
- **What will it provide?**
  - Simplification of troubleshooting process
    - No need to do many restarts and drive the application forward
  - Determinism within a debugging session
  - Provides access to the whole program execution sequence rather than just a single “slice” in time

Reverse Debugging Architecture

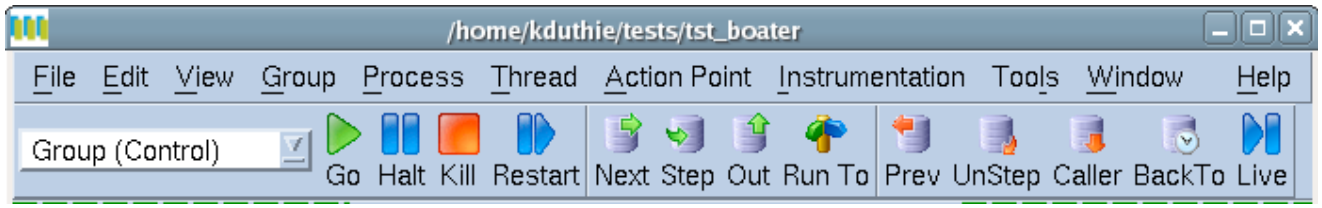
- **Debugger**
  - New concepts
    - Execution time-like metric
    - "Record mode" vs. "replay mode"
      - Some operations can't be allowed during replay
  - New commands
    - Backwards stepping
    - Jumping into history
    - Returning to "live"
- **Instrumentation Library**
  - Preloaded by the debugger
    - Into the target program
  - Handles
    - Recording program state
    - Replaying program state
  - Overhead
    - Execution time
    - Memory usage
    - Writes some data out to a file

Enabling Reverse Debugging

- **Enable before starting the program**
  - A checkbox on the new program dialog box
  - Command line flag
    - totalview -reverse_debugging
  - No recompilation or instrumentation step
  - No reason users can't do it with optimized code
- **Enables the backwards stepping buttons in the GUI**

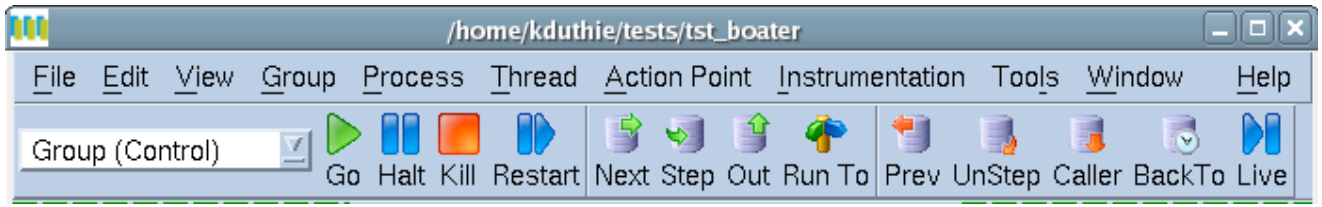


Reverse Stepping



- **Next Backwards "Prev"**
  - Goes over functions
  - Users can step back around a loop
  - Users can step back out of functions
    - Even main() … you end up in the runtime loader.
- **Step Backwards "UnStep"**
  - Goes into functions
- **Out Backwards "Caller"**
  - Returns to caller -- before the call was made

Working with History



- **Run to Backwards "Back To"**
  - Select a line in the source
  - Runs to most recent time that line was executed
    - More complex with multi-addressed lines as in templates
- **Return to Live "Live"**
  - Returns to the "current" time and to record mode
- **Random Access**
  - CLI only
  - Query to get current point in execution history
    - A numeric value
  - Run to any point in execution history

Feedback

- How does reverse debugging sound?
- Do you need/want it on the Cray XT series?
- Please feel free to talk to me after the session
  - Or via email: Chris.Gottbrath@totalviewtech.com
  - We are taking names for the beta program!

Early Experience Program

- **Early view and input into product development**
  - Use cases
  - Usability feedback
  - Detailed product direction input
- **Four Tracks**
  - TracePoint Data Centric Debugger
    - Should this be a stand alone tool?
  - Reverse Debugging
    - How to represent time in the GUI?
  - TAU Performance Tool Integration
    - Most important use cases?
  - Advanced Workbench
    - What else should we integrate?

End of Presentation

- **Thanks!**

5/2/2008