TotalView Workshop
SARA 2010

Nikolay Piskun
Director of Continuing Engineering

Royd Lüdtke
Application Engineer

Workshop Agenda

9:45 Welcome & Reception with coffee

10.00 Introduction to Rogue Wave and TotalView

10.45 Head on Labs I

- Debugger Basics
- Viewing, Examining, Watching and Editing
- Examining and Controlling a Parallel Application

12.15 Lunch Break

13.00 Advanced debugging topics

14.15 Advanced Labs

- Exploring Heap Memory in MPI applications
- Batch Mode Debugging with TVScript
- Reverse Debugging with ReplayEngine

16.15 Debugging user code on user machines (optional)

17.00 The End

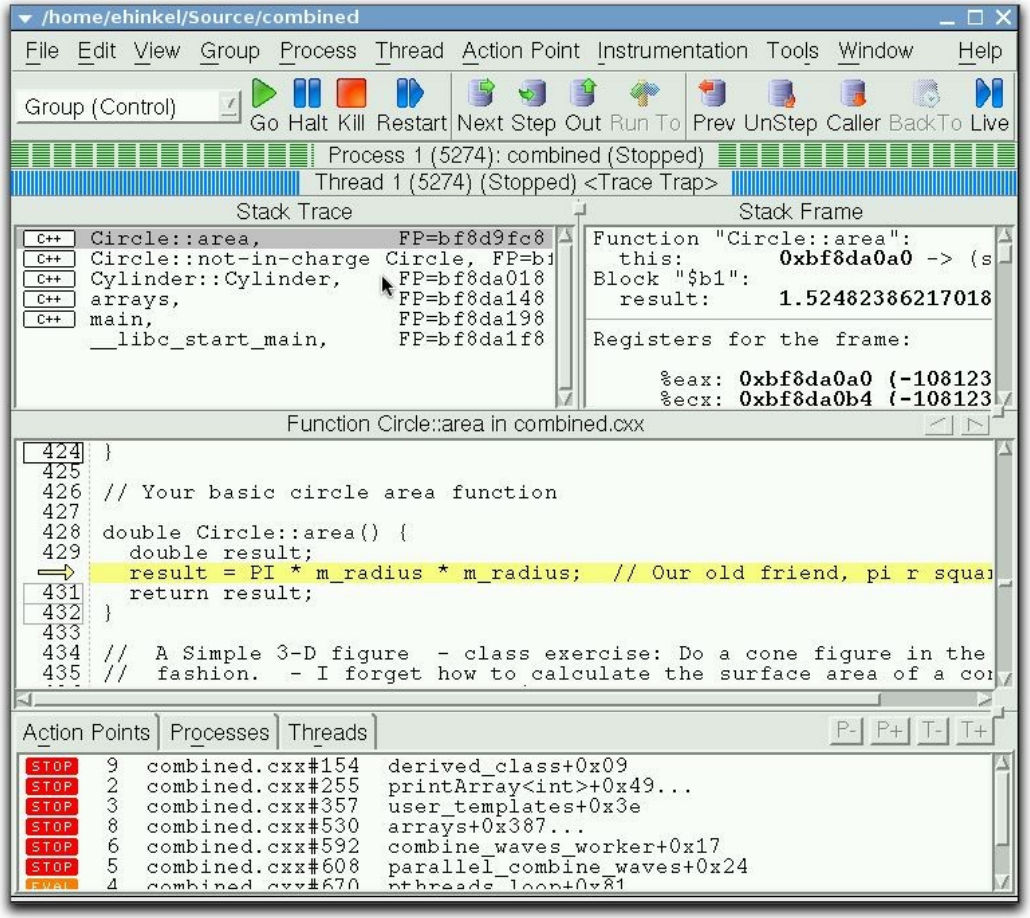
Introduction to Totalview Tech Products

- Introduction
- Totalview Basics
- Parallel Debugging

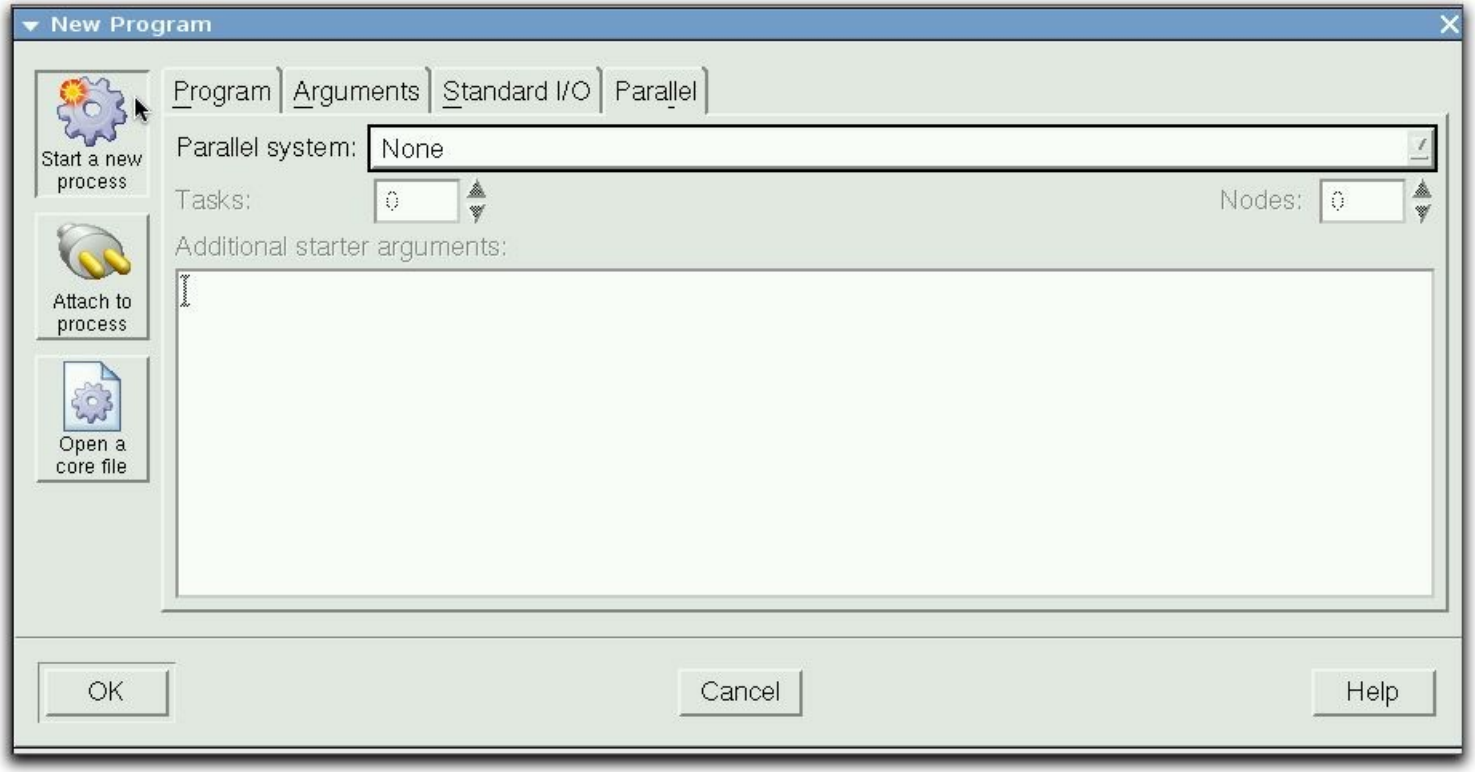
What is TotalView?

A comprehensive debugging solution for demanding parallel and multi-core applications

- Wide compiler & platform support
  - C, C++, Fortran 77 & 90, UPC
  - Unix, Linux, OS X
- Handles Concurrency
  - Multi-threaded Debugging
  - Parallel Debugging
    - MPI, PVM, Others
  - Remote and Client/Server Debugging
- Integrated Memory Debugging
- Reverse Debugging available
  - ReplayEngine add on
- Supports a Variety of Usage Models
  - Powerful and Easy GUI
    - Visualization
  - CLI for Scripting
  - Long Distance Remote Debugging
  - Unattended Batch Debugging



Starting TotalView



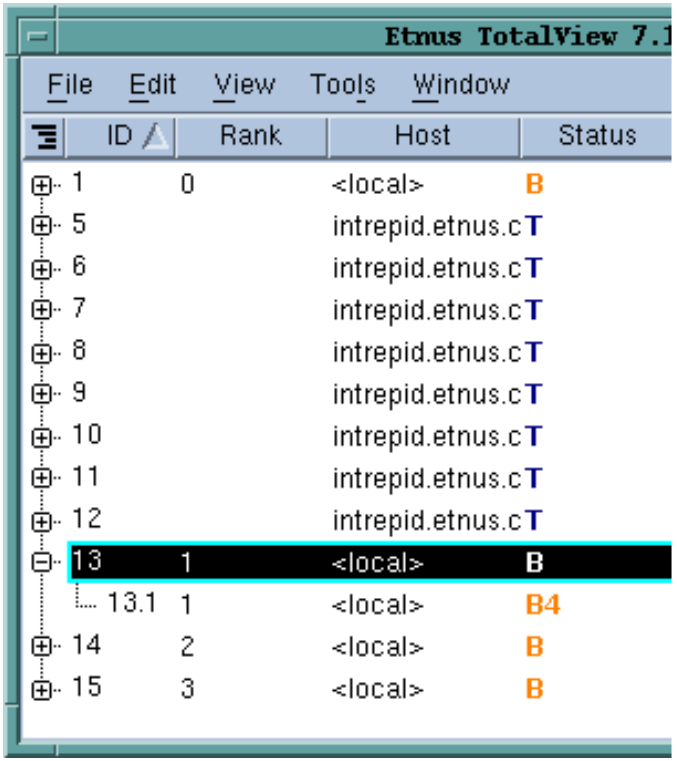
Open a Core File

Process Control & Navigation

Interface Concepts

Root Window

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



Status Info

- T = stopped
- B = Breakpoint
- E = Error
- W = Watchpoint
- R = Running
- M = Mixed
- H = Held

Process Window Overview

Stack Trace Pane

Toolbar

Stack Frame Pane

Source Pane

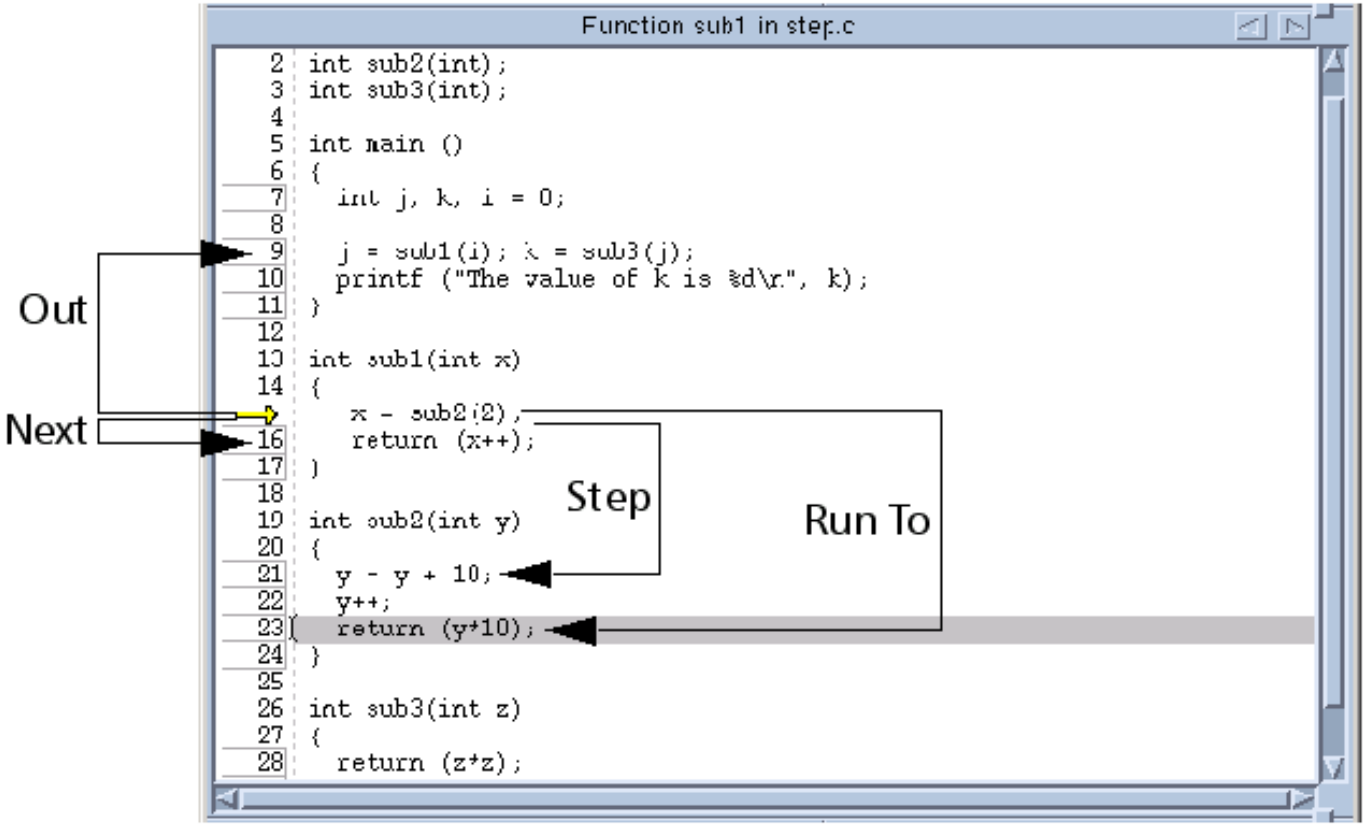
Tabbed Area

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

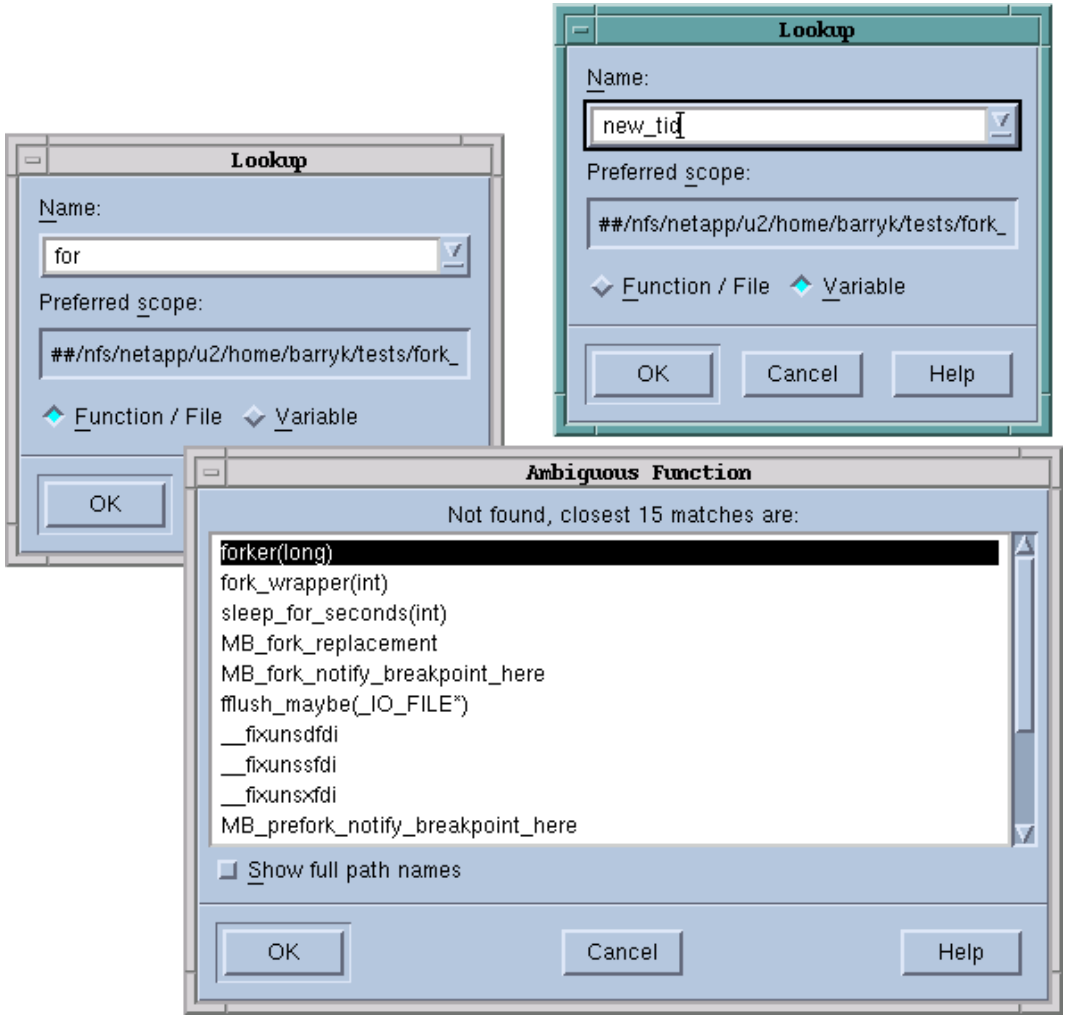
A single point of control for the process and other related processes

Stepping Commands

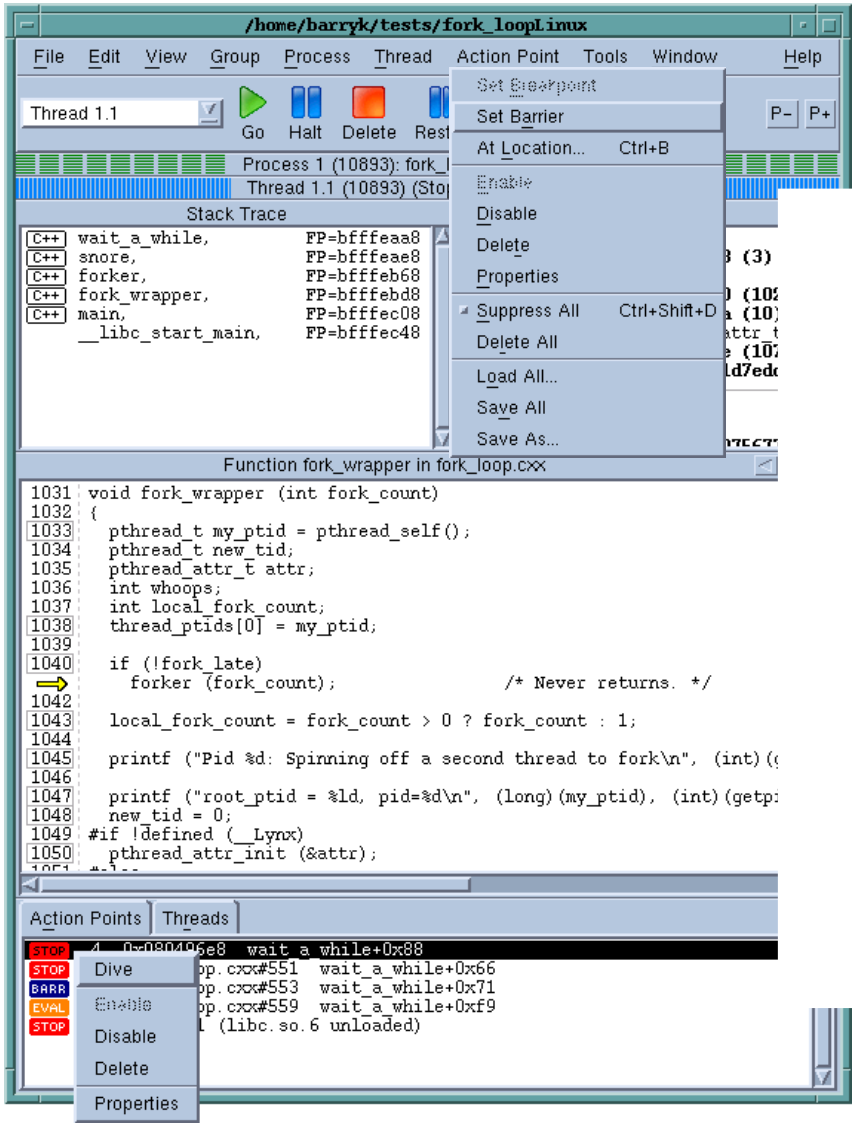
Based on PC location



Finding Functions, Variables, and Source Files



Action Points



Breakpoints

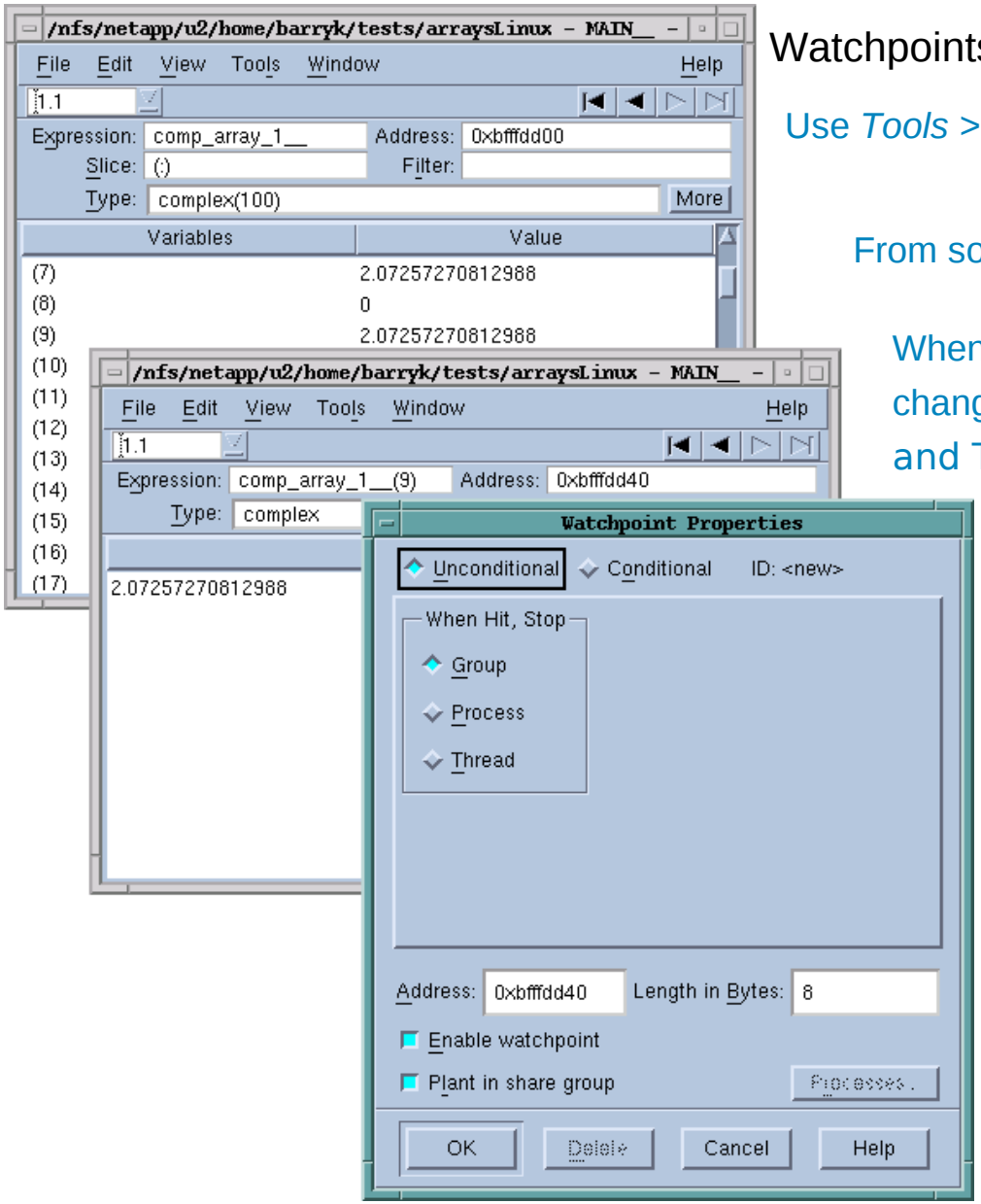
Barrier Points

Conditional Breakpoints

Evaluation Points

Watchpoints

Watchpoints



Watchpoints are set on a fixed memory region

Use *Tools > Watchpoint* from a Variable Window

or

From source pane with contextual menu

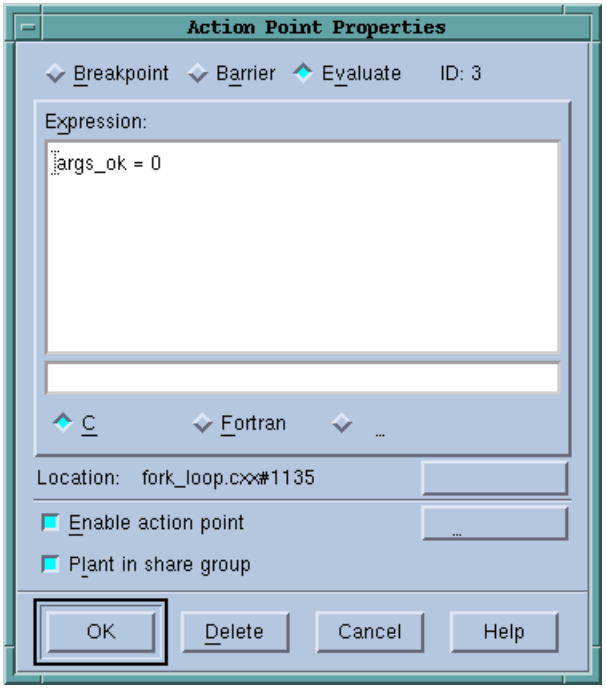
When the contents of watched memory change, the watchpoint is triggered and TotalView stops the program.

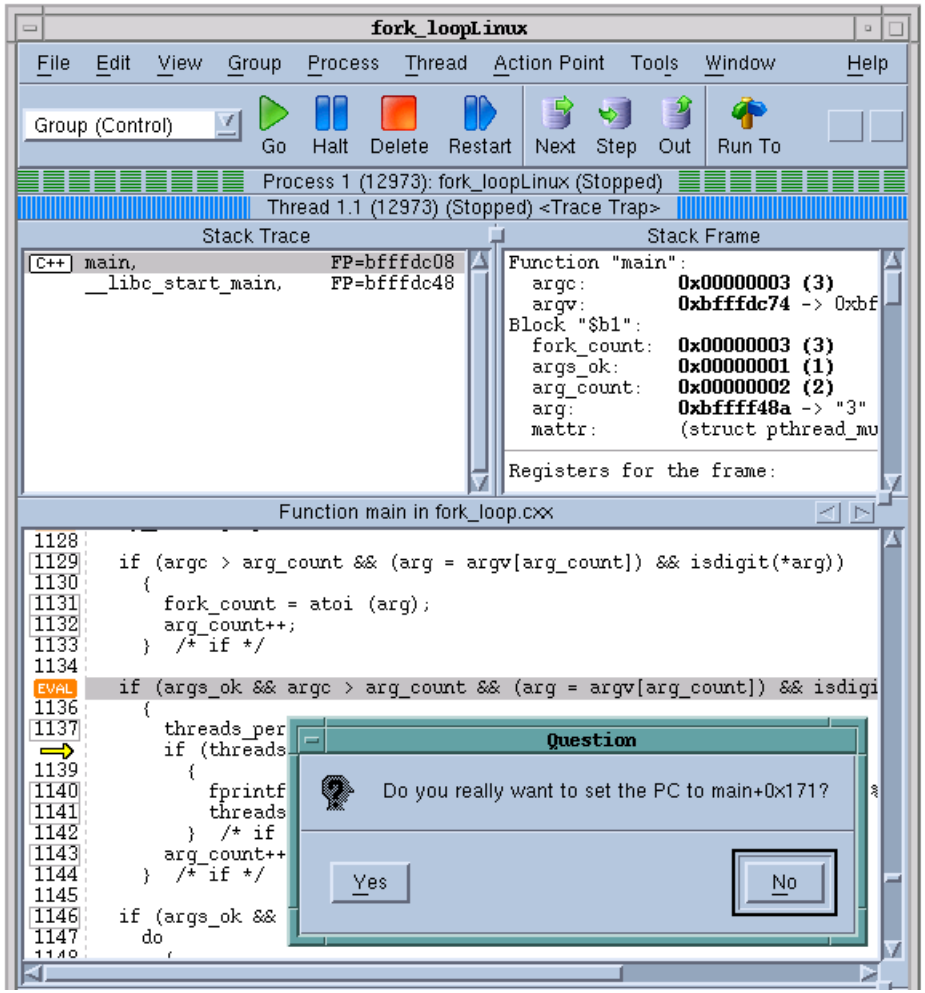
Watchpoints are not set on a variable. You you need to be aware of the variable scope.

Watchpoints can be conditional or unconditional

Uses Hardware Watchpoints with various limitations based on architecture

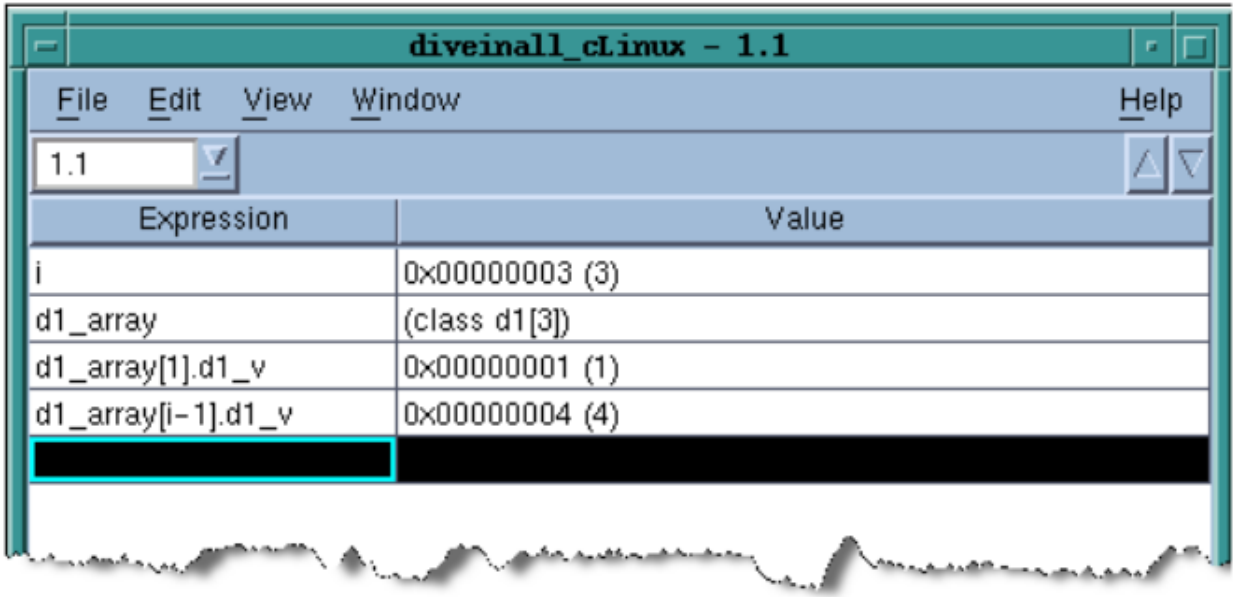
Using Set PC to resume execution at an arbitrary point





Viewing and Editing Data

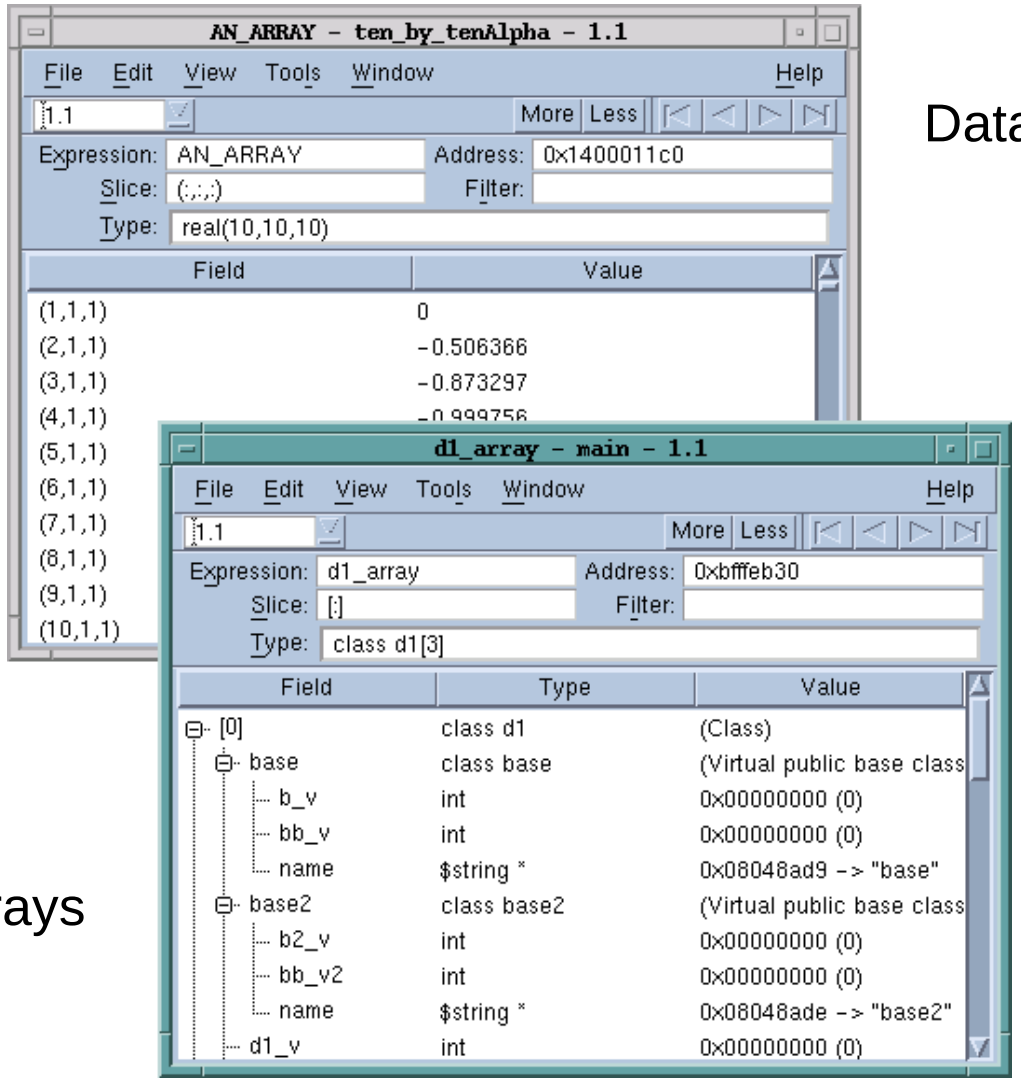
Expression List Window



Add to the expression list using contextual menu with right-click on a variable, or by typing an expression directly in the window

- Reorder, delete, add
- Sort the expressions
- Edit expressions in place
- Dive to get more info
- Updated automatically
- Expression-based
- Simple values/expressions
- View just the values you want to monitor

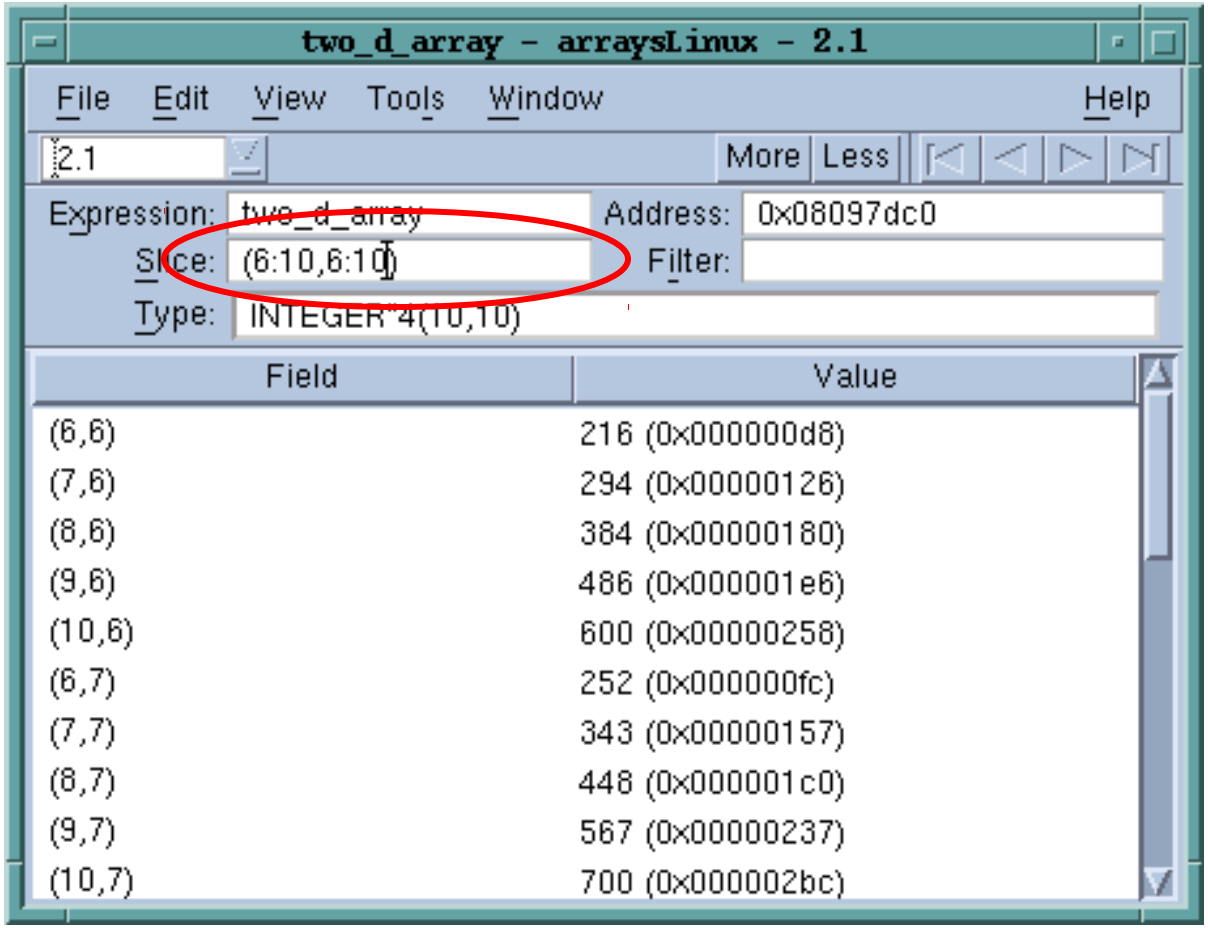
Viewing Arrays



Data Arrays

Structure Arrays

Slicing Arrays



Slice notation is `[start:end:stride]`

Filtering Arrays

ieee_array - /nfs/netapp0/user/home/barryk/tests/arraysAlpha - 3.

File Edit View Tools Window Help

3.1

Expression: ieee_array Address: 0x1408214a0

Slice: (:) Filter: .eq. $inf

Type: $real_4(6)

Field	Value
(1)	INF
(2)	-INF

ieee_array - /nfs/netapp0/user/home/barryk/tests/arraysAlpha - 3.

File Edit View Tools Window Help

3.1

Expression: ieee_array Address: 0x1408214a0

Slice: (:) Filter: .eq. $denorm

Type: $real_4(6)

Field	Value
(5)	1.4013e-45 <denormalized>
(6)	-1.4013e-45 <denormalized>

ieee_array - /nfs/netapp0/user/home/barryk/tests/arraysAlpha - 3

File Edit View Tools Window

3.1

Expression: ieee_array Address: 0x1408214a0

Slice: (:) Filter:

Type: $real_4(6)

Field	Value
(1)	INF
(2)	-INF
(3)	NaNQ
(4)	NaNS
(5)	1.4013e-45 <denormalized>
(6)	-1.4013e-45 <denormalized>

int2_array__ - MAIN__ - 1.1

File Edit View Tools Window Help

1.1

Expression: int2_array__ Address: 0xbfffd450

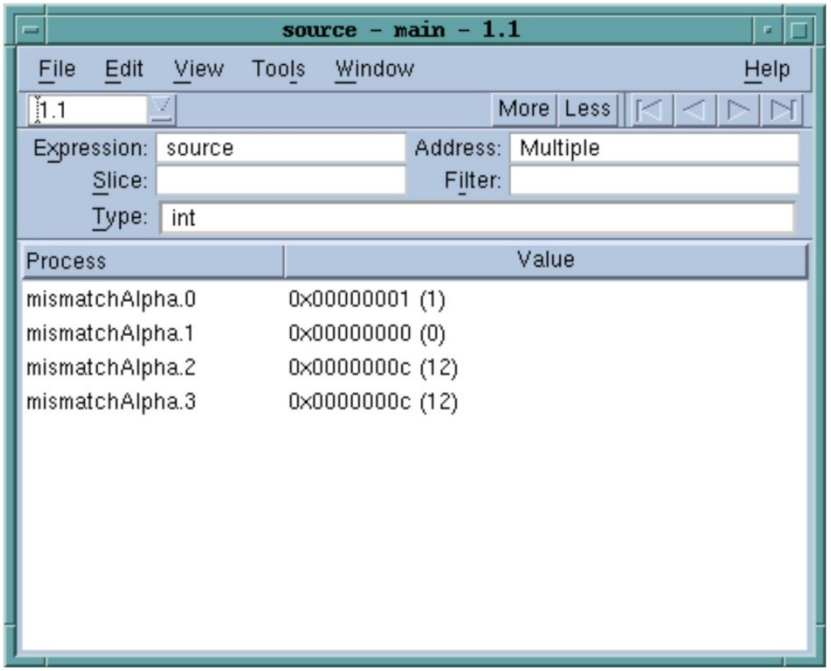
Slice: (:) Filter: $value > 20 .and. $value < 100

Type: word(100) More

Field	Value
(16)	22 (0x0016)
(17)	24 (0x0018)
(18)	26 (0x001a)
(19)	28 (0x001c)
(20)	30 (0x001e)
(21)	32 (0x0020)
(22)	34 (0x0022)
(23)	36 (0x0024)
(24)	38 (0x0026)
(25)	40 (0x0028)

Looking at Variables across Processes

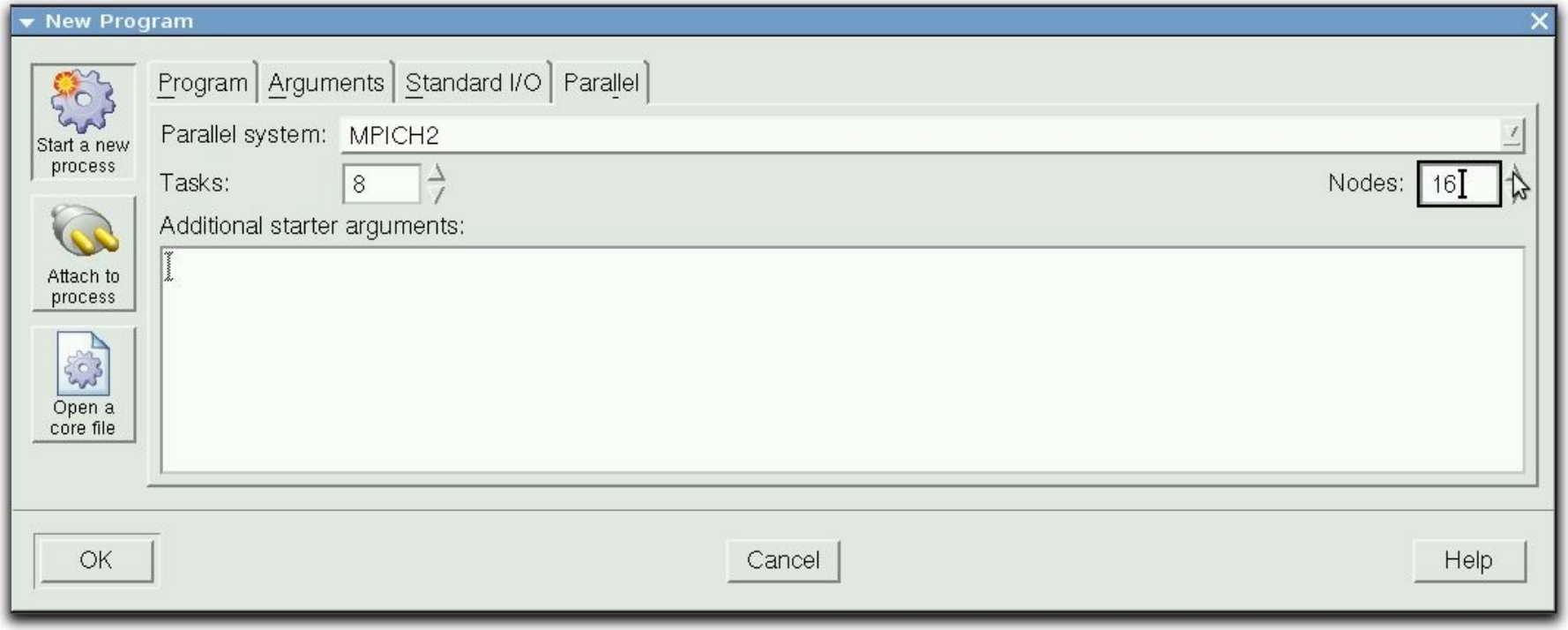
- **TotalView allows you to look at the value of a variable in all MPI processes**
  - Right Click on the variable
  - Select the View > View Across
- **TotalView creates an array indexed by process**
- **You can filter and visualize**
- **Use for viewing distributed arrays as well.**
- **You can also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



Parallel Debugging

TotalView Startup with MPI
TVT Launch

In the Parallel tab, select:

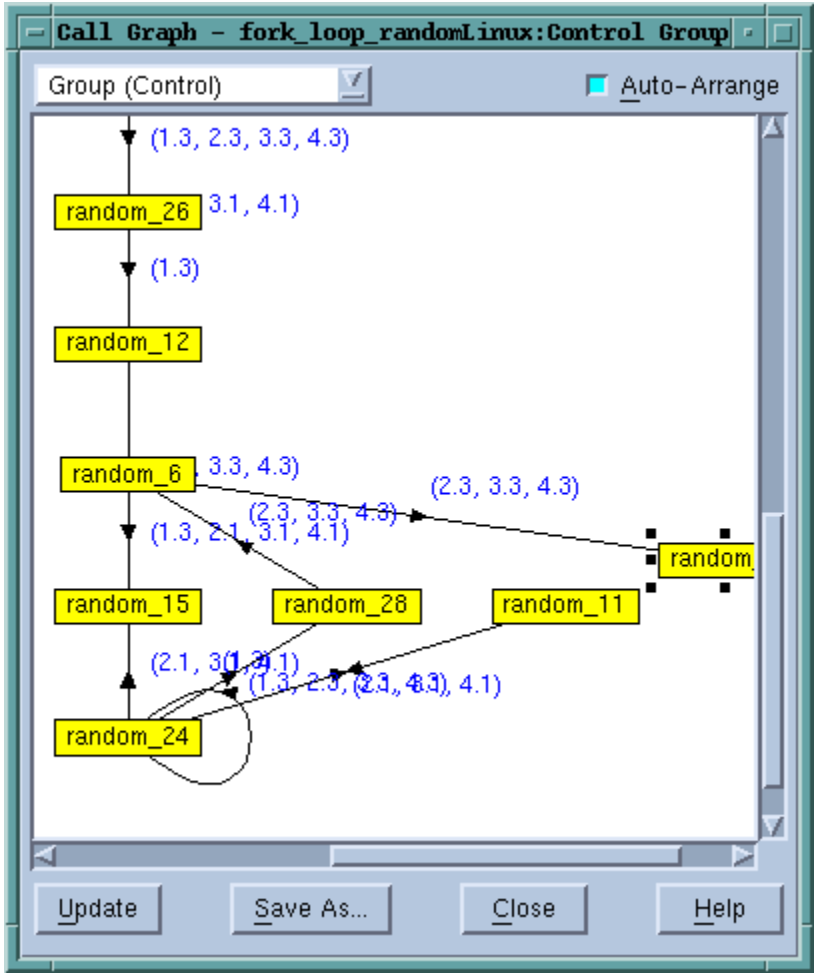


your MPI preference, number of tasks, and number of nodes.

… then add any additional starter arguments

Call Graph

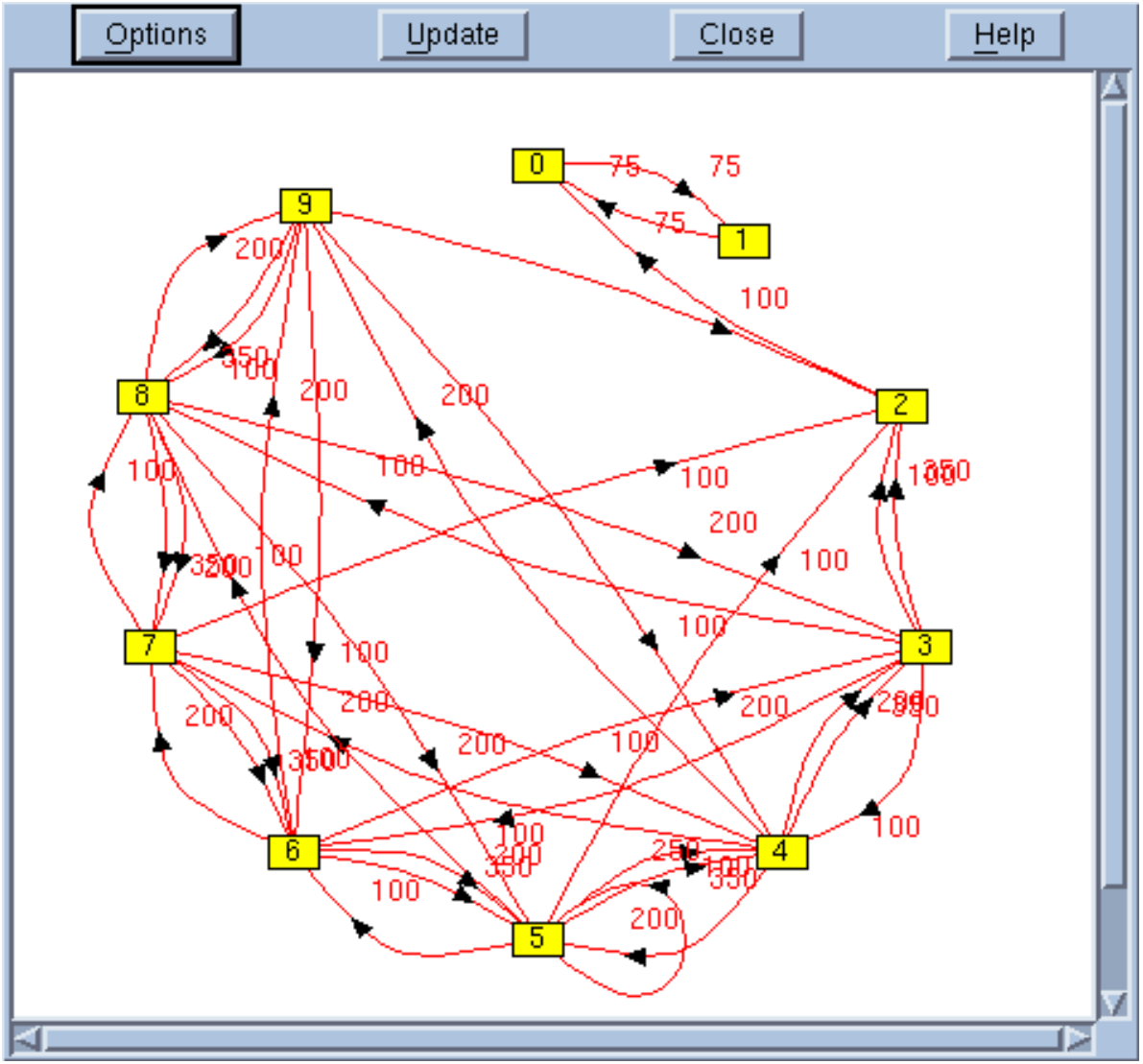
- **Quick view of program state**
  - Each call stack is a path
  - Functions are nodes
  - Calls are edges
    - Labled with the MPI rank
  - Construct process groups
- **Look for outliers**



Dive on a node in the call graph to create a Call Graph group.

Message Queue Graph

- **Hangs & Deadlocks**
- **Pending Messages**
  - Receives
  - Sends
  - Unexpected
- **Inspect**
  - Individual entries
- **Patterns**



Lab time

1. Debugger Basics
2. Viewing, Examining, Watching and Editing
3. Examining and Controlling a Parallel Application

Advanced Debugging

- Memory Debugging
- Reverse Debugging
- Batch Debugging

Memory Debugging

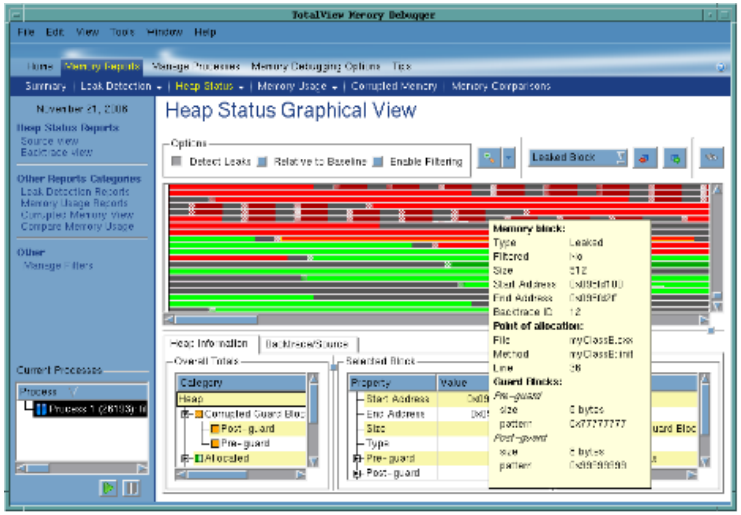
What is a Memory Bug?

- **A Memory Bug is a mistake in the management of heap memory**
  - Failure to check for error conditions
  - Leaking: Failure to free memory
  - Dangling references: Failure to clear pointers
  - Memory Corruption
    - Writing to memory not allocated
    - Over running array bounds

What is MemoryScape?

Simple to use, intuitive memory debugging

- **What is MemoryScape?**
  - Streamlined
  - Lightweight
  - Intuitive
  - Collaborative
  - Memory Debugging
- **Features**
  - Shows
    - Memory errors
    - Memory status
    - Memory leaks
    - Buffer overflows
  - MPI memory debugging
  - Remote memory debugging



- Tech
  - Low overhead
  - No Instrumentation
- Interface
  - Inductive
  - Collaboration
  - Multi-process

MemoryScape Features

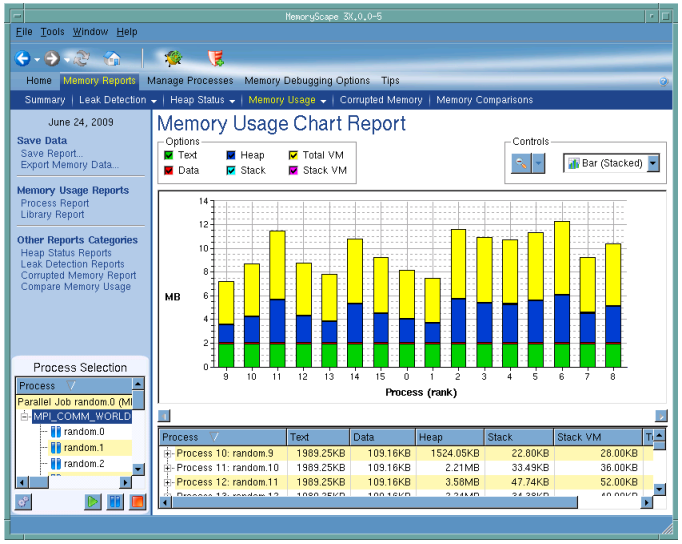
- **Automatically detect allocation problems**
- **View the heap**
- **Leak detection**
- **Block painting**
- **Memory Hoarding**
- **Dangling pointer detection**
- **Deallocation/reallocation notification**
- **Memory Corruption Detection**

  Guard Blocks

  Red Zones
- **Memory Comparisons between processes**
- **Collaboration features**

Strategies for Memory Debugging in Parallel

- **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?



Memory Debugging at Block level

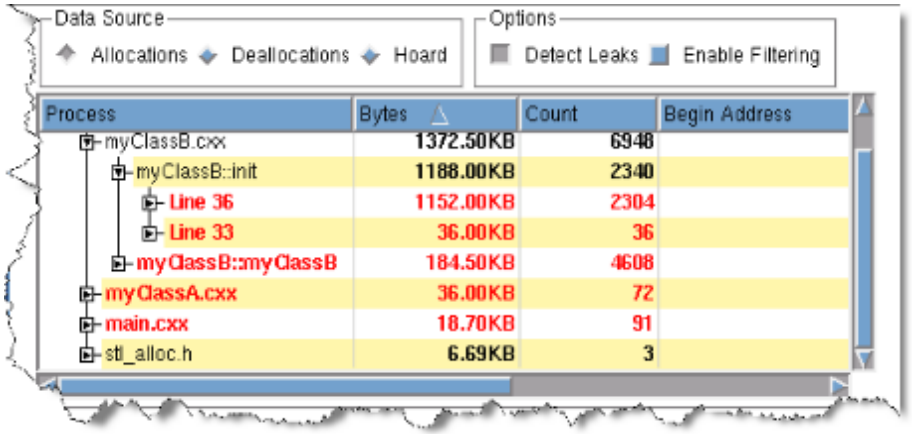
Leak Detection

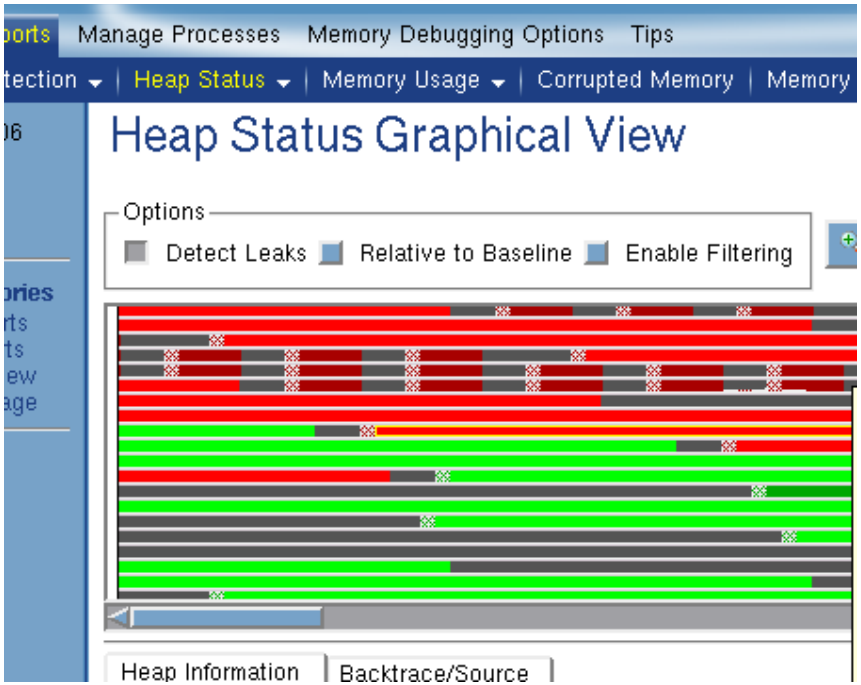
Bound Overwrites

Leak Detection

MemoryScape Leak Detection

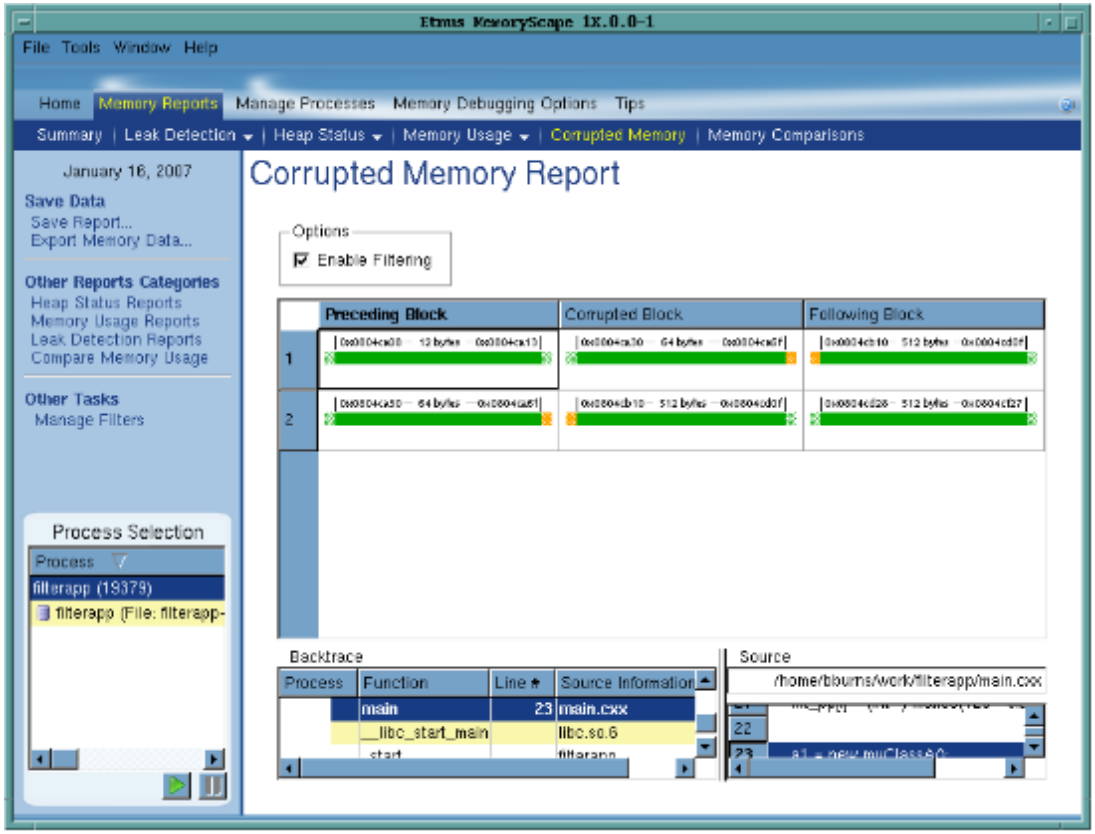
- Based on Conservative Garbage Collection
- Can be performed at any point in runtime
  - Helps localize leaks in time
- Multiple Reports
  - Backtrace Report
  - Source Code Structure
  - Graphically Memory Location





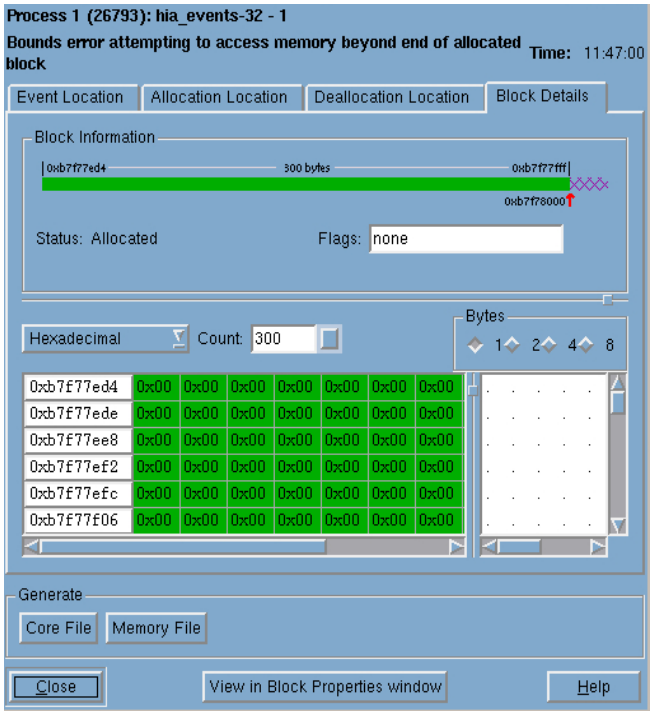
Array Bounds Violations

- **Heap Guard Blocks**
  - Before and/or After
  - All Allocations or just a few
  - Variable Size
  - Check at Any Time
  - Reports
    - By Memory Address
    - Only Corrupted



RedZones catch buffer overflows

- **Allocates a "protected page"**
  - adjacent to selected heap allocations
  - Before or after allocated block
- **Writes into this space trigger events**
  - Event occurs as the write is happening
- **Pages have a fixed size**
  - If there are many heap allocations this can potential have a large memory usage overhead
- **Ways to manage RedZones memory overhead**
  - Turn RedZones on and off manually
  - Specify (by size) what allocations you want to have RedZones on

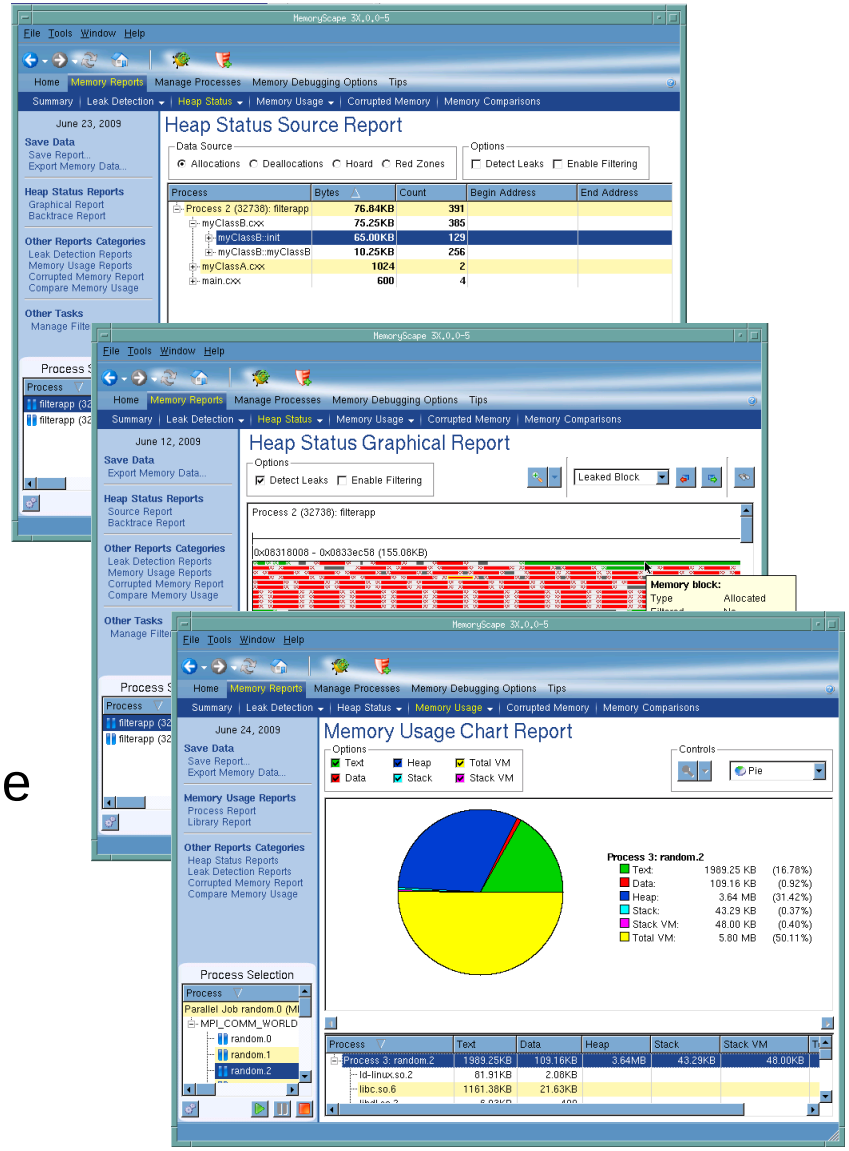


Memory Debugging

Memory Reports and Analysis

Memory Reports

- **Multiple Reports**
  - Memory Statistics
  - Interactive Graphical Display
  - Source Code Display
  - Backtrace Display
- **Allow the user to**
  - Monitor Program Memory Usage
  - Discover Allocation Layout
  - Look for Inefficient Allocation
  - Look for Memory Leaks



Reverse Debugging

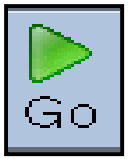
Replay Engine – The right way to debug

Step forward over functions

Step forward into functions

Advance forward out of current Function, after the call

Advance forward to selected line



Run forward

Step *backward* over functions

Step *backward* into functions

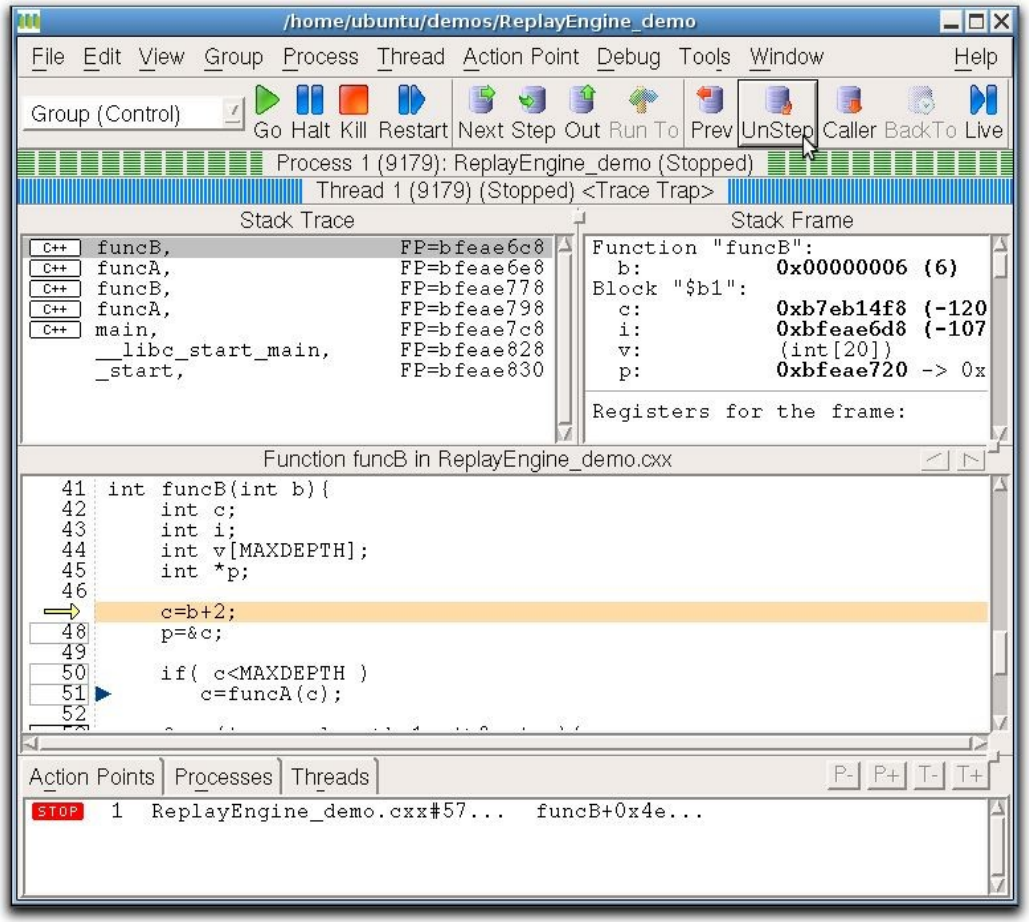
Advance backward out of current Function, to before the call

Advance backward to selected line

Run *backward*

Advance forward to “live” session

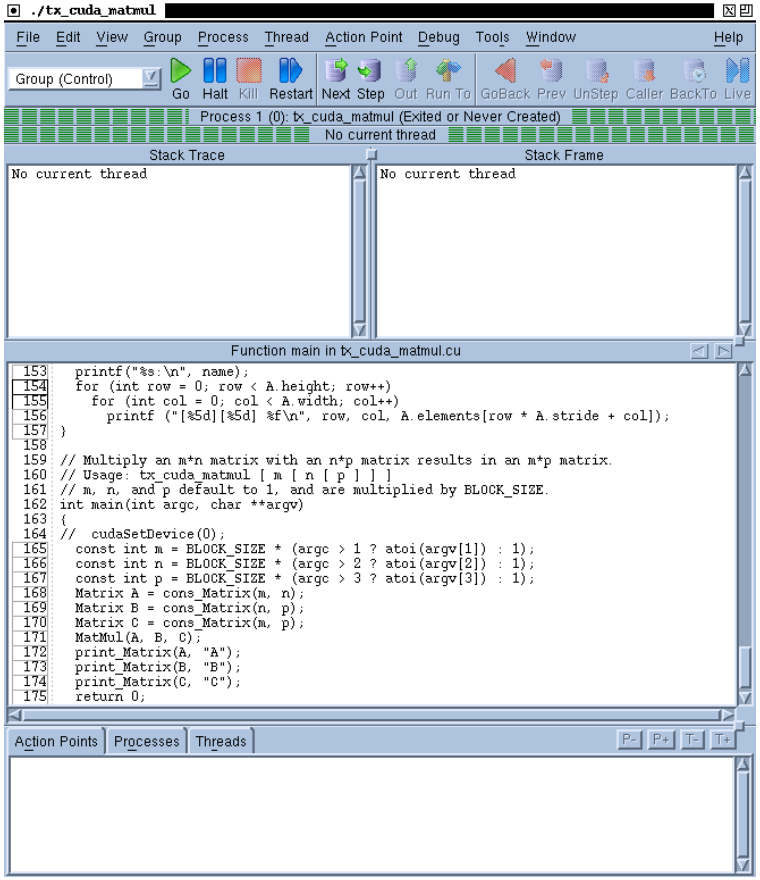
Replay Engine

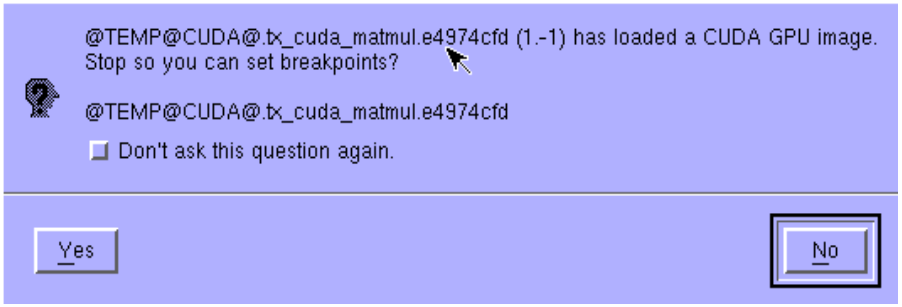


- **Captures execution history**
  - Records all external input to program
  - Records internal sources of non-determinism
- **Replays execution history**
  - Examine any part of the execution history
  - Step back as easily as forward
  - Jump to points of interest
- **An add-on product to TotalView**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

TotalView Debugger for CUDA

Starting TotalView





- When a new kernel is loaded you get the option of setting breakpoints

- You can debug the CUDA host code using the normal TotalView commands and procedures

Debugging CUDA

Thread (x,y,z)

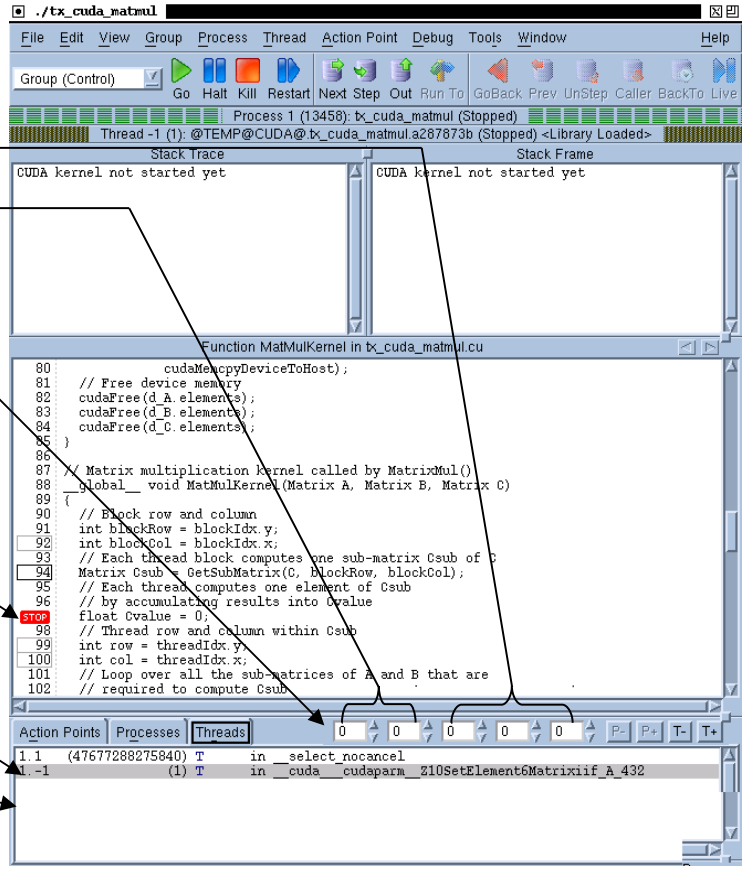
Block (x,y)

GPU focus thread selector for changing the block (x,y) and thread (x,y,z) indexes of the CUDA thread

Select a line number in a box to plant a breakpoint

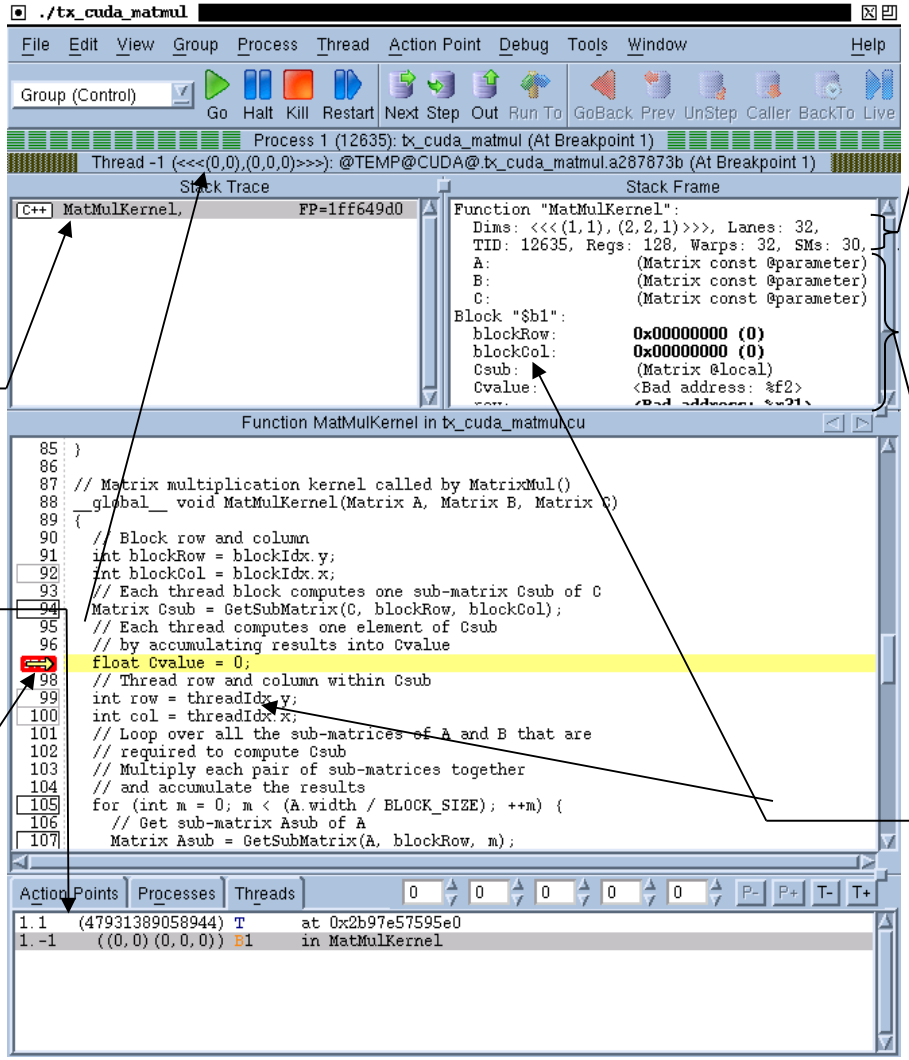
CUDA host threads have a positive TotalView thread ID

CUDA GPU threads have a negative TotalView thread ID



Dive on a variable name to open a variable window

Running to a Breakpoint in the GPU code



Stack backtrace and inlined functions

GPU focus thread logical coordinates

PC arrow for the warp

CUDA grid and block dimensions, lanes/warp, warps/SM, SMs, etc.

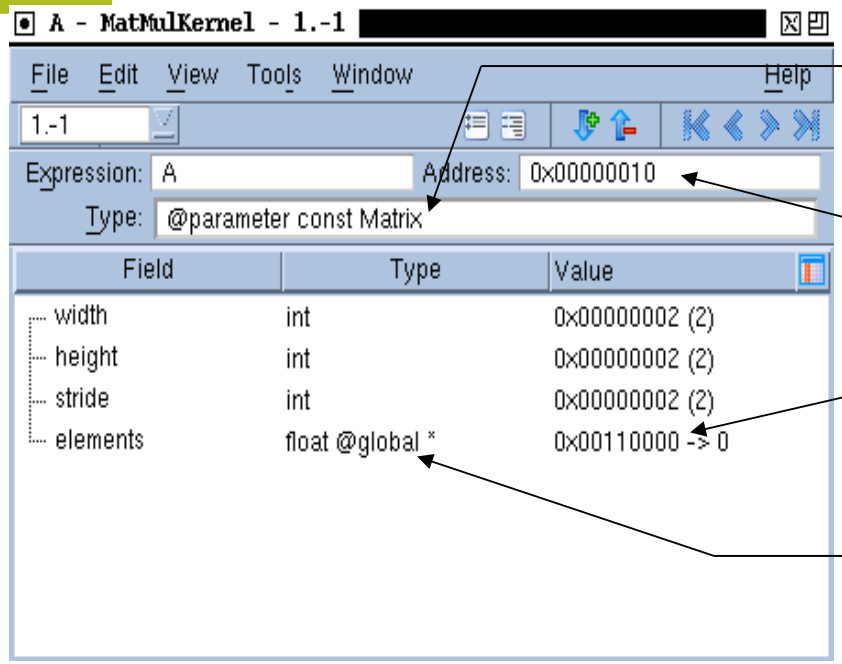
Parameter, register, local and shared variables

Dive on a variable name to open a variable window

Stepping GPU Code

- **single-step operation advances all of the GPU hardware threads in the *same* warp**
- **To advance the execution of more than one warp, you may either:**
  - set a breakpoint and continue the process, or
  - select a line number in the source pane and select “Run To”.

GPU Variables and Data Display



"@parameter" type qualifier indicates that variable "A" is in parameter storage

Address 0x10 is an offset within parameter storage

Pointer value 0x110000 is an offset within global storage

"elements" is a pointer to a float in global storage

Labs Part II

4. Exploring Heap Memory in MPI applications
5. Reverse Debugging with ReplayEngine
6. Batch Mode Debugging with TVScript

TotalView Customer Support

support@roguewave.com

Thanks!

QUESTIONS?

www.roguewave.com

www.totalviewtech.com