

TotalView Workshop SARA 2010

Nikolay Piskun
Director of Continuing Engineering

Royd Lüdtké
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

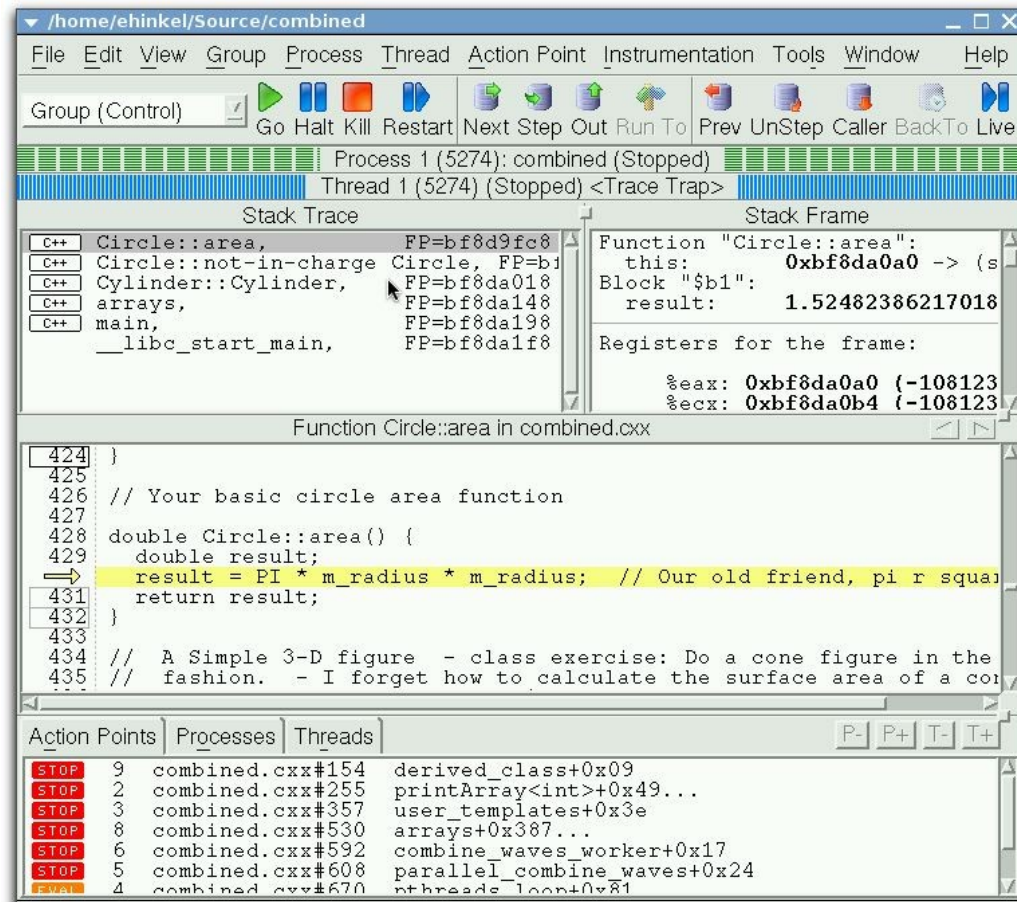
Introduction to Totalview Tech Products

- **Introduction**
- **Totalview Basics**
- **Parallel Debugging**

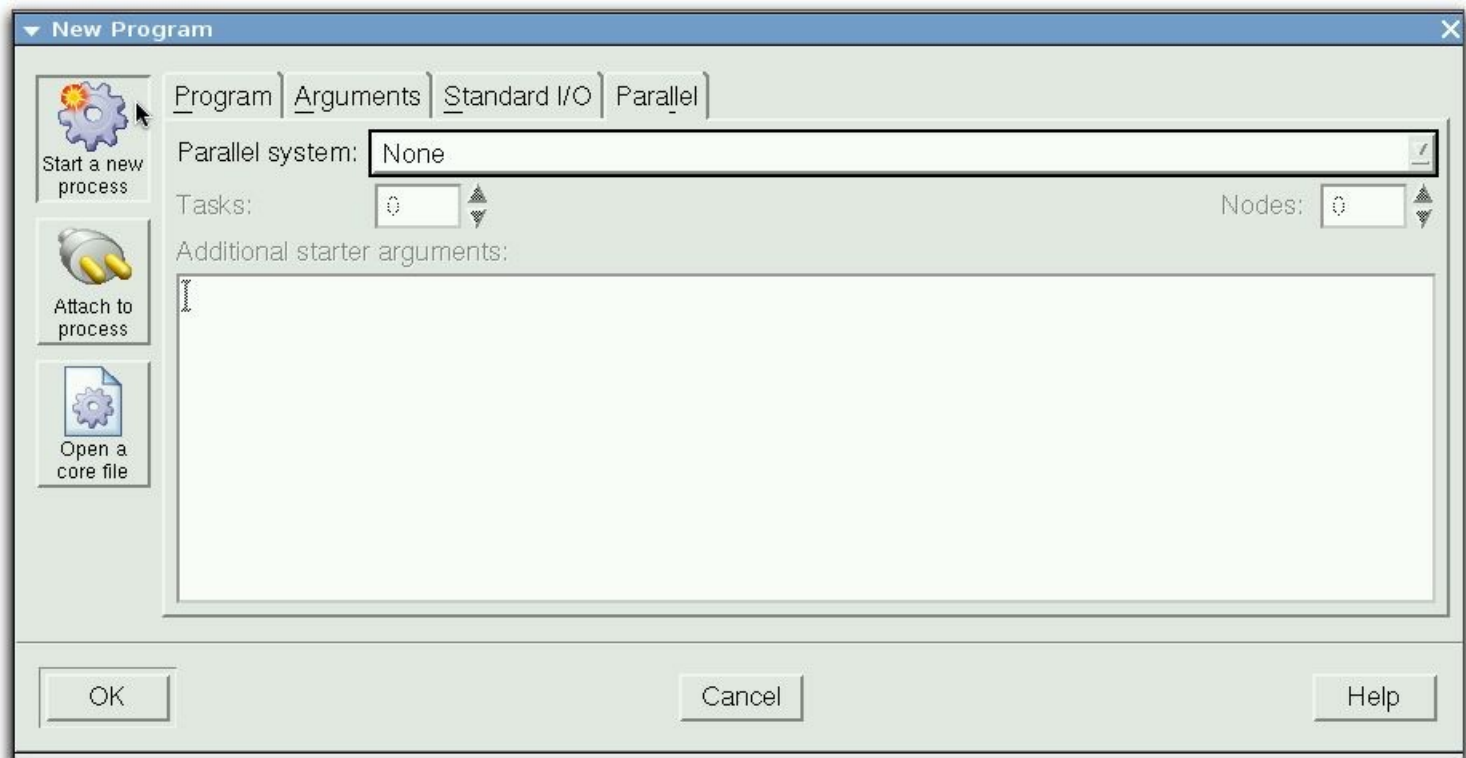
What is TotalView?

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



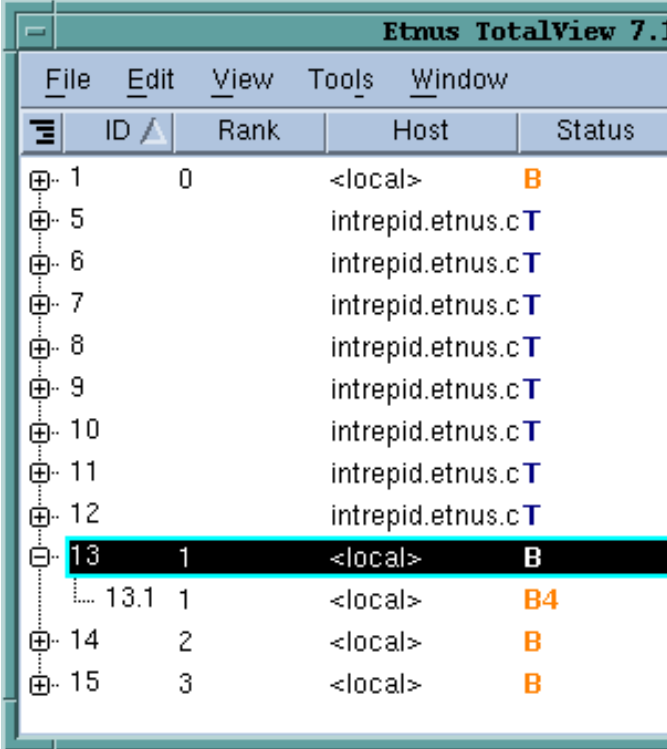
Open a Core File

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



ID	Rank	Host	Status
1	0	<local>	B
5		intrepid.etnus.c	T
6		intrepid.etnus.c	T
7		intrepid.etnus.c	T
8		intrepid.etnus.c	T
9		intrepid.etnus.c	T
10		intrepid.etnus.c	T
11		intrepid.etnus.c	T
12		intrepid.etnus.c	T
13	1	<local>	B
13.1	1	<local>	B4
14	2	<local>	B
15	3	<local>	B

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

The screenshot shows the TotalView Process Window for a process named 'fork_loopLinux'. The window is divided into several panes:

- Toolbar:** Located at the top, it contains buttons for 'Go', 'Halt', 'Kill', 'Restart', 'Next', 'Step', 'Out', and 'Run To'. A 'Group (Control)' dropdown menu is also present.
- Stack Trace Pane:** Located on the left side, it displays a list of stack frames with their respective frame pointers (FP). The frames are: `__select` (FP=bffff088), `wait_a_while` (FP=bffff088), `snore` (FP=bffff0c8), `forker` (FP=bffff148), `fork_wrapper` (FP=bffff1b8), `main` (FP=bffff1e8), and `__libc_start_main` (FP=bffff218).
- Stack Frame Pane:** Located on the right side, it shows the details of the current stack frame for the 'snore' function. It includes arguments (arg: 0x00000000), block addresses (Block "\$b1#\$b2", Block "\$b1"), and various variables (me: 0x00000000 (0), entry_count (%eax): 0xffffdfe, old_ticket: 0xffffffff (-1), ticket: 0x00000004 (4), pts: 0x00004000 (16384)).
- Source Pane:** Located in the center, it displays the source code for the 'snore' function in 'fork_loop.cxx'. The code is shown with line numbers (645-665) and a yellow arrow pointing to line 656, which contains the `if (verbose)` statement.
- Tabbed Area:** Located at the bottom, it shows a tabbed area with tabs for 'Action Points', 'Processes', and 'Threads'. The 'Threads' tab is active, showing a list of threads numbered 1 through 8.

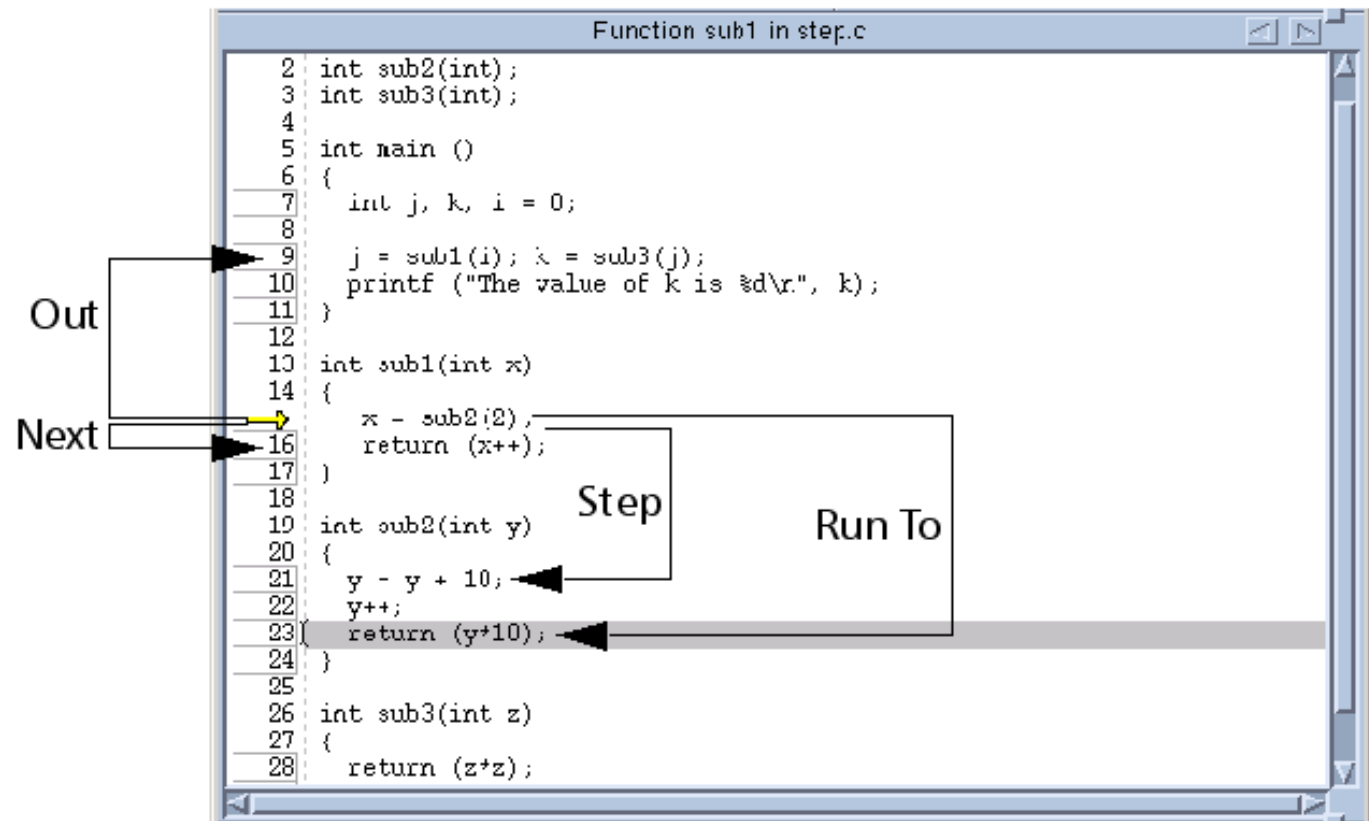
Provides detailed state of one process, or a single thread within a process

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

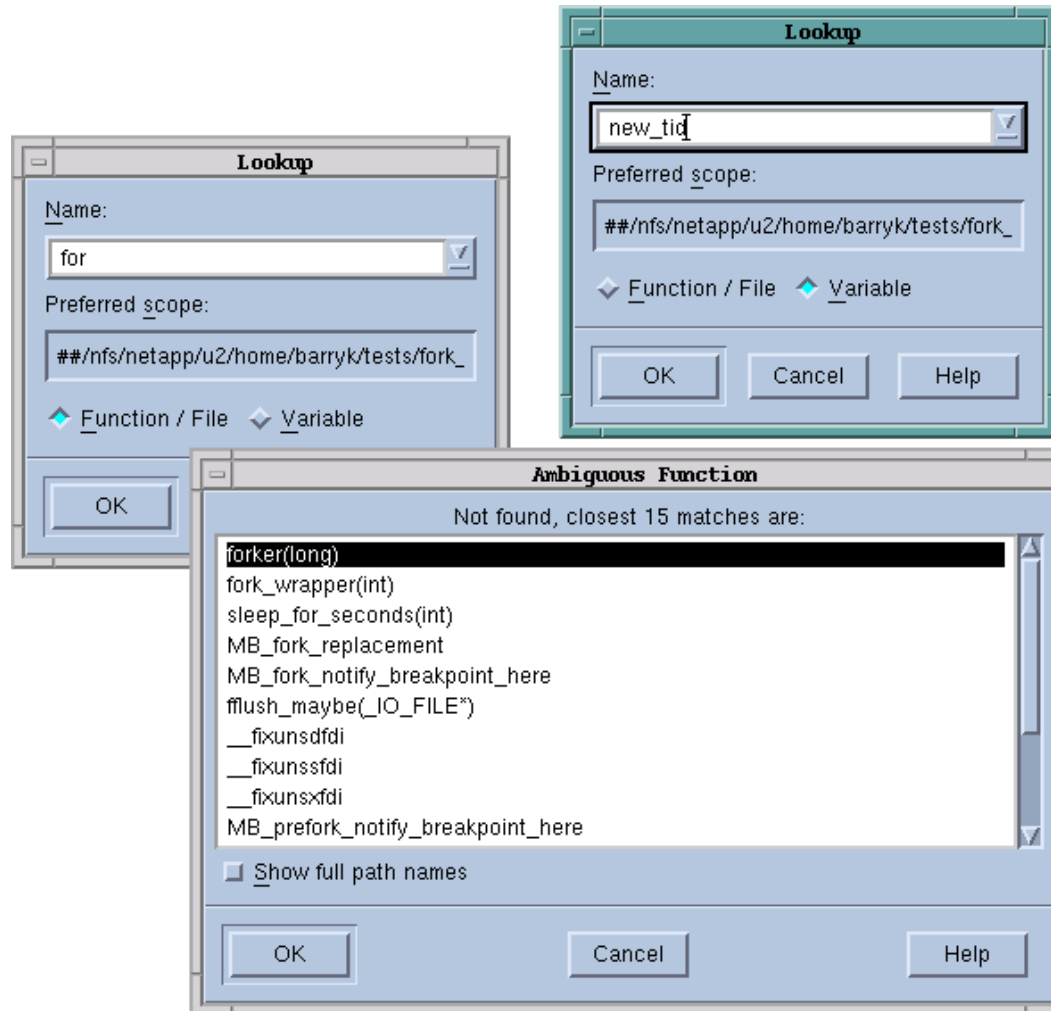
Stepping Commands



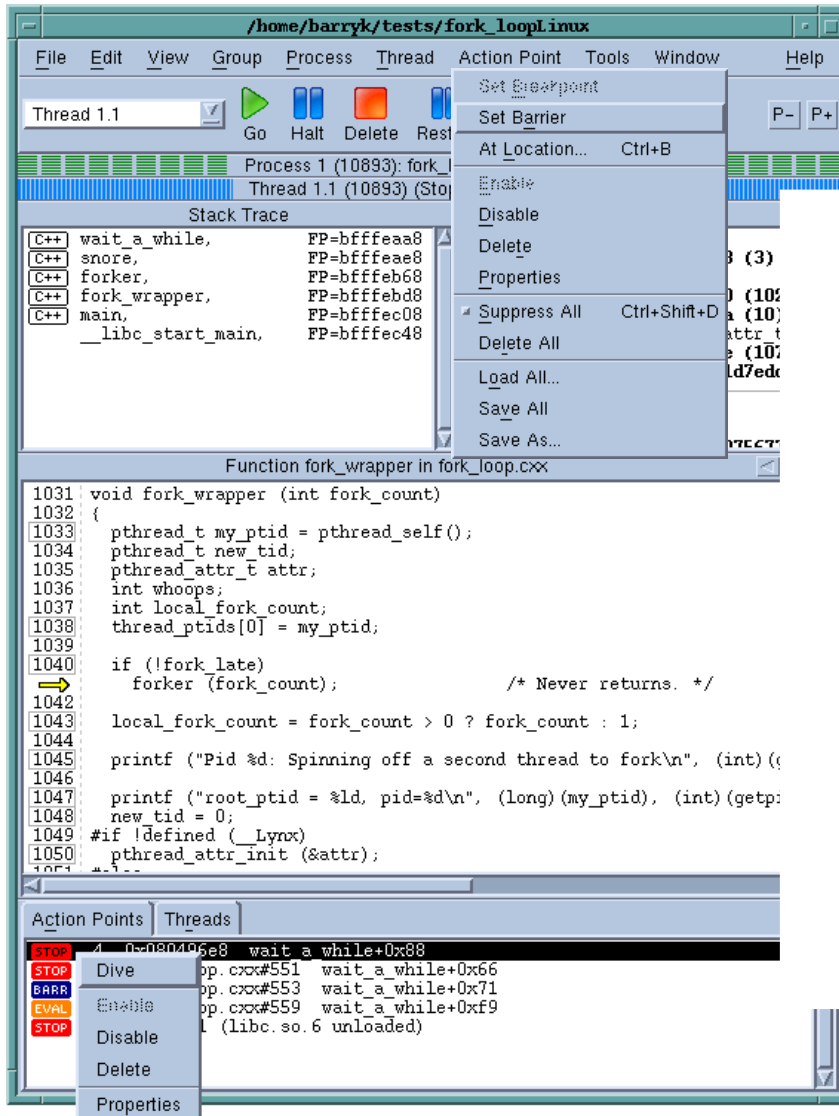
Based on
PC location



Finding Functions, Variables, and Source Files



Action Points



Breakpoints

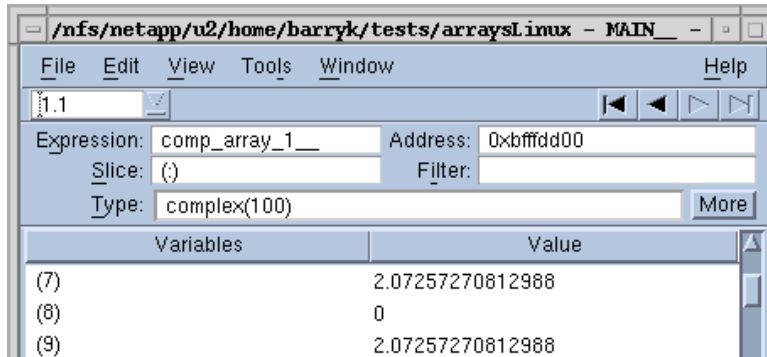
Barrier Points

Conditional Breakpoints

Evaluation Points

Watchpoints

Watchpoints

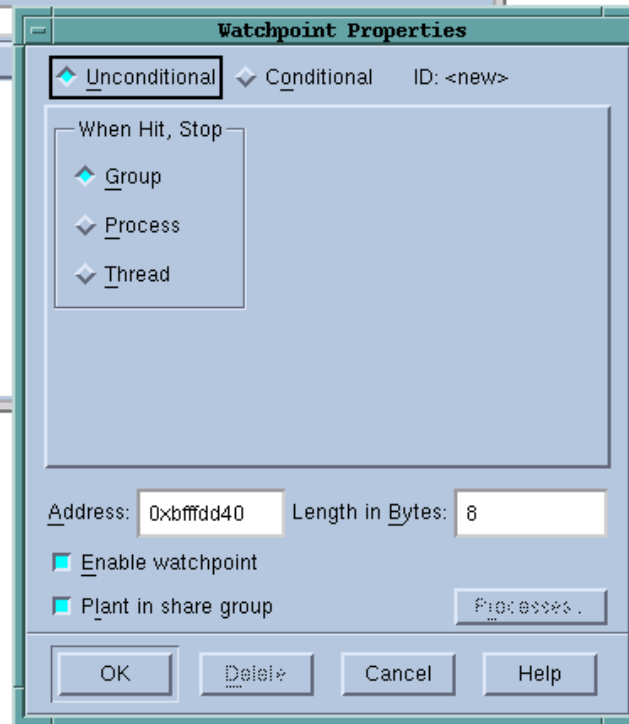
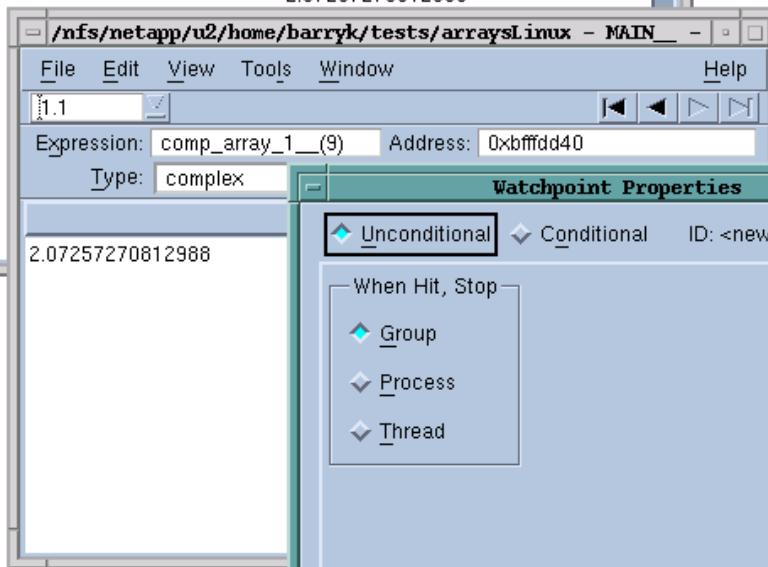


Watchpoints are set on a fixed memory region

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.

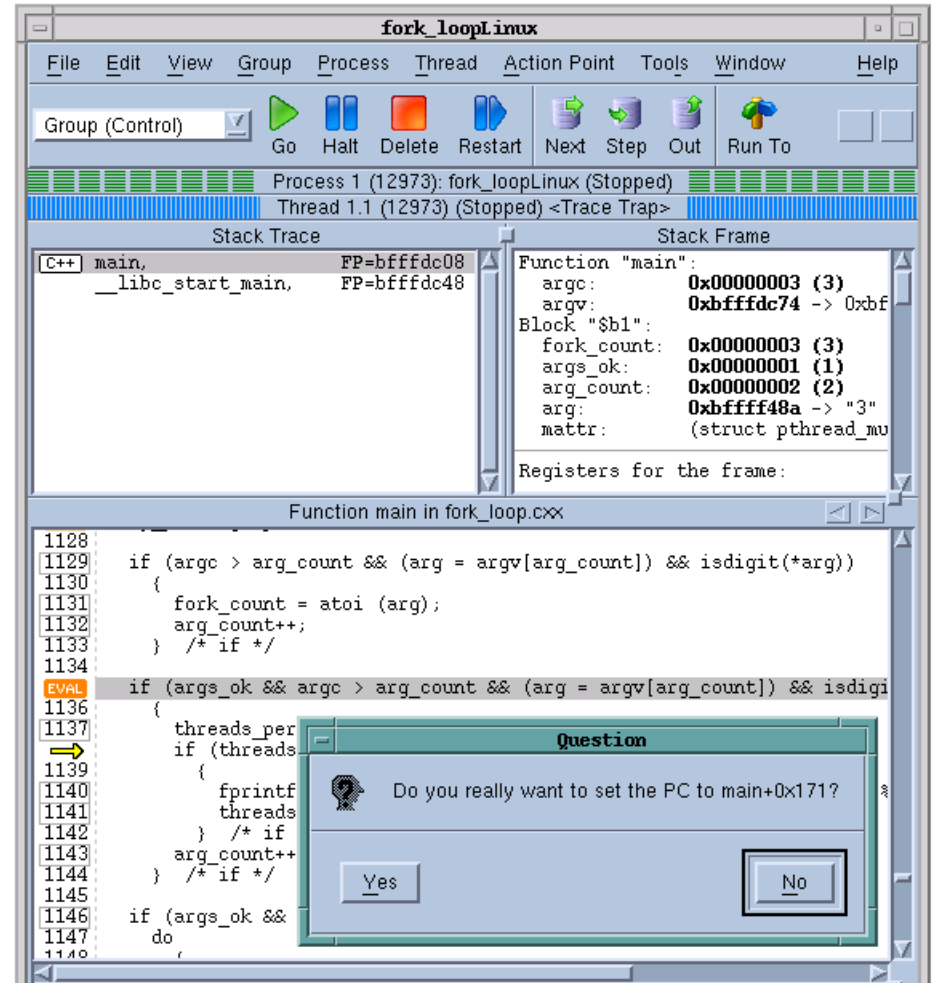
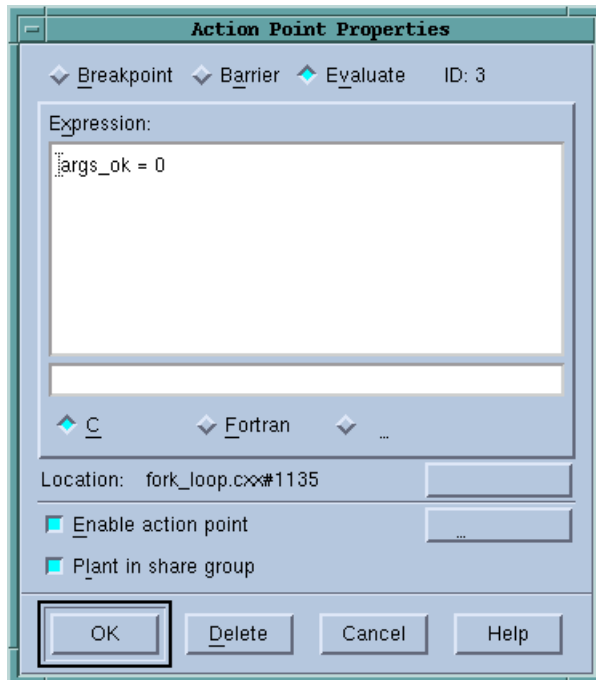


Watchpoints are not set on a variable. You you need to be aware of the variable scope.

Watchpoints can be conditional or unconditional

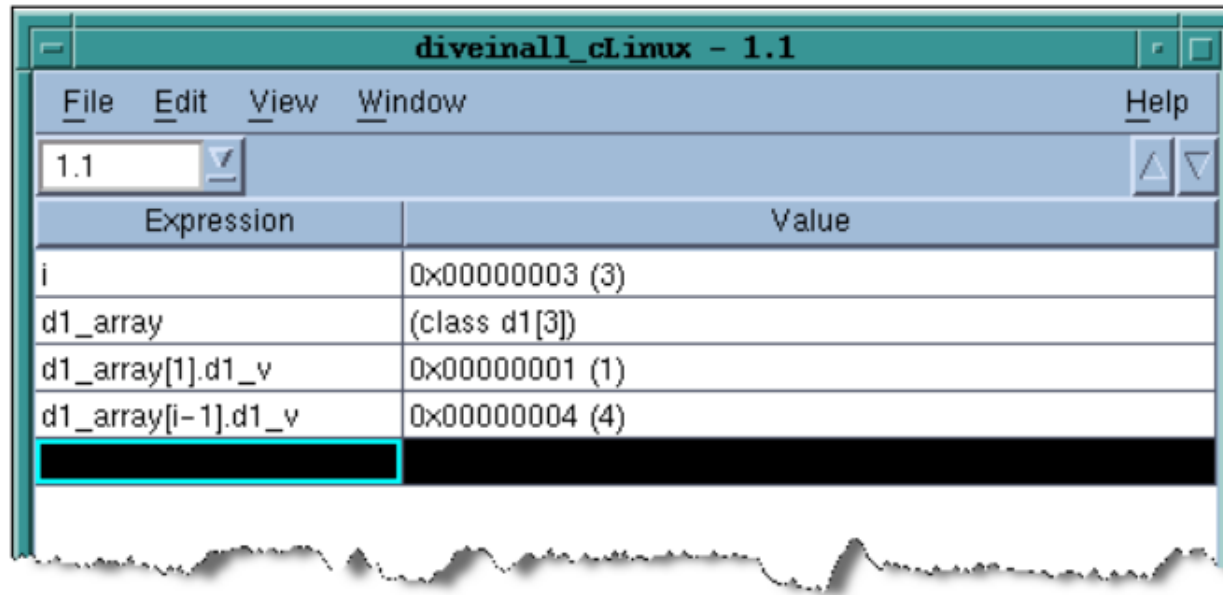
Uses Hardware Watchpoints with various limitations based on architecture

Using Set PC to resume execution at an arbitrary point



Viewing and Editing Data

Expression List Window

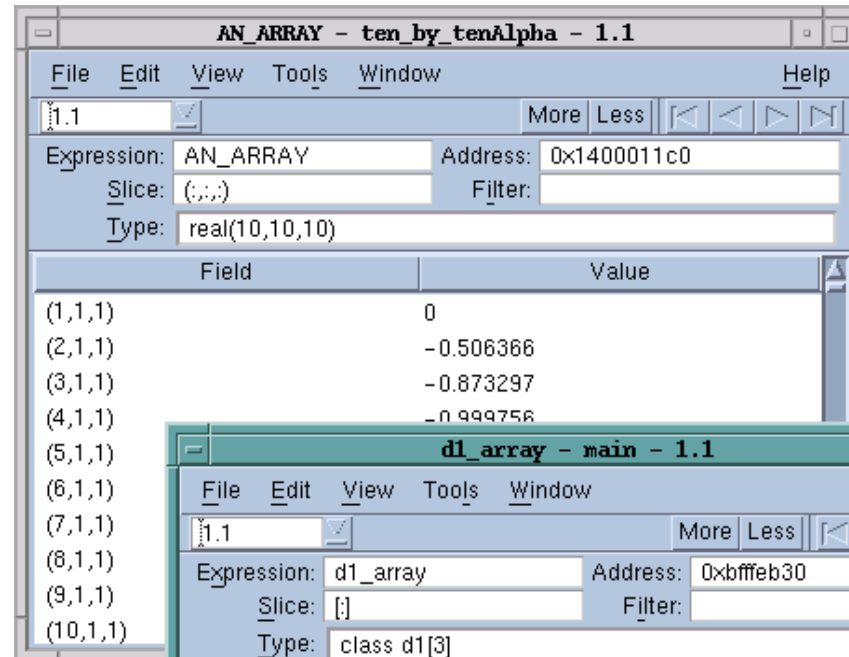


Add to the expression list using contextual menu with right-click on a variable, or by typing an expression directly in the window

- 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



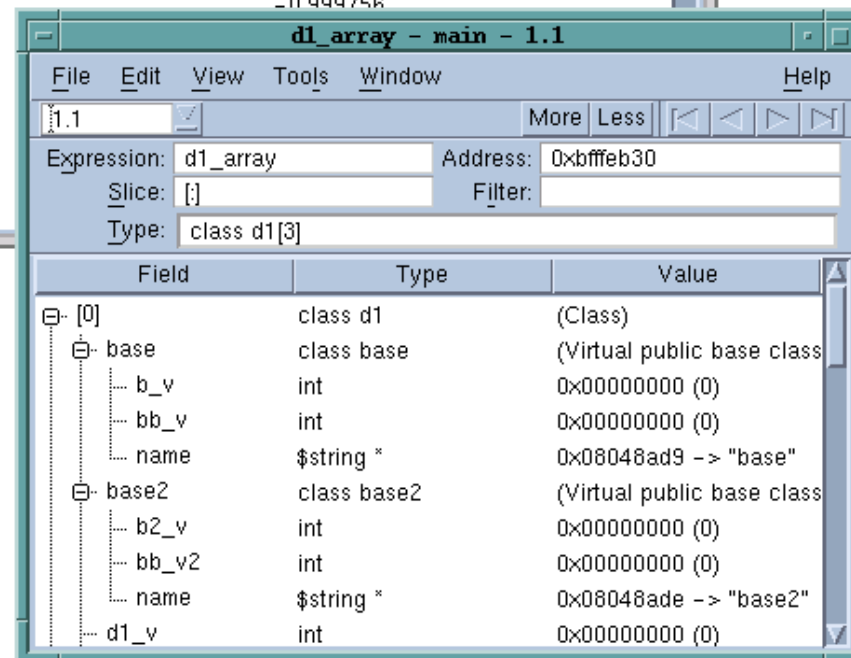
AN_ARRAY - ten_by_tenAlpha - 1.1

File Edit View Tools Window Help

1.1 More Less

Expression: AN_ARRAY Address: 0x1400011c0
 Slice: (...) Filter:
 Type: real(10,10,10)

Field	Value
(1,1,1)	0
(2,1,1)	-0.506386
(3,1,1)	-0.873297
(4,1,1)	-0.999756
(5,1,1)	
(6,1,1)	
(7,1,1)	
(8,1,1)	
(9,1,1)	
(10,1,1)	



d1_array - main - 1.1

File Edit View Tools Window Help

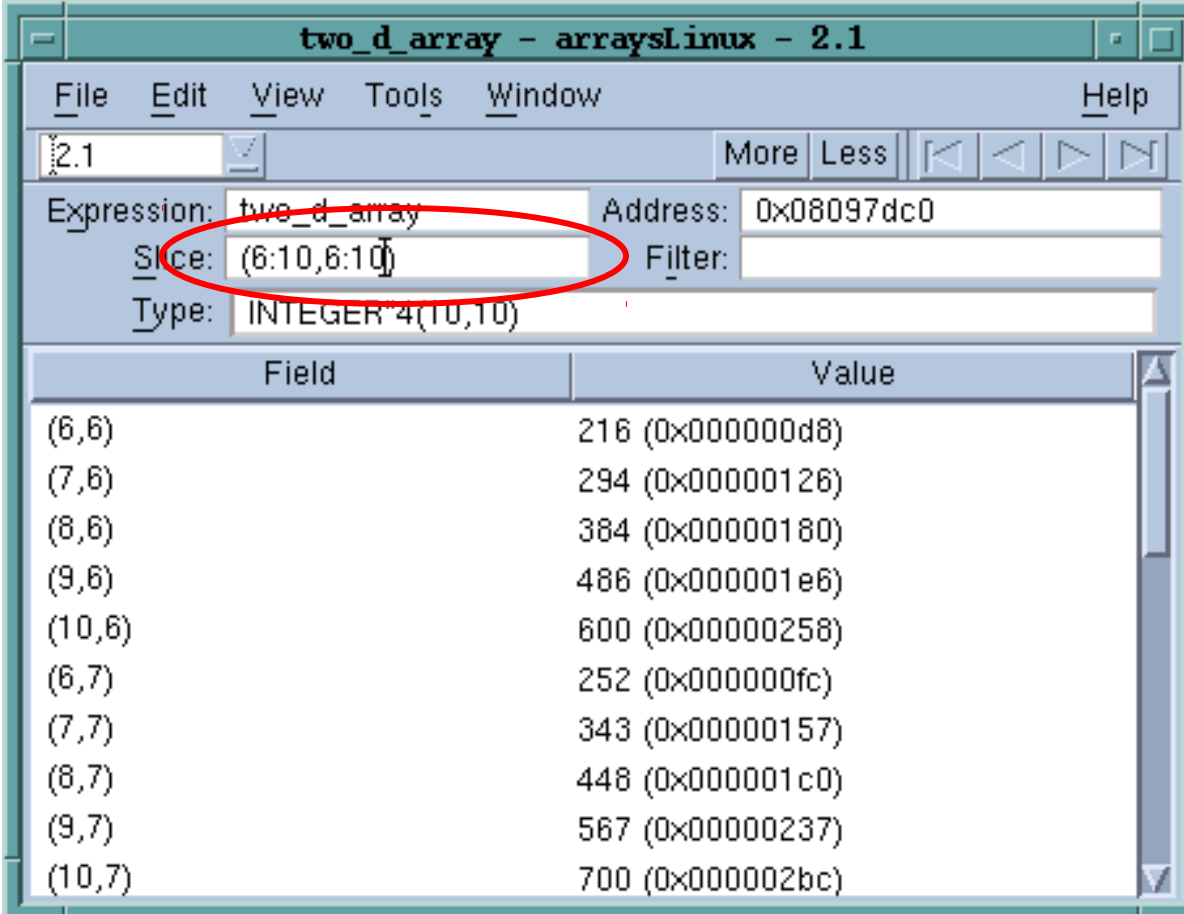
1.1 More Less

Expression: d1_array Address: 0xbfffeb30
 Slice: [] Filter:
 Type: class d1[3]

Field	Type	Value
[0]	class d1	(Class)
base	class base	(Virtual public base class
b_v	int	0x00000000 (0)
bb_v	int	0x00000000 (0)
name	\$string *	0x08048ad9 -> "base"
base2	class base2	(Virtual public base class
b2_v	int	0x00000000 (0)
bb_v2	int	0x00000000 (0)
name	\$string *	0x08048ade -> "base2"
d1_v	int	0x00000000 (0)

Structure Arrays

Slicing Arrays



two_d_array - arraysLinux - 2.1

File Edit View Tools Window Help

2.1 More Less

Expression: two_d_array Address: 0x08097dc0

Slice: (6:10,6:10) Filter:

Type: INTEGER 4(10,10)

Field	Value
(6,6)	216 (0x000000d8)
(7,6)	294 (0x00000126)
(8,6)	384 (0x00000180)
(9,6)	486 (0x000001e6)
(10,6)	600 (0x00000258)
(6,7)	252 (0x000000fc)
(7,7)	343 (0x00000157)
(8,7)	448 (0x000001c0)
(9,7)	567 (0x00000237)
(10,7)	700 (0x000002bc)

Slice notation is [start:end:stride]

Filtering Arrays

IEEE Array Filter Example 1

Expression: ieee_array Address: 0x1408214a0
 Slice: () Filter: **.eq. \$inf**
 Type: \$real_4(6)

Field	Value
(1)	INF
(2)	-INF

IEEE Array Filter Example 2

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 Filter Example 3

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>

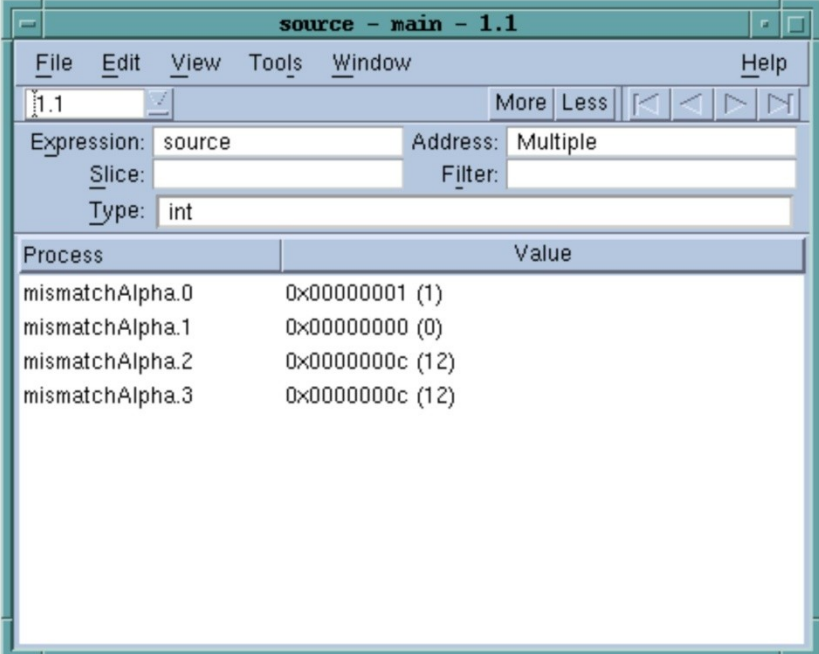
int2 Array Filter Example

Expression: int2_array__ Address: 0xbfffd450
 Slice: () Filter: **\$value > 20 .and. \$value < 100**
 Type: word(100)

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**



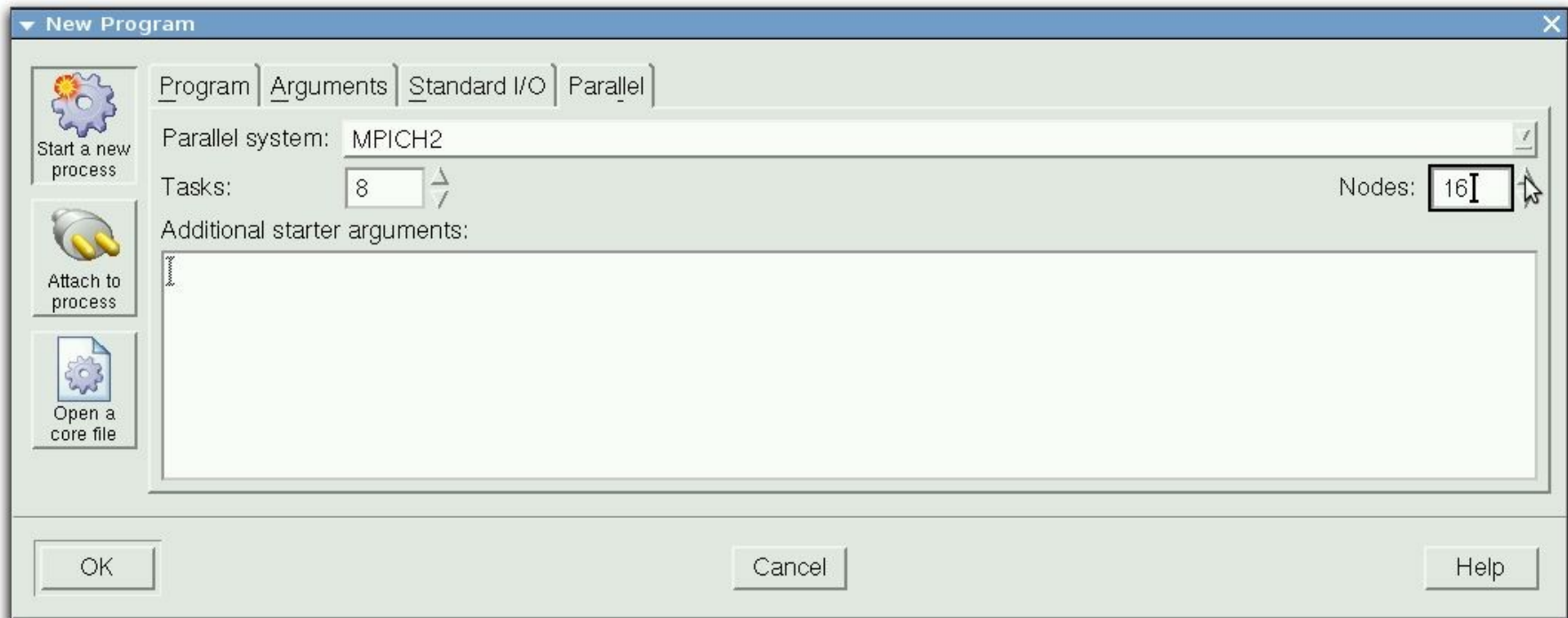
The screenshot shows a TotalView window titled "source - main - 1.1". The interface includes a menu bar (File, Edit, View, Tools, Window, Help) and a toolbar with "More" and "Less" buttons. The "Expression" field contains "source", "Address" is "Multiple", "Type" is "int", and "Filter" is empty. Below this is a table with two columns: "Process" and "Value".

Process	Value
mismatchAlpha.0	0x00000001 (1)
mismatchAlpha.1	0x00000000 (0)
mismatchAlpha.2	0x0000000c (12)
mismatchAlpha.3	0x0000000c (12)

Parallel Debugging

TotalView Startup with MPI TVT Launch

In the Parallel tab, select:

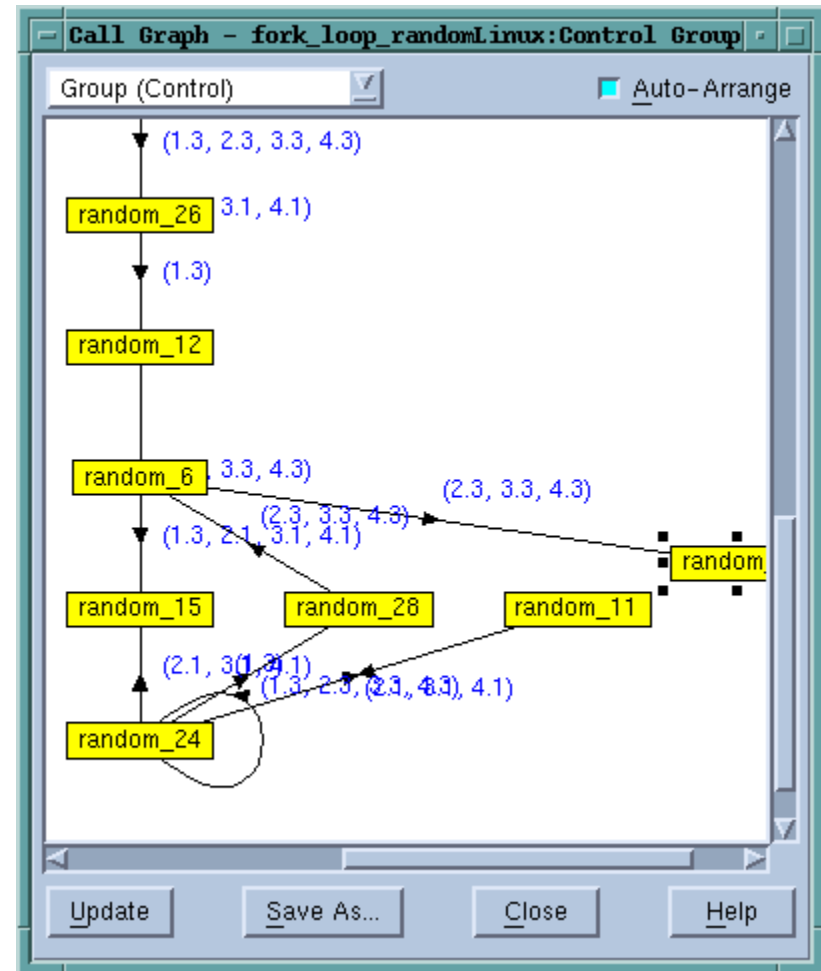


your MPI preference, number of tasks, and number of nodes.

... then add any additional starter arguments

Call Graph

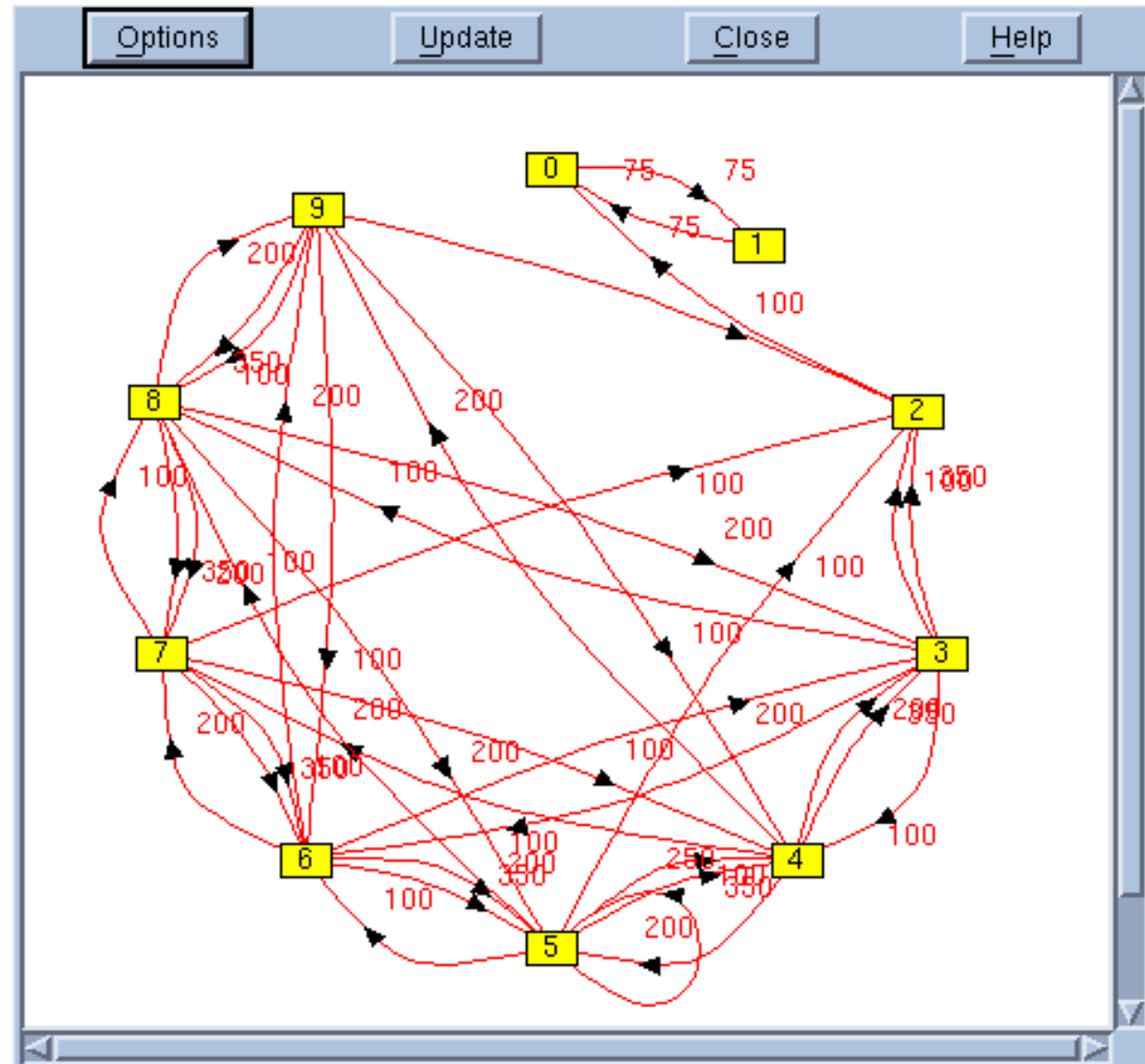
- **Quick view of program state**
 - Each call stack is a path
 - Functions are nodes
 - Calls are edges
 - Labeled with the MPI rank
 - Construct process groups
- **Look for outliers**



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?

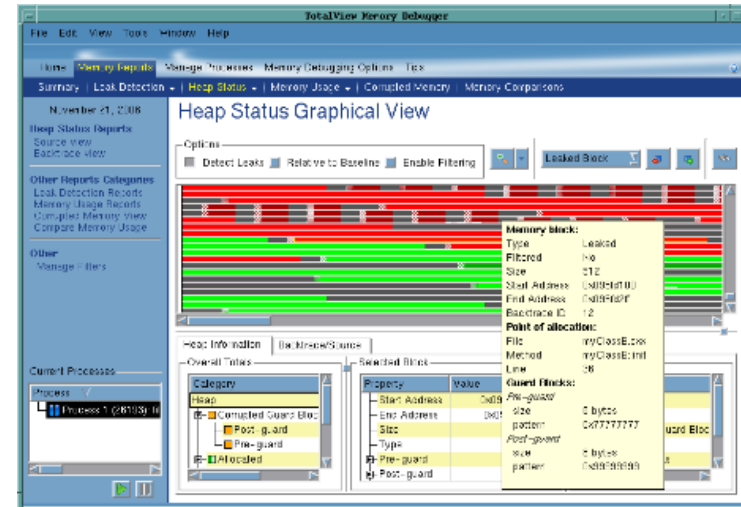
Simple to use, intuitive memory debugging

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



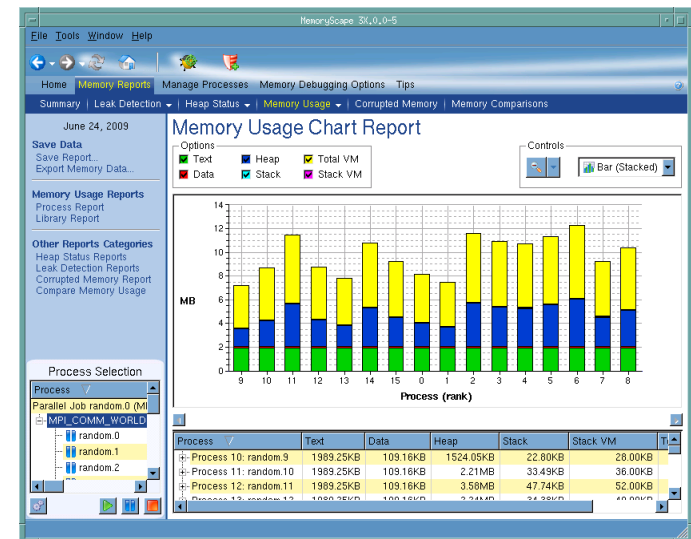
- Tech
 - Low overhead
 - No Instrumentation
- Interface
 - Inductive
 - Collaboration
 - Multi-process

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?



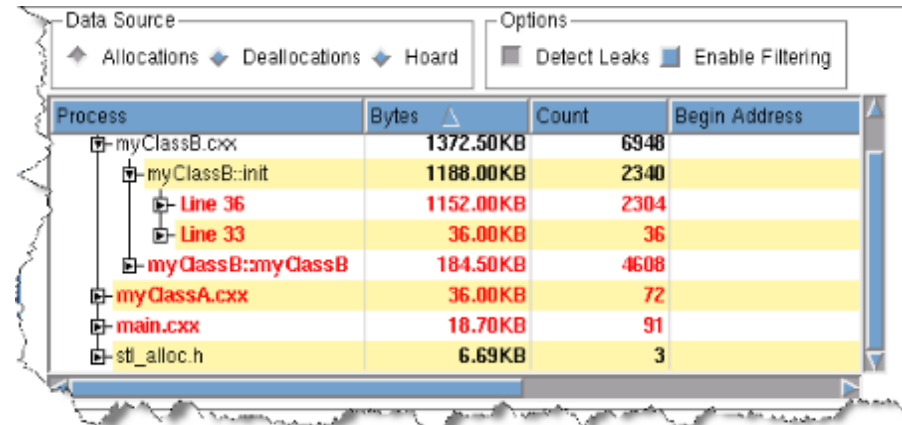
Memory Debugging at Block level

Leak Detection
Bound Overwrites

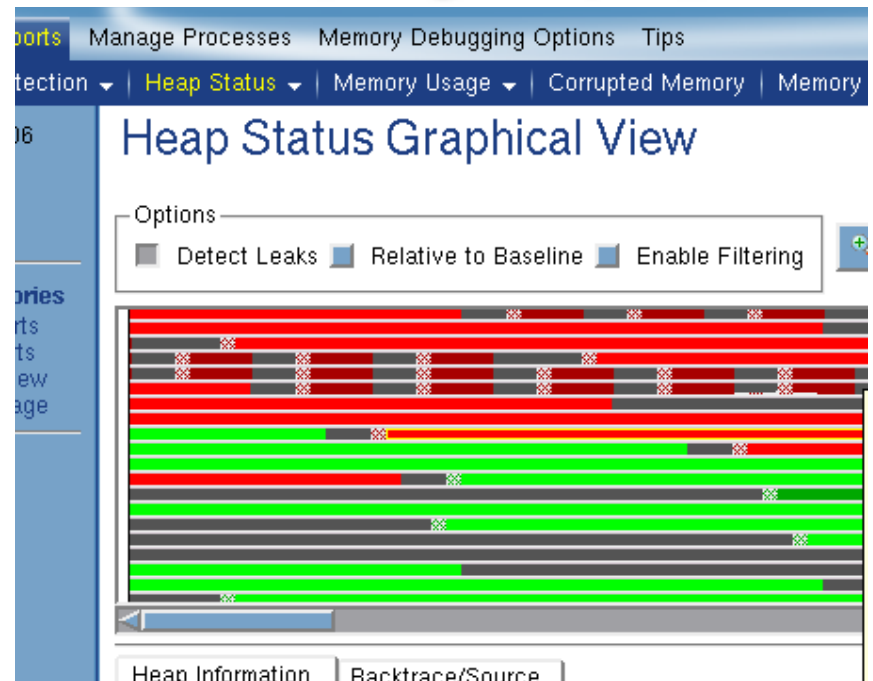
Leak Detection

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



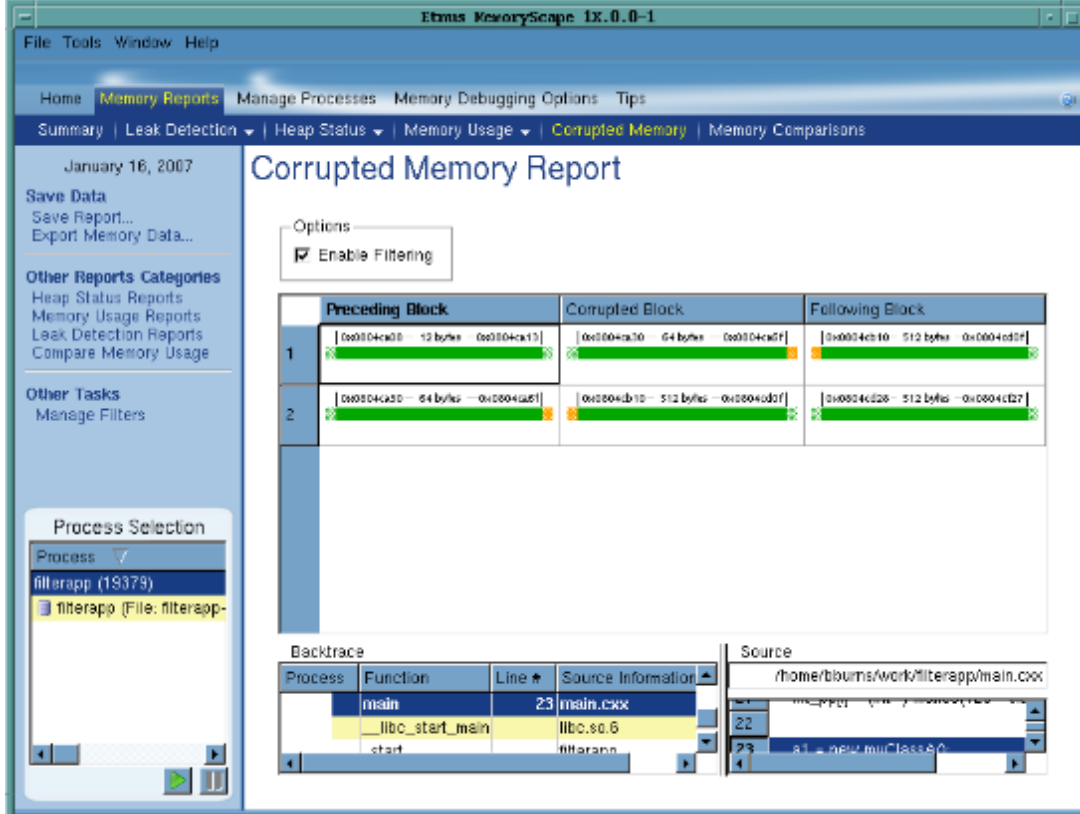
Process	Bytes	Count	Begin Address
-myClassB.cxx	1372.50KB	6948	
-myClassB::init	1188.00KB	2340	
-Line 36	1152.00KB	2304	
-Line 33	36.00KB	36	
-myClassB::myClassB	184.50KB	4608	
-myClassA.cxx	36.00KB	72	
-main.cxx	18.70KB	91	
-stl_alloc.h	6.69KB	3	



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



Etms MemoryScope 1X.0.0-1

File Tools Window Help

Home **Memory Reports** Manage Processes Memory Debugging Options Tips

Summary Leak Detection Heap Status Memory Usage **Corrupted Memory** Memory Comparisons

January 16, 2007

Save Data
Save Report...
Export Memory Data...

Other Reports Categories
Heap Status Reports
Memory Usage Reports
Leak Detection Reports
Compare Memory Usage

Other Tasks
Manage Filters

Process Selection
Process: /
fillerapp (19379)
fillerapp (File: fillerapp-)

Options
 Enable Filtering

	Preceding Block	Corrupted Block	Following Block
1	[0x0040a00 - 12 bytes - 0x0040a13]	[0x0040a30 - 64 bytes - 0x0040d7f]	[0x0040b10 - 512 bytes - 0x0040d7f]
2	[0x0040a30 - 64 bytes - 0x0040d7f]	[0x0040b10 - 512 bytes - 0x0040d7f]	[0x0040d26 - 512 bytes - 0x0040d7f]

Backtrace

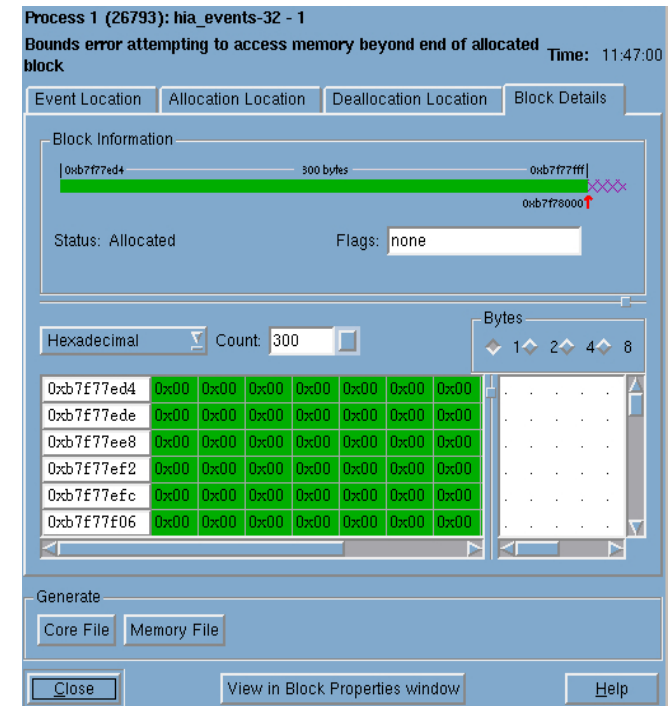
Process	Function	Line #	Source Information
	main	23	main.cxx
	__libc_start_main		libc.so.6
	start		fillerapp

Source
/home/bburns/work/fillerapp/main.cxx

22
23

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

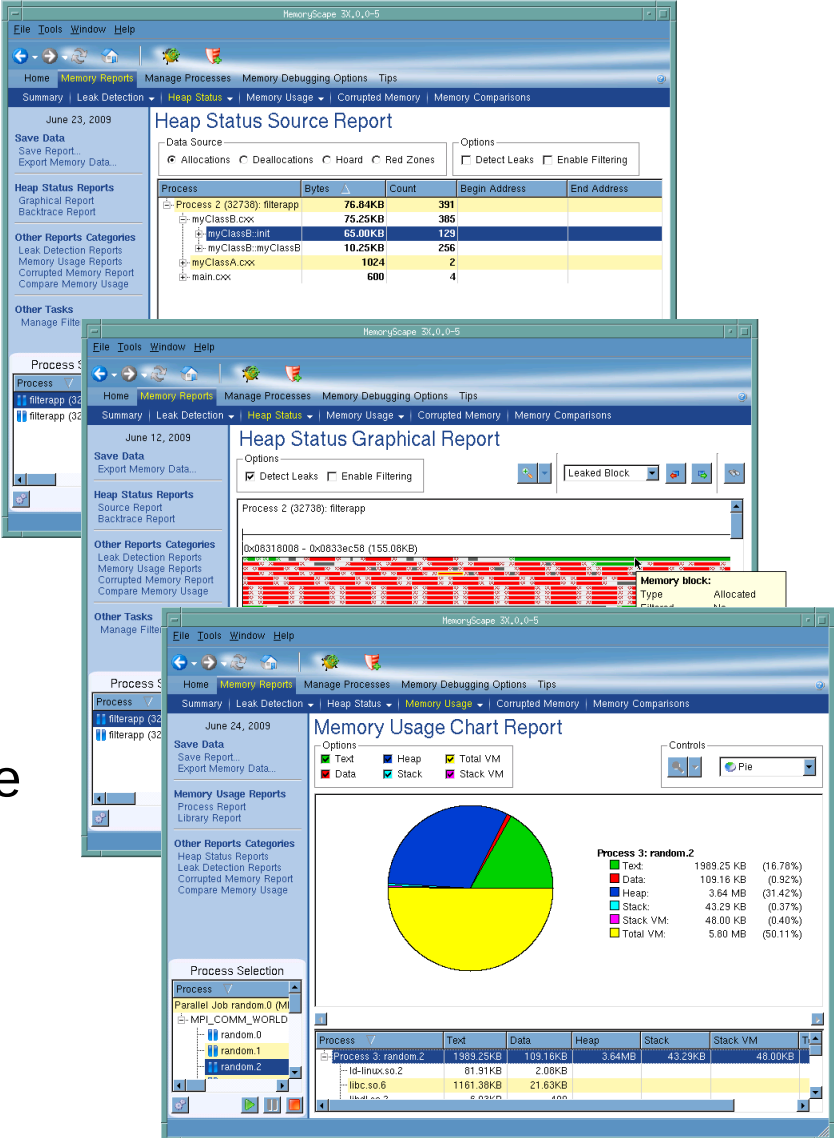
Memory Reports

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



Heap Status Source Report

Process	Bytes	Count	Begin Address	End Address
Process 2 (32738): filterapp	76.84KB	391		
myClassB.cxx	75.25KB	385		
myClassB::init	65.00KB	129		
myClassB::myClassB	10.25KB	256		
myClassA.cxx	1024	2		
main.cxx	600	4		

Heap Status Graphical Report

Process 2 (32738): filterapp

0x08318008 - 0x0833ec58 (155.08KB)

Memory block: Type: Allocated

Memory Usage Chart Report

Process 3: random.2

Category	Size	Percentage
Text	1989.25 KB	(16.78%)
Data	109.18 KB	(0.92%)
Heap	3.84 MB	(31.42%)
Stack	43.29 KB	(0.37%)
Stack VM	48.00 KB	(0.40%)
Total VM	5.80 MB	(50.11%)

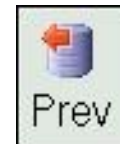
Process	Text	Data	Heap	Stack	Stack VM
Process 3: random.2	1989.25KB	109.18KB	3.84MB	43.29KB	48.00KB
lib-mx.so.2	81.51KB	2.08KB			
libc.so.6	1161.38KB	21.63KB			

Reverse Debugging

Replay Engine – The right way to debug



Step forward over functions



Step *backward* over functions



Step forward into functions



Step *backward* into functions



Advance forward out of current Function, after the call



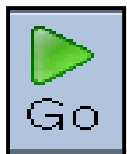
Advance backward out of current Function, to before the call



Advance forward to selected line



Advance backward to selected line



Run forward

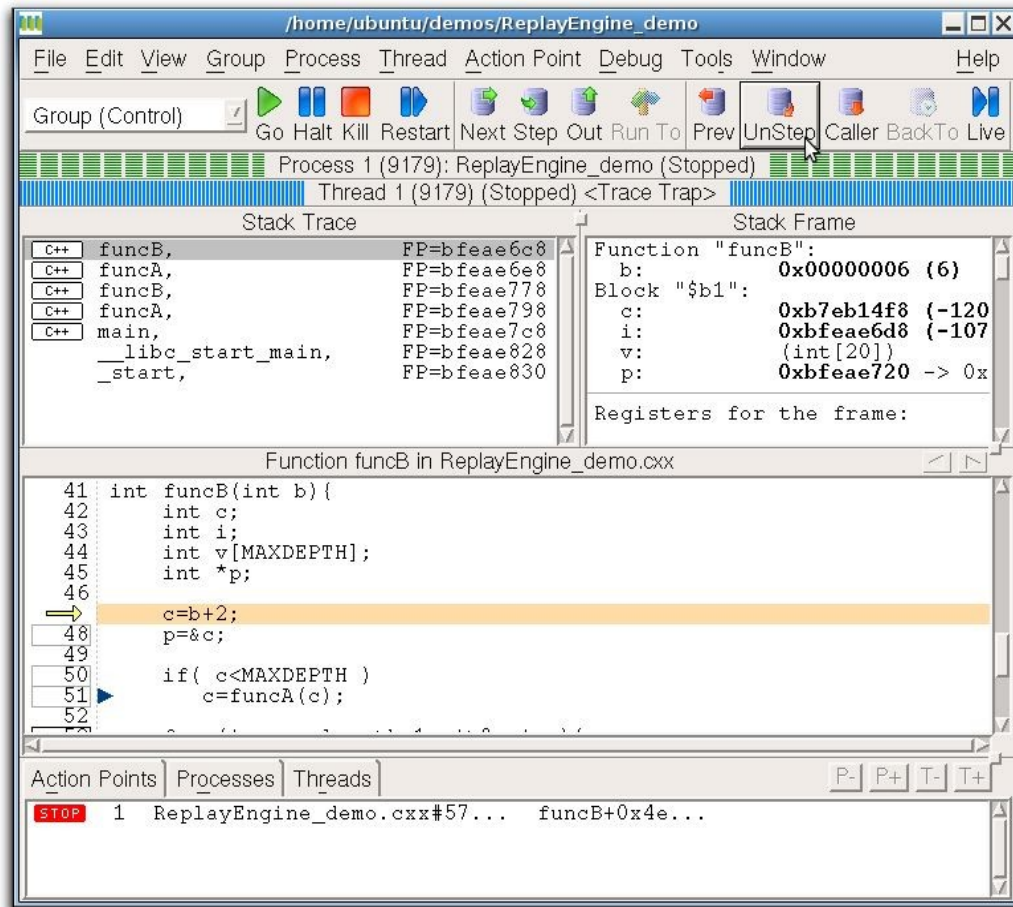
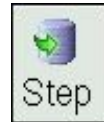
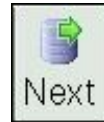


Run *backward*



Advance forward to “live” session

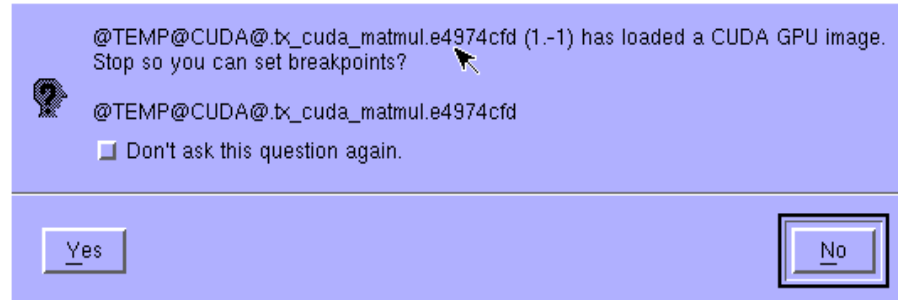
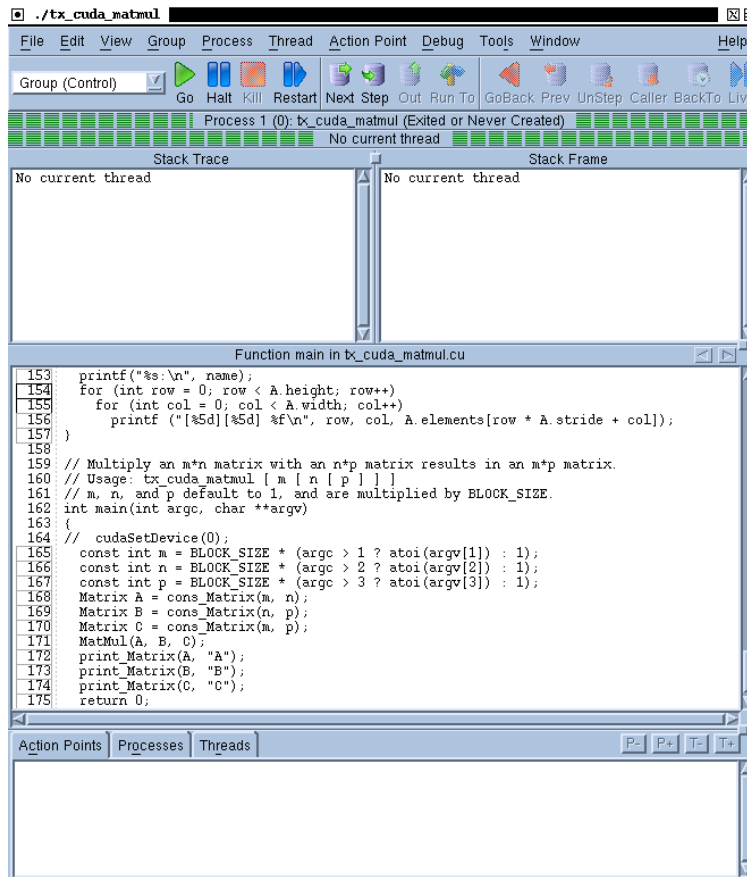
Replay Engine



- **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

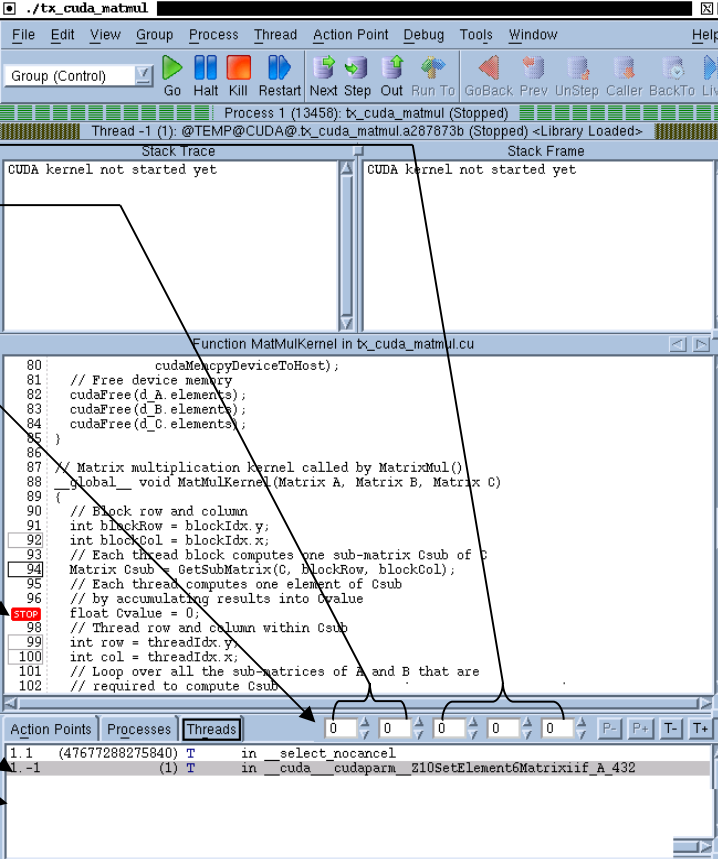
Starting TotalView



- 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

```

80  cudaMemcpyDeviceToHost);
81  // Free device memory
82  cudaFree(d_A.elements);
83  cudaFree(d_B.elements);
84  cudaFree(d_C.elements);
85  }
86
87  // Matrix multiplication kernel called by MatrixMul()
88  __global__ void MatMulKernel(Matrix A, Matrix B, Matrix C)
89  {
90  // Block row and column
91  int blockRow = blockIdx.y;
92  int blockCol = blockIdx.x;
93  // Each thread block computes one sub-matrix Csub of C
94  Matrix Csub = GetSubMatrix(C, blockRow, blockCol);
95  // Each thread computes one element of Csub
96  // by accumulating results into Cvalue
97  float Cvalue = 0;
98  // Thread row and column within Csub
99  int row = threadIdx.y;
100 int col = threadIdx.x;
101 // Loop over all the sub-matrices of A and B that are
102 // required to compute Csub
  
```

Action Points | Processes | Threads

1.1	(47677288275840)	T	in _select_nocancel
1.-1	(1)	T	in __cuda_cudaparam_210SetElementMatrixiif_A_432

Running to a Breakpoint in the GPU code

The screenshot shows the TotalView debugger interface with the following components:

- Thread List:** Shows Process 1 (12635) and Thread -1 at a breakpoint in the MatMulKernel.
- Stack Trace:** Shows the current function MatMulKernel and its caller.
- Function Details:** Provides metadata for MatMulKernel, including dimensions, lanes, warps, and registers. It also shows block and sub-block coordinates.
- Source Code:** Displays the C++ code for MatMulKernel, with a breakpoint set on line 94 (float Cvalue = 0;).
- Action Points:** A table at the bottom showing the current thread's state.

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

Stack backtrace and inlined functions

Parameter, register, local and shared variables

GPU focus thread logical coordinates

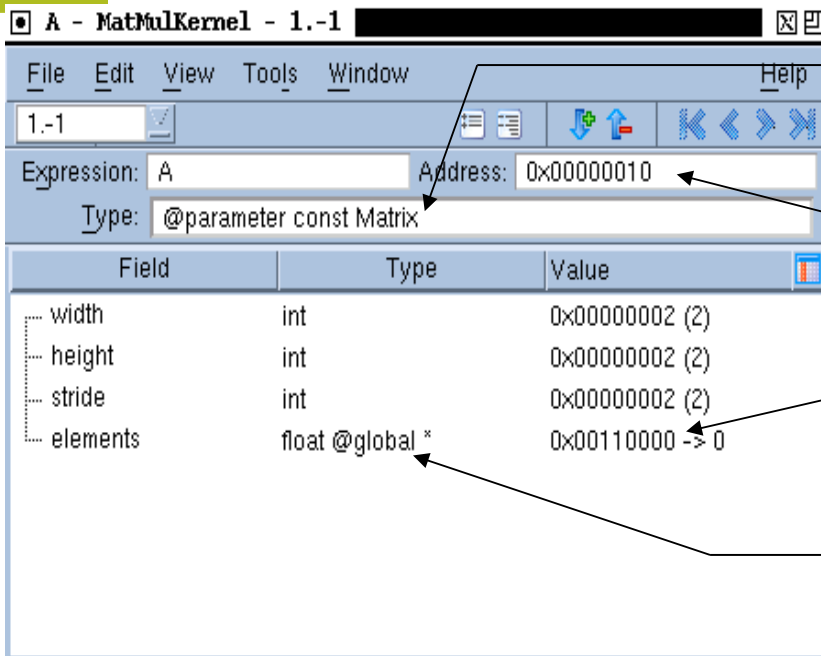
Dive on a variable name to open a variable window

PC arrow for the warp

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



The screenshot shows a debugger window titled "A - MatMulKernel - 1.-1". The "Expression" field contains "A" and the "Address" field contains "0x00000010". The "Type" field contains "@parameter const Matrix". Below this is a table with the following data:

Field	Type	Value
width	int	0x00000002 (2)
height	int	0x00000002 (2)
stride	int	0x00000002 (2)
elements	float @global *	0x00110000 -> 0

"@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