



Debugging Scalable MPI, Hybrid and/or Accelerated Applications with TotalView

**Extreme Scale Computing
Training Program**

August 2014

Chris Gottbrath



Agenda

- Introduction
- TotalView Debugger
- Demo
- Debugging MPI / OpenMP Hybrid Codes
- Memory Debugging
- Debugging Accelerators and Coprocessors
- Batch Debugging
- Reverse Debugging
- Running on ANL systems

Hybrid and Accelerated Applications

- What do we see
 - NVIDIA Tesla GP-GPU computational accelerators
 - Intel Xeon Phi Coprocessors
 - Complex memory hierarchies (numa, device vs host, etc)
 - Custom languages such as CUDA and OpenCL
 - Directive based programming such as OpenACC and OpenMP
 - Core and thread counts going up
- A lot of complexity to deal with if you want performance
 - C or Fortran with MPI starts to look “simple”
 - Everything is Multiple Languages / Parallel Paradigms
 - Up to 4 “kinds” of parallelism (cluster, thread, heterogeneous, vector)
 - Data movement and load balancing

How does Rogue Wave help?

TotalView debugger

- Troubleshooting and analysis tool
 - Visibility Into
 - Control Over
- Scalability
- Usability
- Advanced features/functionality
- Support for HPC platforms and languages

TotalView Overview

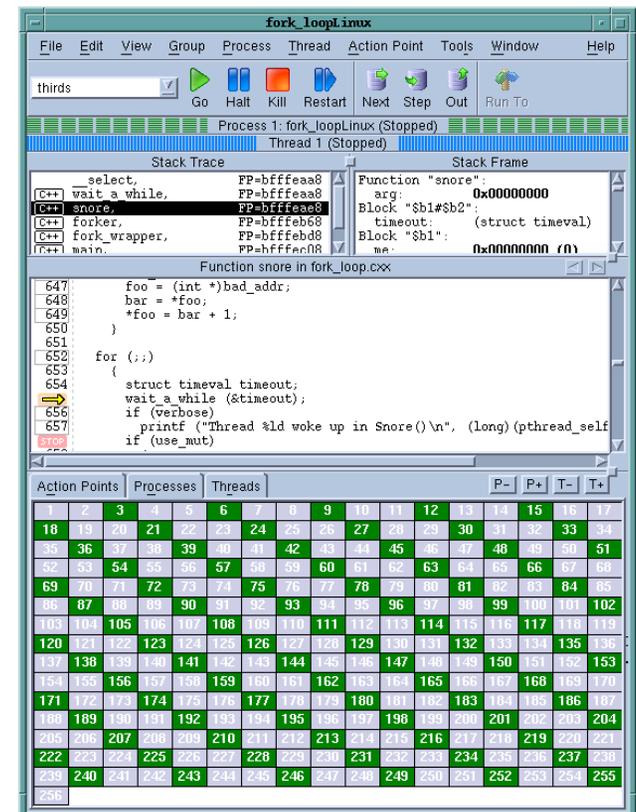
What is TotalView®?

Application Analysis and Debugging Tool: Code Confidently

- Debug and Analyse C/C++ and Fortran on Linux™, Unix or Mac OS X
- Laptops to supercomputers
- Makes developing, maintaining, and supporting critical apps easier and less risky

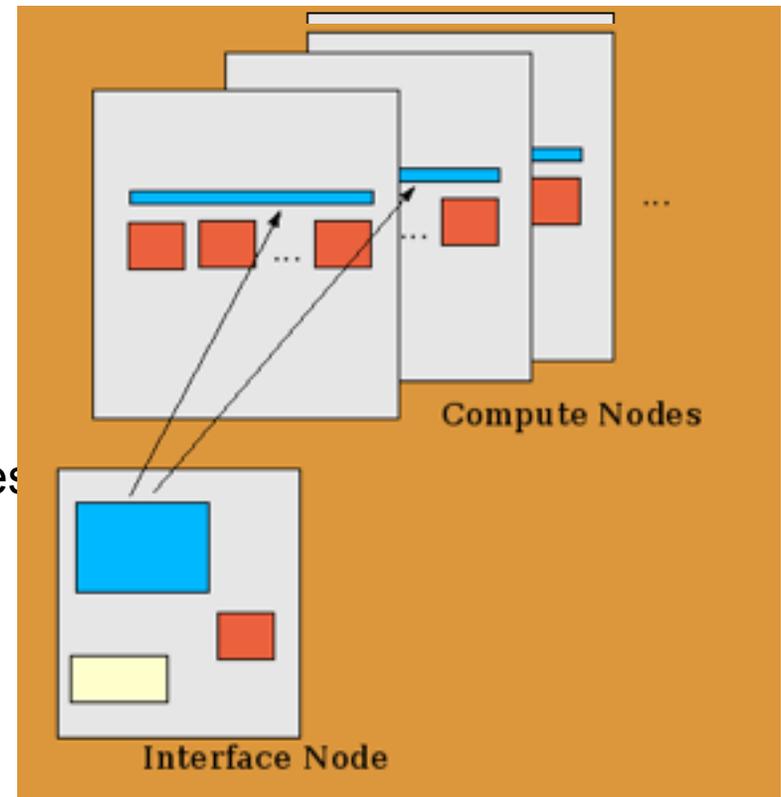
Major Features

- Easy to learn **graphical user interface** with data visualization
- **Parallel Debugging**
 - MPI, Pthreads, OpenMP™, GA, UPC
 - CUDA™, OpenACC®, and Intel® Xeon Phi™ coprocessor
- **Low** tool overhead resource usage
- Includes a **Remote Display Client** which frees you to work from anywhere
- **Memory Debugging** with MemoryScape™
- Deterministic **Replay Capability** Included on Linux/x86-64
- Non-interactive **Batch Debugging** with TVScript and the CLI
- **TTF & C++View** to transform user defined objects



Architecture for Cluster Debugging

- Single Front End (TotalView)
 - GUI
 - debug engine
- Debugger Agents (tvdsvr)
 - Low overhead, 1 per node
 - Traces multiple rank processes
- TotalView communicates directly with tvdsvrs
 - Not using MPI
 - Protocol optimization



Provides Robust, Scalable and efficient operation with Minimal Program Impact

What is new in 8.13 and 8.14

- 8.13 (Nov 2013)
 - CUDA 5.0 and 5.5
 - Dynamic Parallelism
 - Xeon Phi Symmetric
 - MemoryScape Xeon Phi support
 - Native and symmetric
 - OS X Mavericks
 - Performance
 - Setting breakpoints
 - Scalable dwhere & dstatus
 - Platform updates
- 8.14 (July 2014)
 - CUDA 6.0
 - Unified Memory
 - Early Access ReplayEngine Save/Load functionality (CLI)
 - STLView for unordered_X
 - GCC only, for now
 - Unordered set/multiset & map/multimap
 - Performance improvements
 - Startup performance
 - Complex C++ codes
 - Handling dlopen()
 - Platform updates

Multi-phase R&D Projects Underway

- Massive Scalability
 - Collaboration with LLNL and Tri-lab partners
 - Targeting Cray, Blue Gene and Linux Clusters
- Shiny new GUI
 - Sleek, Modern and Fast
 - Configurable
 - Improved Usability
 - Provides aggregation capabilities for big data and scale
 - Leveraging math and stat expertise from IMSL
- Working with customers through early access programs
 - Customer input is key to the success of both programs

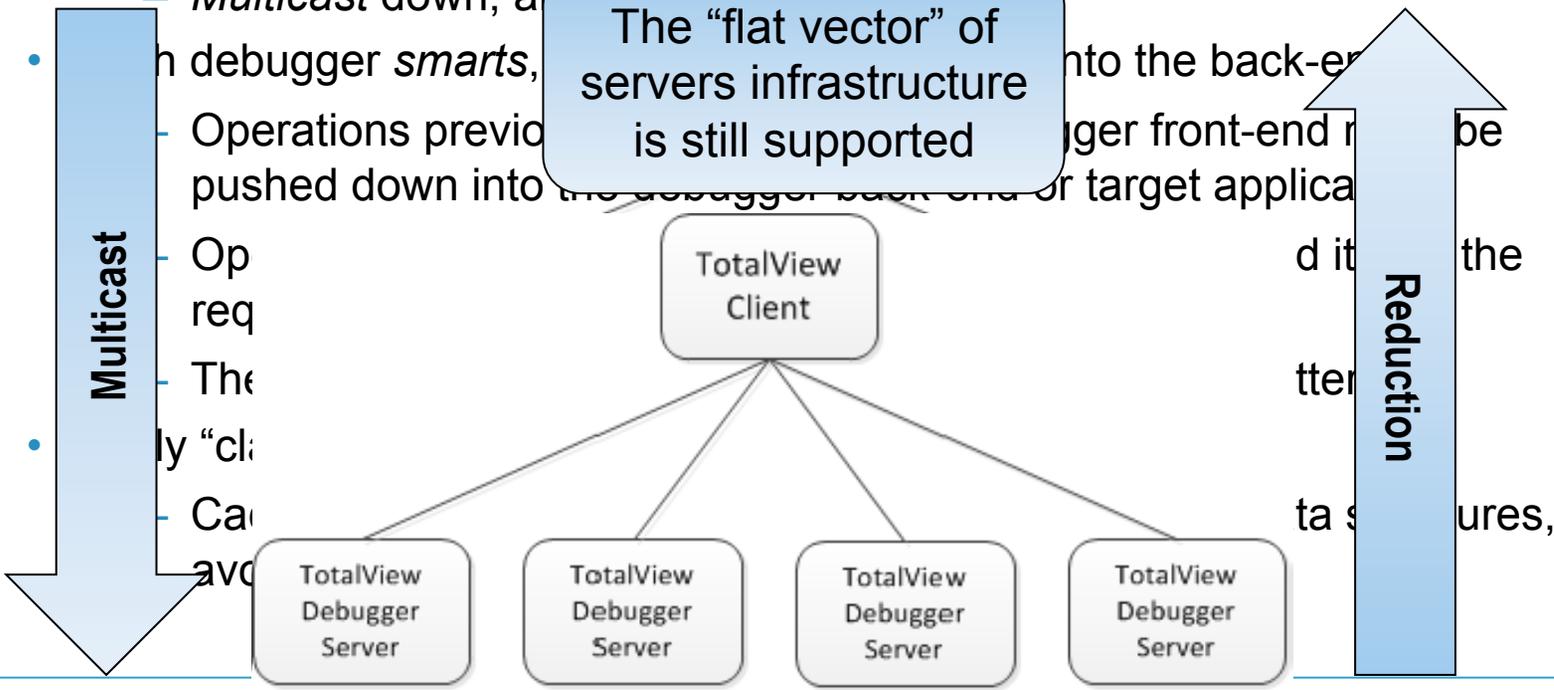
TotalView Infrastructure Scalability Strategy

- Implement an additional layer of servers using MRNet
- Parallelize debugger operations
 - Convert *iteration*

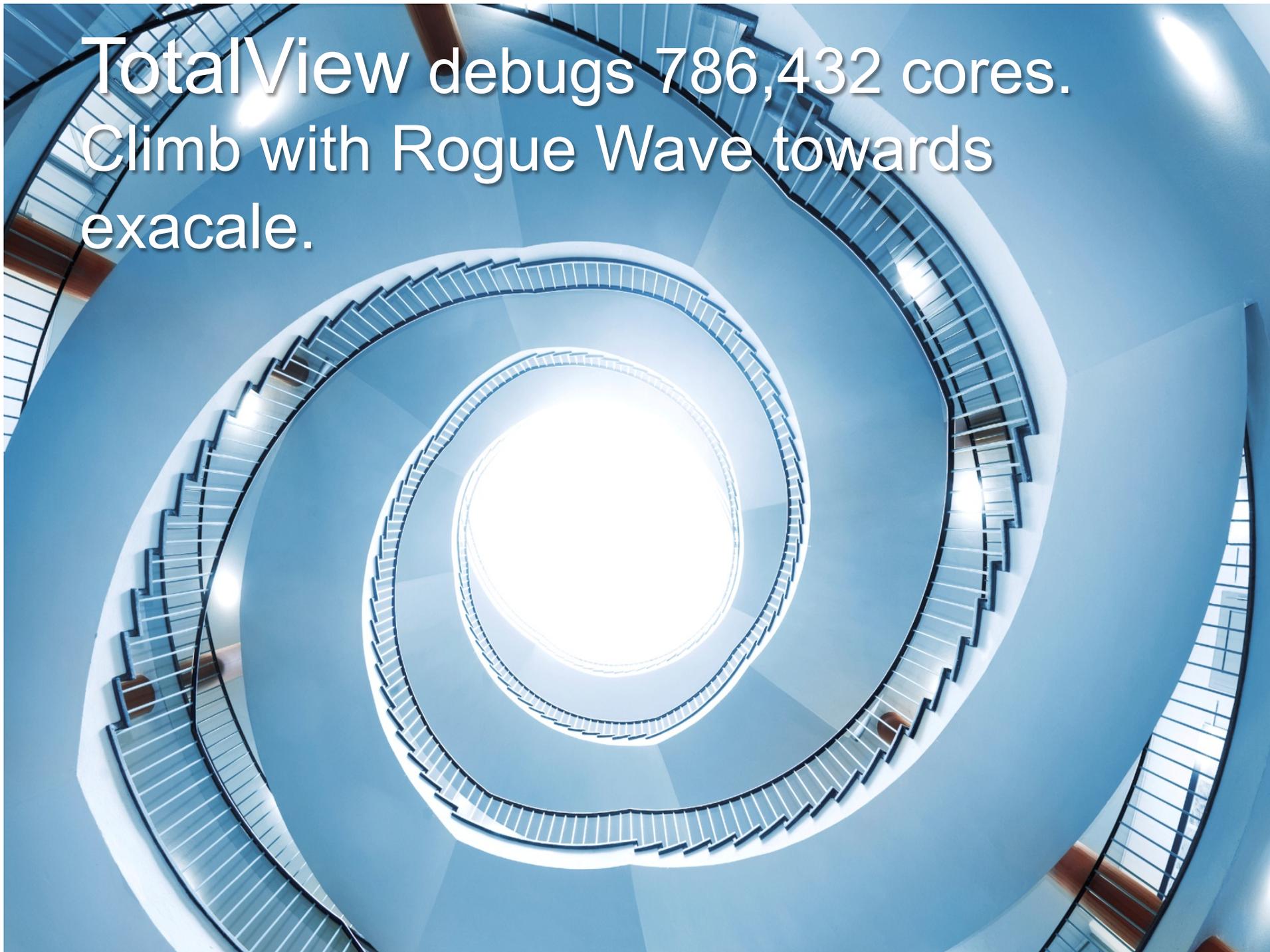
The “MRNet tree” of servers infrastructure has been added

- *Multicast* down, and
- When debugger *smarts*, operations previously pushed down into the debugger back-end or target application

The “flat vector” of servers infrastructure is still supported

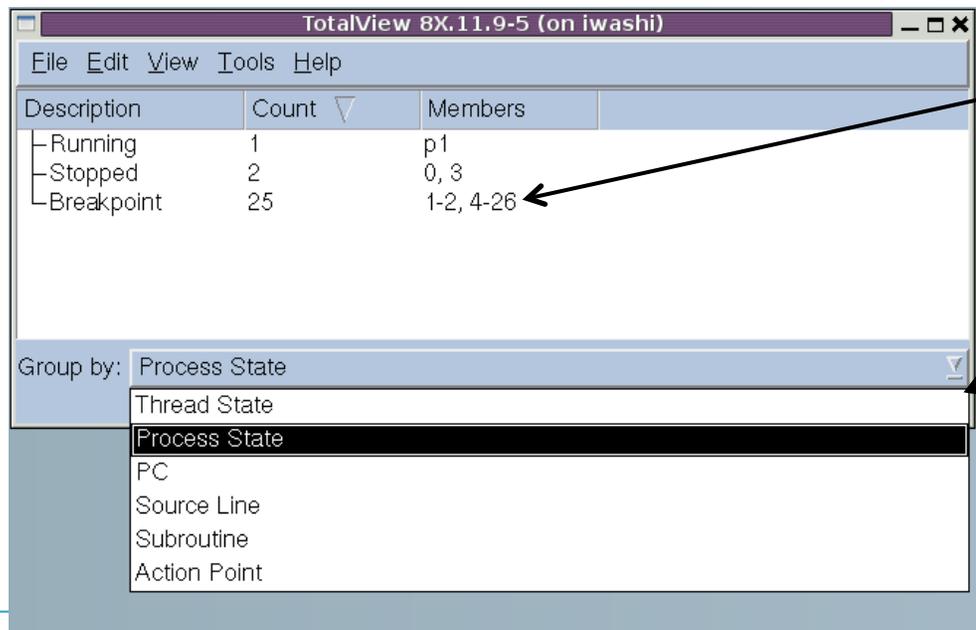


TotalView debugs 786,432 cores.
Climb with Rogue Wave towards
exacale.



New-Style Root Window (SEA2+)

- A prototype new-style root window w/ “-demo_ui”
- Displays aggregated program information
- Intended to eventually replace the old-style root window
- Menu items that are not yet implemented are disabled



- Diving selects a representative of the group and refocuses the process window
- Current aggregations
- Hierarchical groupings planned

Compressed *ptlist* Syntax

- Aggregation requires a compact process/thread set representation (for both CLI *and* GUI output)
- General syntax of a *ptlist*

ptlist : *pcount* ':' *tcount* '[' *ptrange* [',' *ptrange*] ... '['

ptrange : *prange* '.' *trange*

prange : *rank* ['-' *rank*]

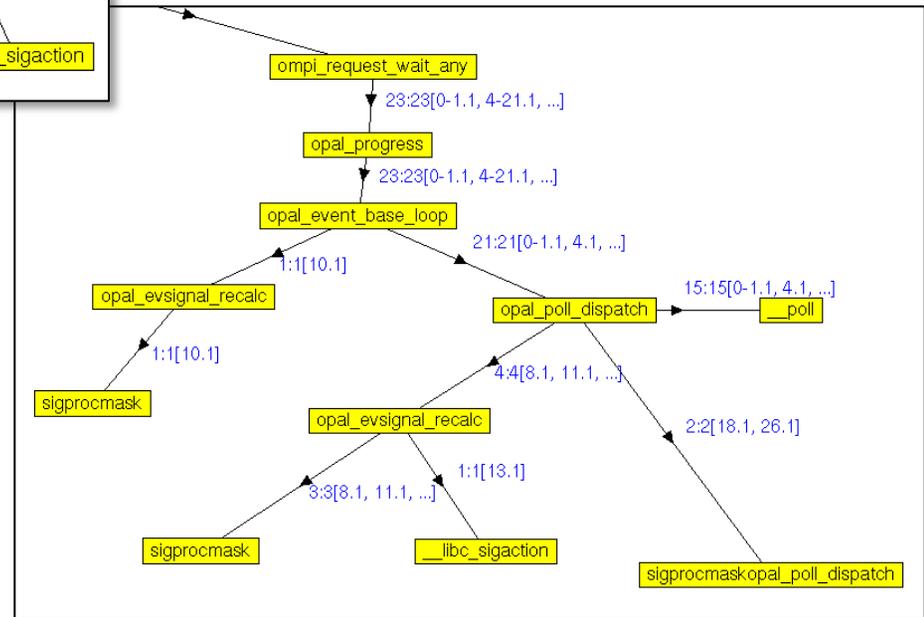
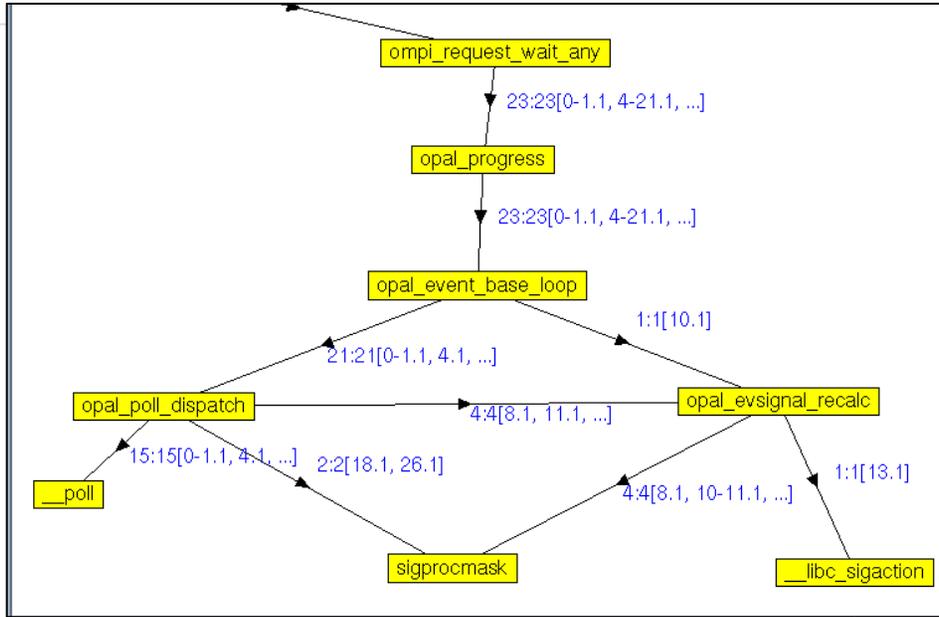
| 'p' *dpid* ['-' *dpid*]

trange : *dtid* ['-' *dtid*]

- Inspired by STAT and previous TotalView implementations
- Example

28:28[0-26.1, p1.1]

Call Graph vs. Call Tree (SEA3+)



TotalView Scalable Early Access Summary

Please give it try!

- We value your feedback
- Enable MRNet and the demo UI
 - `totalview -mrnet -demo_ui ...`
- Many infrastructure changes are in place already
 - Though not all operations parallelized yet
- User interface changes in prototype phase
 - More improvements coming in existing UI
 - Remaining improvements coming in new UI

- Questions?

Demo

Debugging Hybrid MPI + OMP codes

Process Window Overview

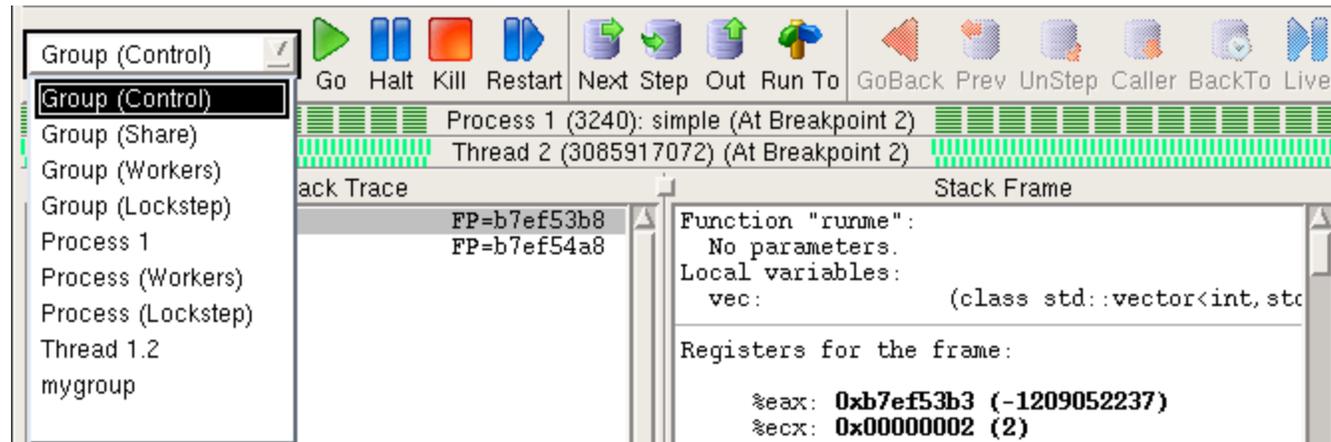
The screenshot shows the Process Window interface with the following components labeled:

- Toolbar:** Located at the top, containing icons for Go, Halt, Kill, Restart, Next Step, Out, Run, To, GoBack, Prev, UnStep, Caller, and BackTo Live.
- Stack Trace Pane:** Located on the left side of the middle section, showing a list of stack frames including Circle::area, Circle::Circle, arrays, and main.
- Stack Frame Pane:** Located on the right side of the middle section, showing details for the selected frame: Function "Circle::area", this: 0xbf8ffba0, Block "b1", result: 0, and Registers for the frame.
- Source Pane:** Located in the bottom middle section, displaying the source code for the function Circle::area in combined.cxx. Line 430 is highlighted in yellow.
- Tabbed Area:** Located at the bottom, showing a list of action points with columns for file, line, and function name.

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

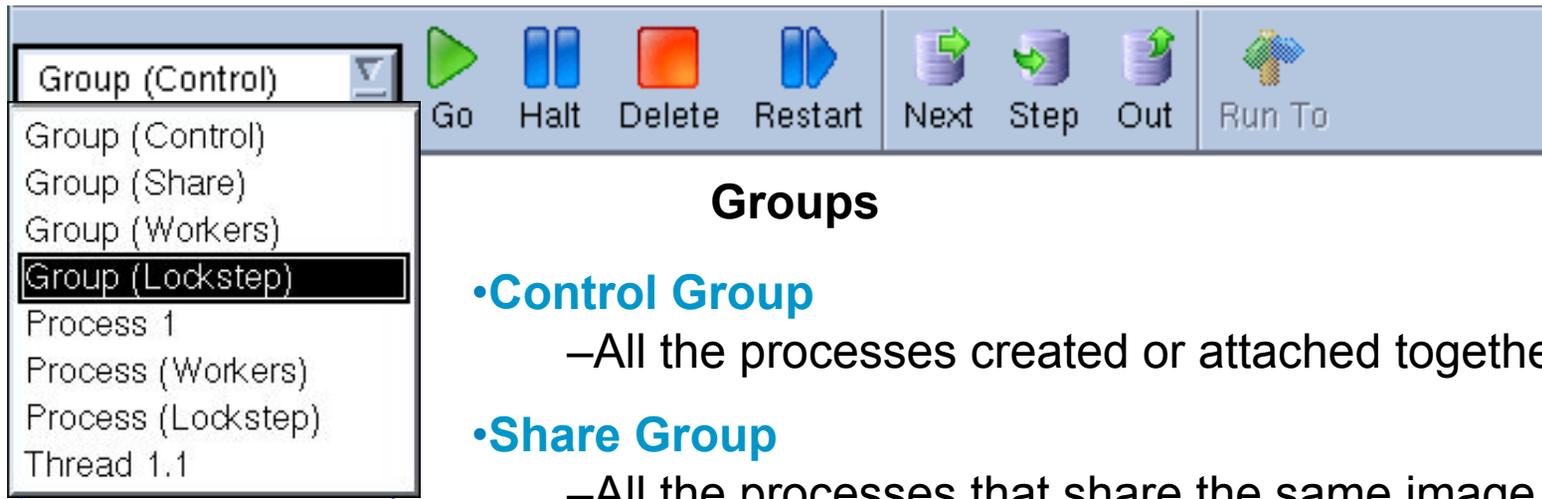


Group	Process	Thread	Action
Go			Shift+G
Halt			Shift+H
Next			Shift+N
Step			Shift+S
Out			Shift+O
Run To			Shift+R
Next Instruction			Shift+X
Step Instruction			Shift+I
Hold			
Release			
Attach Subset...			
Detach			
Custom Groups...			
Restart			
Kill			Ctrl+Z

Process	Thread	Action Point
Go		g
Halt		h
Next		n
Step		s
Out		o
Run To		r
Next Instruction		x
Step Instruction		i
Hold		w
Hold Threads		
Release Threads		
Create		
Detach		
Startup Parameters...		Ctrl+A

Thread	Action Point	Debug
Go		
Halt		
Next		
Step		
Out		
Run To		
Next Instruction		
Step Instruction		
Set PC		p
Hold		
Continuation Signal...		

Basic Process Control



Groups

- **Control Group**

- All the processes created or attached together

- **Share Group**

- All the processes that share the same image

- **Workers Group**

- All the threads that are not recognized as manager or service threads

- **Lockstep Group**

- All threads at the same PC

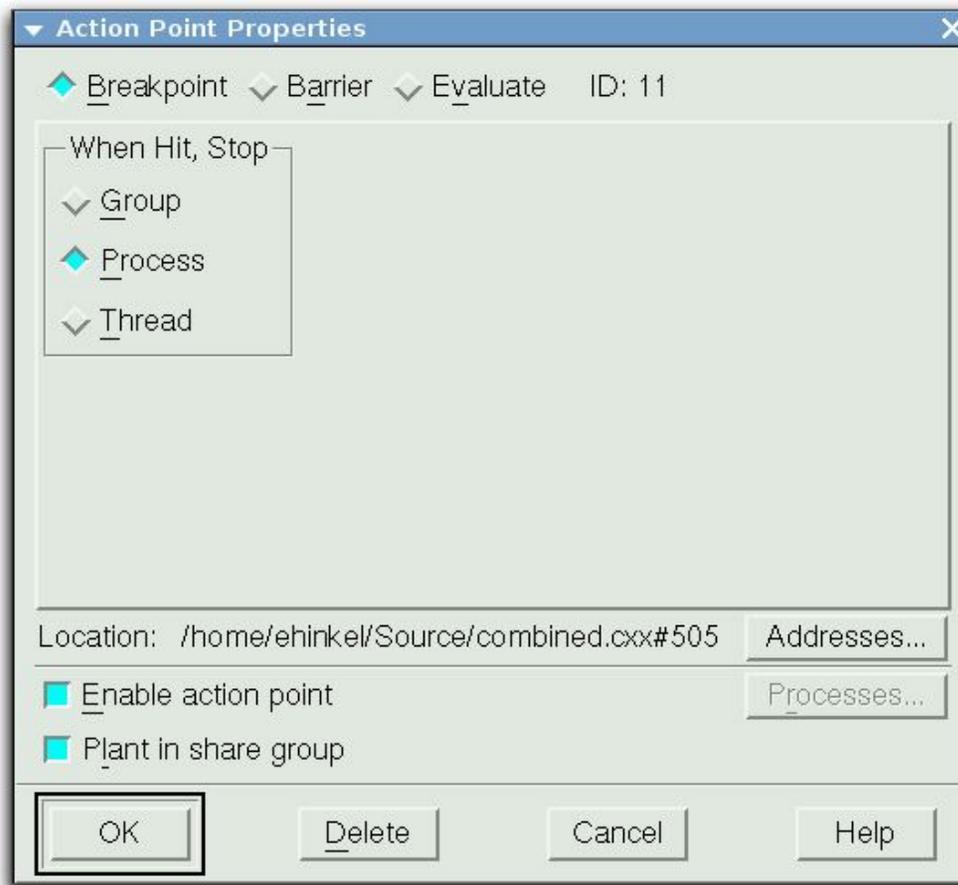
- **Process, Process (Workers), Process (Lockstep)**

- All process members as above

- **User Defined Group**

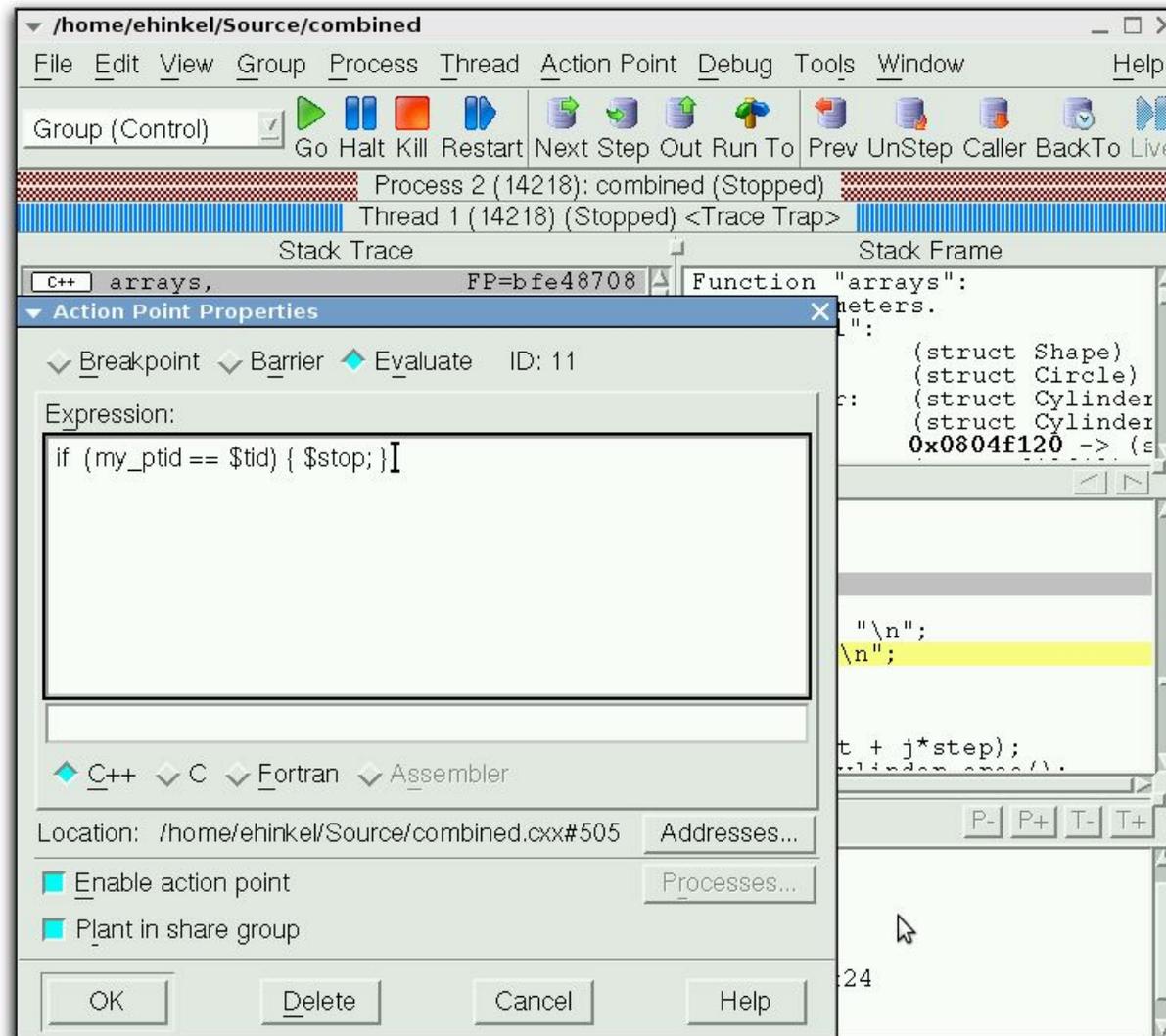
- Process group defined in Custom Groups dialog

Setting Breakpoints



- Breakpoint type
- What to stop
- Set conditions
- Enable/disable
- In 1 process or share group

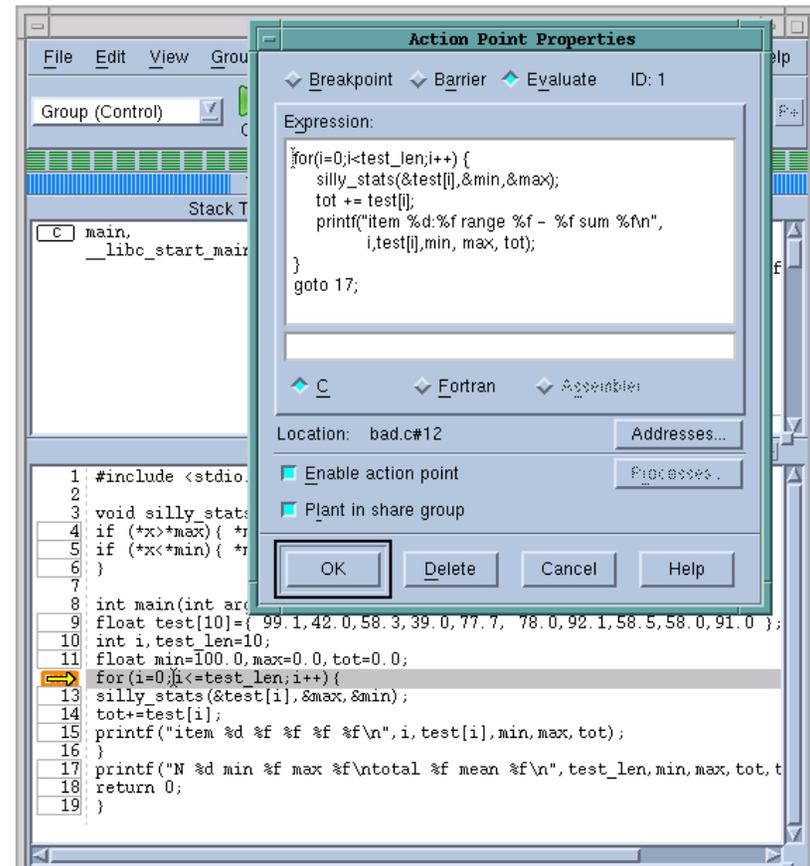
Conditional Breakpoint



Evaluation Breakpoint...

Test Fixes on the Fly!

- Test small source code patches
- Call functions
- Set variables
- Test conditions
- C/C++ or Fortran
- Can't use C++ constructors
- Use program variables
- ReplayEngine records changes but won't step through them



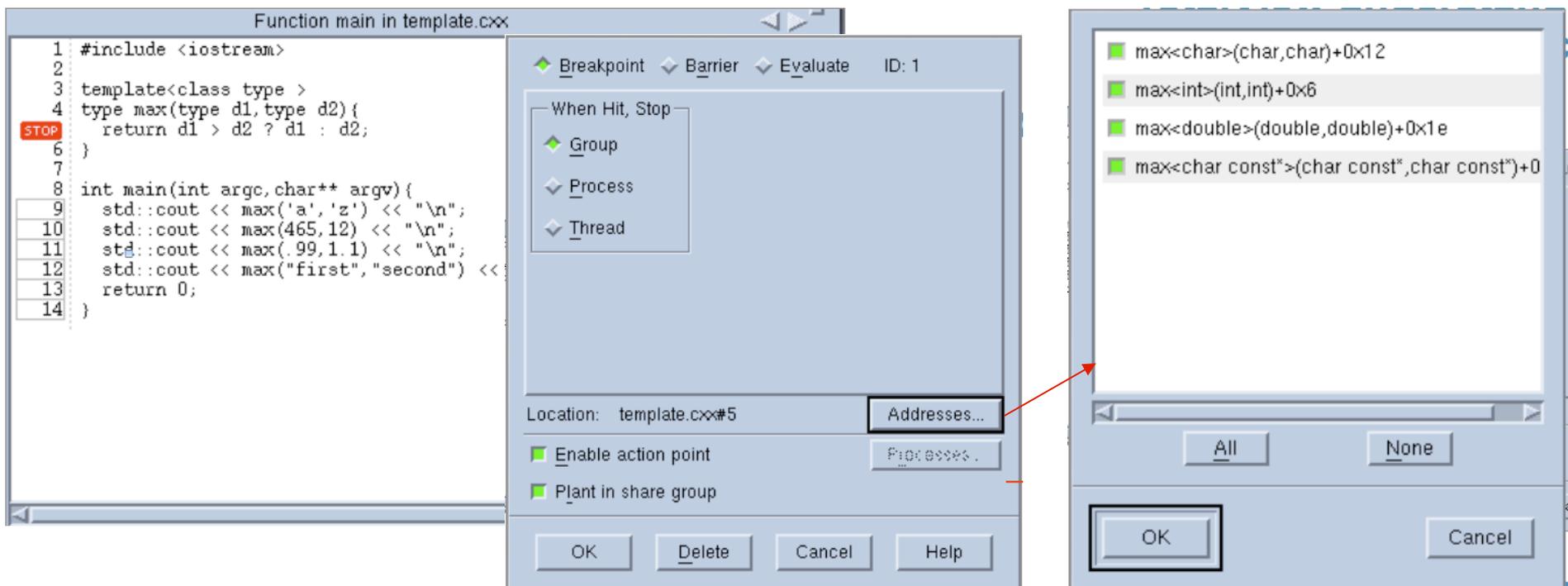
```

item 0:99.099998 range 99.099998 - 99.099998 sum 99.099998
item 1:42.000000 range 42.000000 - 99.099998 sum 141.100006
item 2:58.299999 range 42.000000 - 99.099998 sum 199.400009
item 3:39.000000 range 39.000000 - 99.099998 sum 238.400009
item 4:77.699997 range 39.000000 - 99.099998 sum 316.100006
item 5:78.000000 range 39.000000 - 99.099998 sum 394.100006
item 6:92.099998 range 39.000000 - 99.099998 sum 486.200012
item 7:58.500000 range 39.000000 - 99.099998 sum 544.700012
item 8:58.000000 range 39.000000 - 99.099998 sum 602.700012
item 9:91.000000 range 39.000000 - 99.099998 sum 693.700012
N 10 min 39.000000 max 99.099998
total 693.700012 mean 69.370001

```

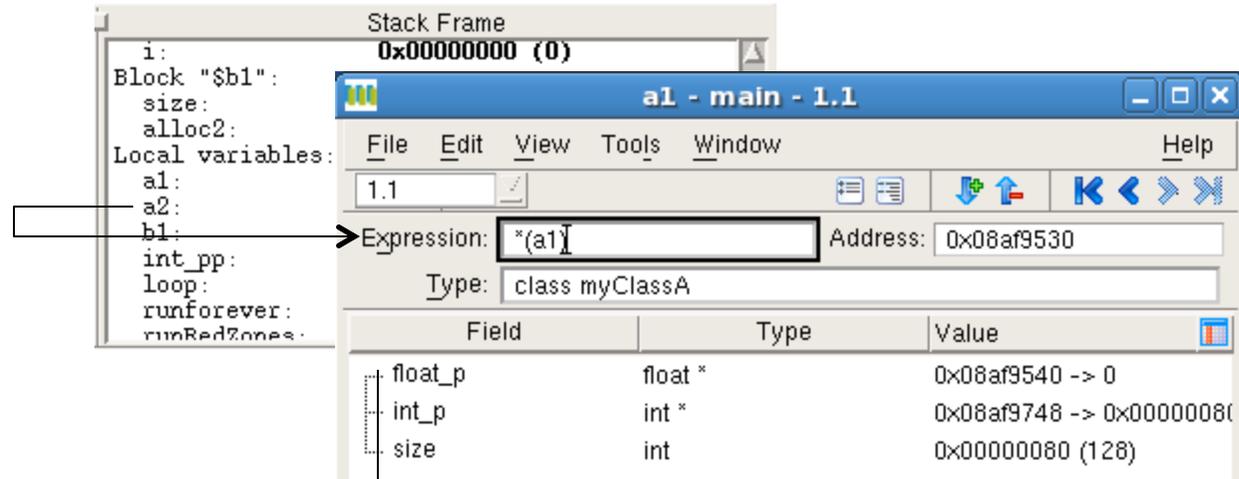
Setting Breakpoints With C++ Templates

TotalView understands C++ templates and gives you a choice ...



Boxes with solid lines around line numbers indicate code that exists at more than one location.

Diving



Stack Frame

```
i: 0x00000000 (0)
Block "$b1":
size:
alloc2:
Local variables:
a1:
a2:
b1:
int_pp:
loop:
runForever:
runRedZones:
```

a1 - main - 1.1

File Edit View Tools Window Help

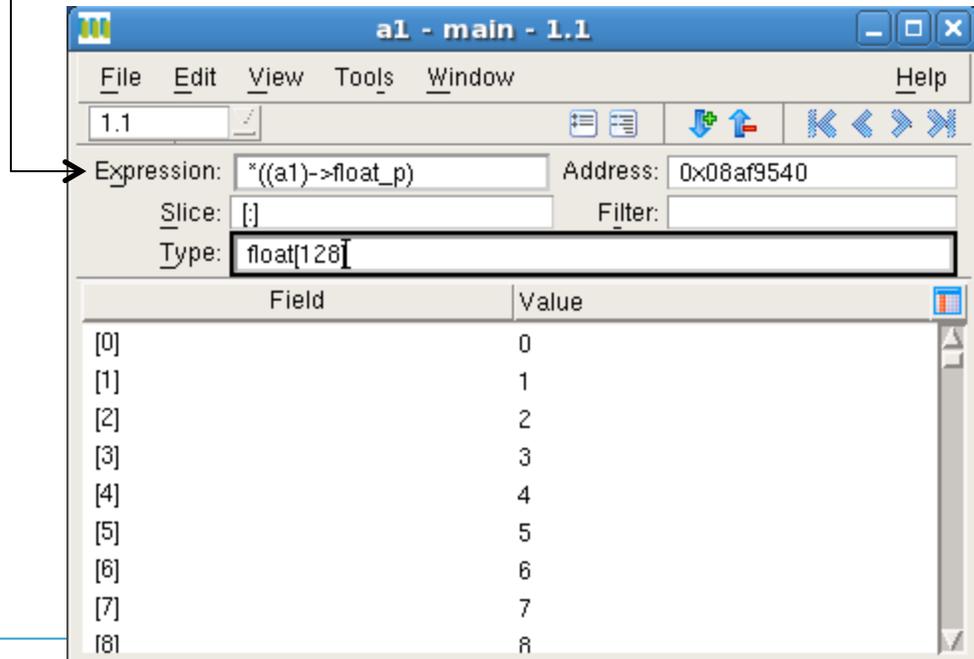
1.1

Expression: `*(a1)` Address: 0x08af9530

Type: class myClassA

Field	Type	Value
float_p	float *	0x08af9540 -> 0
int_p	int *	0x08af9748 -> 0x00000008
size	int	0x00000080 (128)

Diving on a Common Block in the Stack Frame Pane



a1 - main - 1.1

File Edit View Tools Window Help

1.1

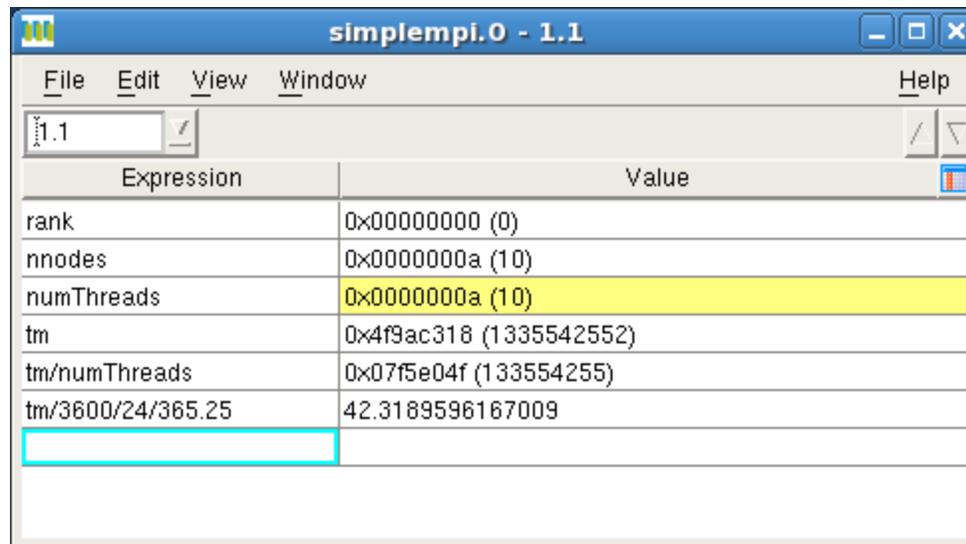
Expression: `*((a1)->float_p)` Address: 0x08af9540

Slice: `[]` Filter:

Type: float[128]

Field	Value
[0]	0
[1]	1
[2]	2
[3]	3
[4]	4
[5]	5
[6]	6
[7]	7
[8]	8

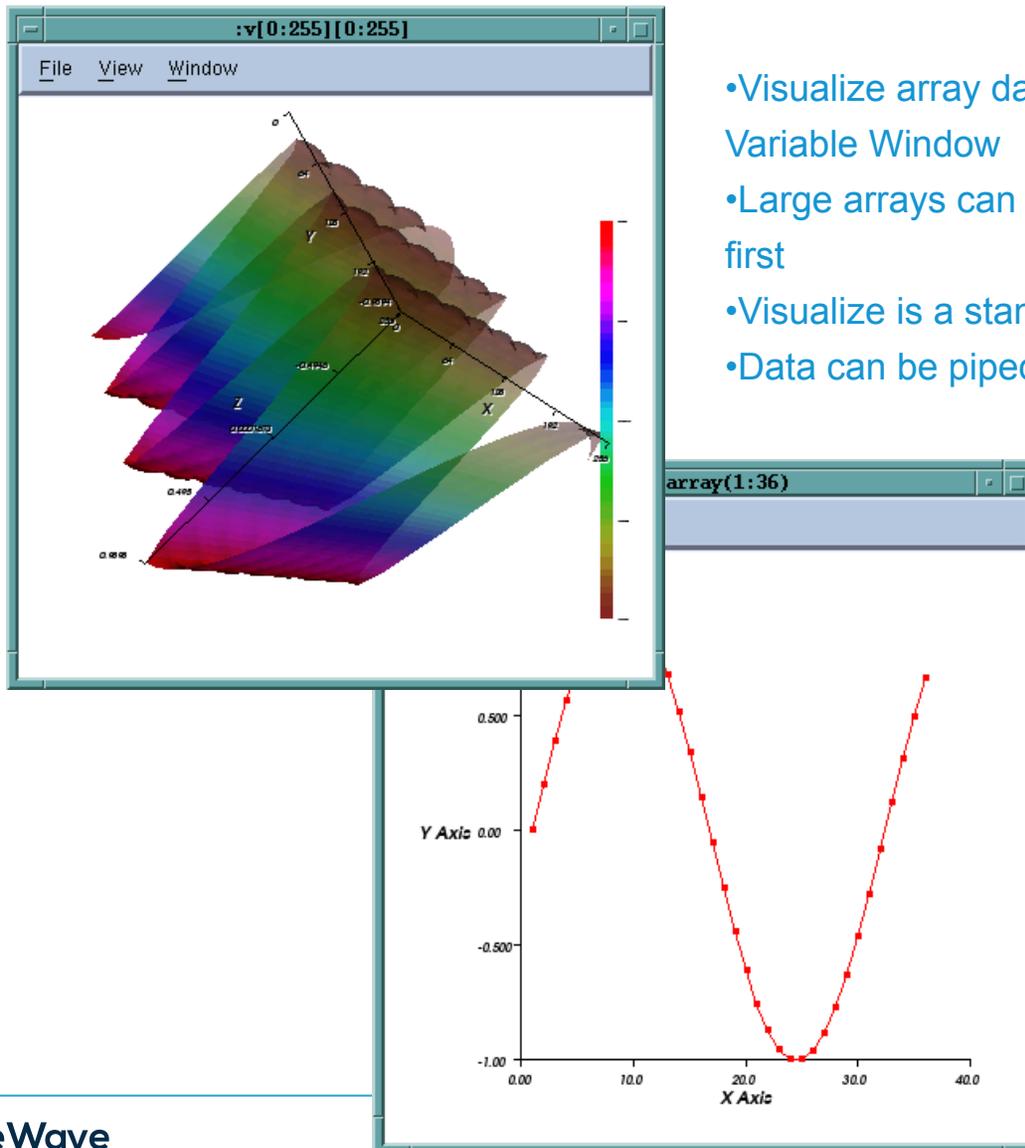
Expression List Window



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

Visualizing Arrays



- Visualize array data using Tools > Visualize from the Variable Window
- Large arrays can be sliced down to a reasonable size first
- Visualize is a standalone program
- Data can be piped out to other visualization tools

- Visualize allows to spin, zoom, etc.
- Data is not updated with Variable Window; You must revisualize
- `$visualize()` is a directive in the expression system, and can be used in evaluation point expressions.

Array Viewer

- Variable Window select Tools -> Array Viewer
- View 2 dimensions of data

The screenshot shows the 'Array Viewer' window with the following details:

- Expression: *((a1)->float_p)
- Type: float[8][16]
- Modify array slice table:

	Dimension	Start Index	End Index	Stride
Row	[i]	0	7	1
Column	[j]	0	15	1
- Format: Automatic
- Slice: [0:7:1][0:15:1]
- Array data table:

	[j]:0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
[i]:0	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
2	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
3	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
4	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
5	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
6	96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111
7	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127

Dive in All

Dive in All will display an element in an array of structures as if it were a simple array.

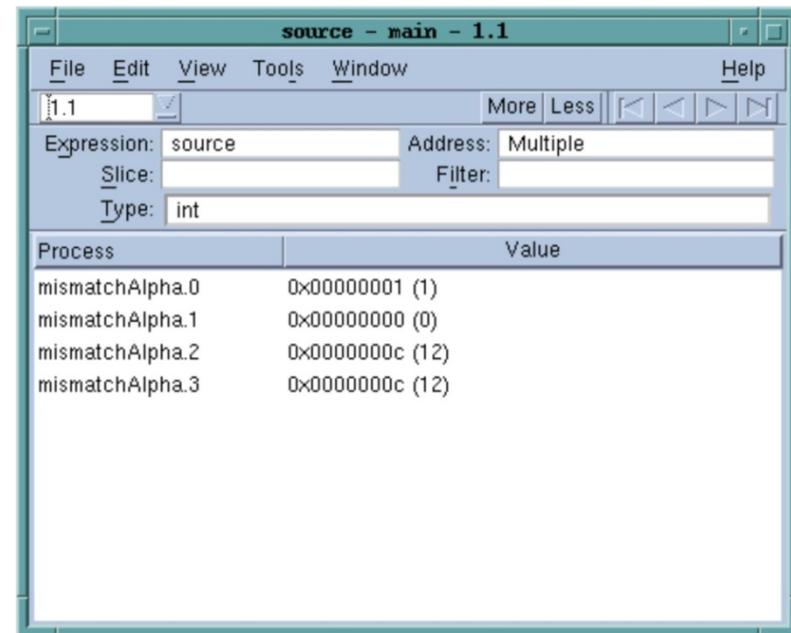
The image shows three overlapping debugger windows from RogueWave. The top window shows an array of structures: `*(array)` at address `0x092920a0`, type `struct compound_t[20]`, with fields `a` (int), `b` (int), and `c` (float). The middle window shows the same array expanded, with the field `b` selected and a context menu open. The bottom window shows the result of the `Dive in All` action: a simple integer array `array[:].x.b` at address `0x092920a0`, type `int[20]`, with values ranging from `0x00000000` to `0x00000026`.

Field	Type	Value
[0]	struct compound_t	(Struct)
[1]	struct compound_t	(Struct)
[2]	struct compound_t	(Struct)
[3]	struct compound_t	(Struct)
[4]	struct compound_t	(Struct)
[5]	struct compound_t	(Struct)
[6]	struct compound_t	(Struct)
[7]	struct compound_t	(Struct)
[8]	struct compound_t	(Struct)
[9]	struct compound_t	(Struct)
[10]	struct compound_t	(Struct)
[11]	struct compound_t	(Struct)
[12]	struct compound_t	(Struct)
[13]	struct compound_t	(Struct)
[14]	struct compound_t	(Struct)
[15]	struct compound_t	(Struct)
[16]	struct compound_t	(Struct)
[17]	struct compound_t	(Struct)
[18]	struct compound_t	(Struct)
[19]	struct compound_t	(Struct)

Field	Type	Value
[0]	int	0x00000000 (0)
[1]	int	0x00000002 (2)
[2]	int	0x00000004 (4)
[3]	int	0x00000006 (6)
[4]	int	0x00000008 (8)
[5]	int	0x0000000a (10)
[6]	int	0x0000000c (12)
[7]	int	0x0000000e (14)
[8]	int	0x00000010 (16)
[9]	int	0x00000012 (18)
[10]	int	0x00000014 (20)
[11]	int	0x00000016 (22)
[12]	int	0x00000018 (24)
[13]	int	0x0000001a (26)
[14]	int	0x0000001c (28)
[15]	int	0x0000001e (30)
[16]	int	0x00000020 (32)
[17]	int	0x00000022 (34)
[18]	int	0x00000024 (36)
[19]	int	0x00000026 (38)

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.



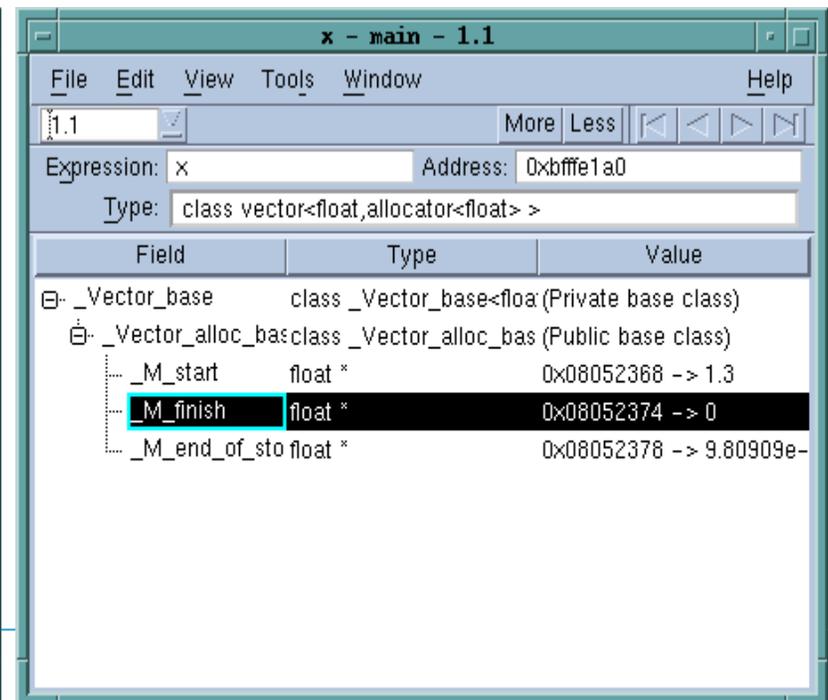
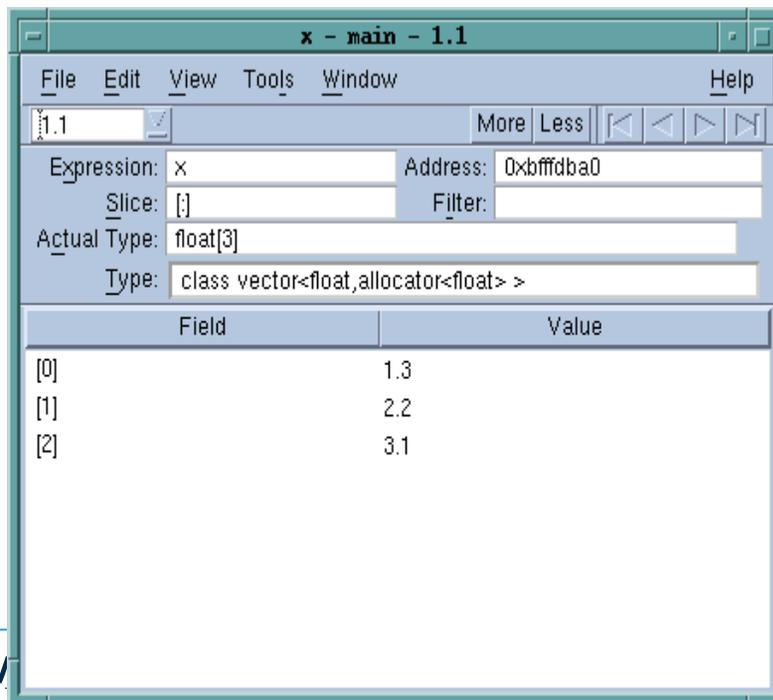
The screenshot shows a TotalView window titled "source - main - 1.1". The window has a menu bar with "File", "Edit", "View", "Tools", "Window", and "Help". Below the menu bar is a toolbar with "More", "Less", and navigation arrows. The main area contains a table with the following data:

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

STLView

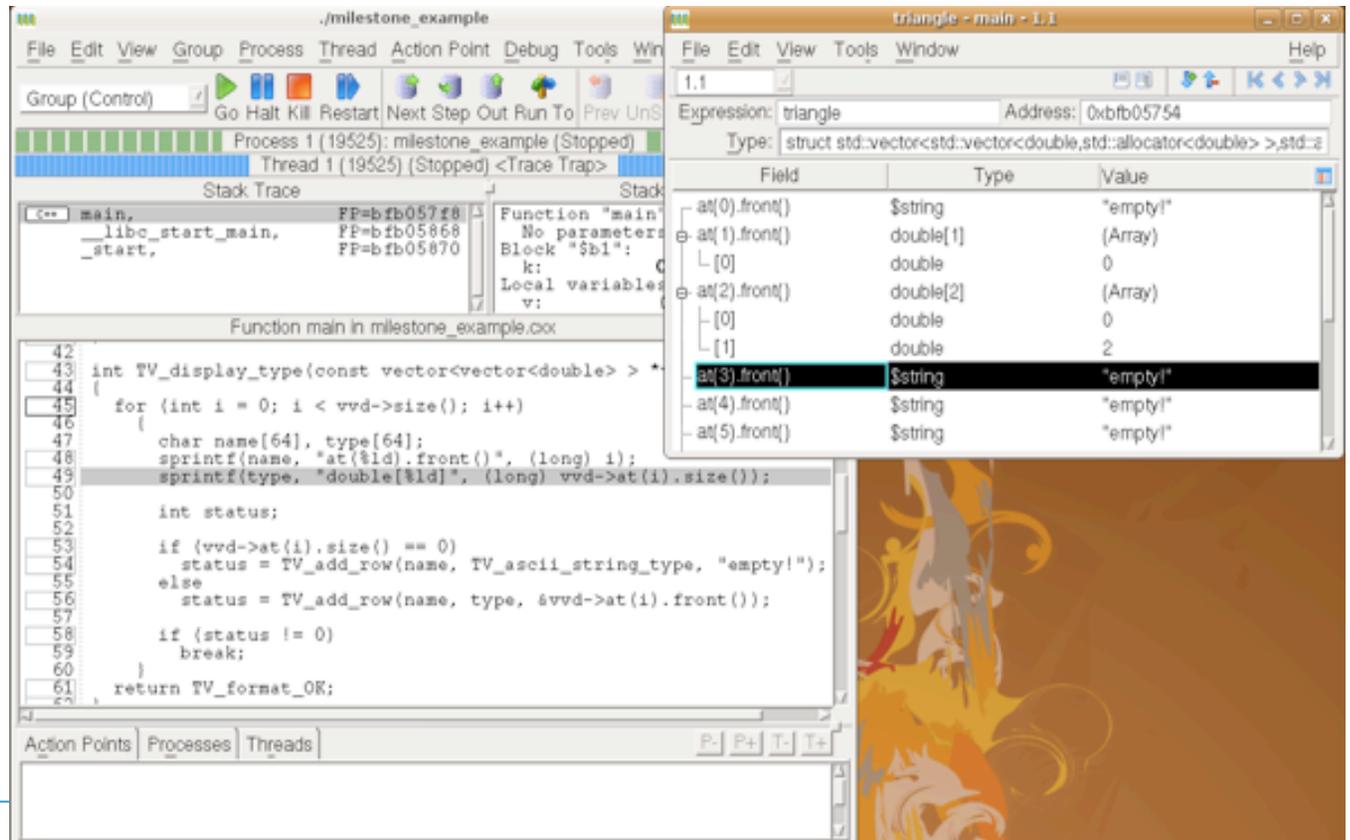
STLView transforms templates into readable and understandable information

- STLView supports `std::vector`, `std::list`, `std::map`, `std::string`
- See doc for which STL implementations are supported



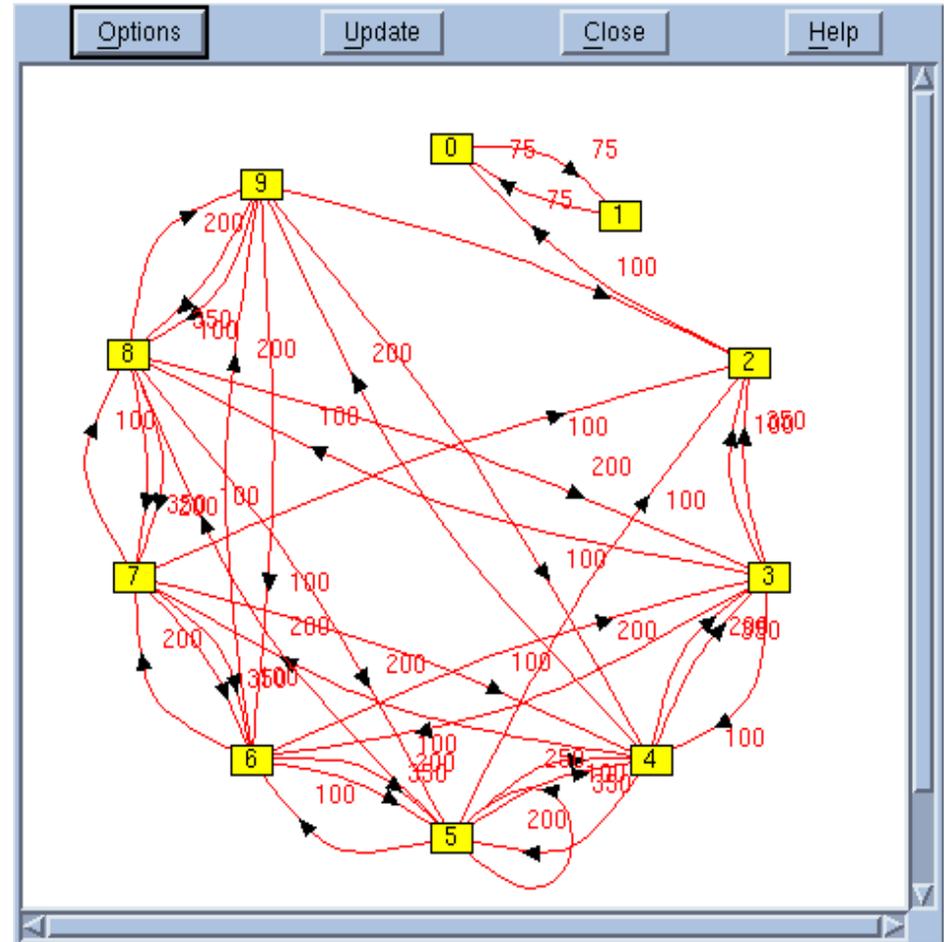
C++View

- C++View is a simple way for you to define type transformations
 - Simplify complex data
 - Aggregate and summarize
 - Check validity
- Transforms
 - Type-based
 - Compose-able
 - Automatically visible
- Code
 - C++
 - Easy to write
 - Resides in target
 - Only called by TotalView



Message Queue Graph

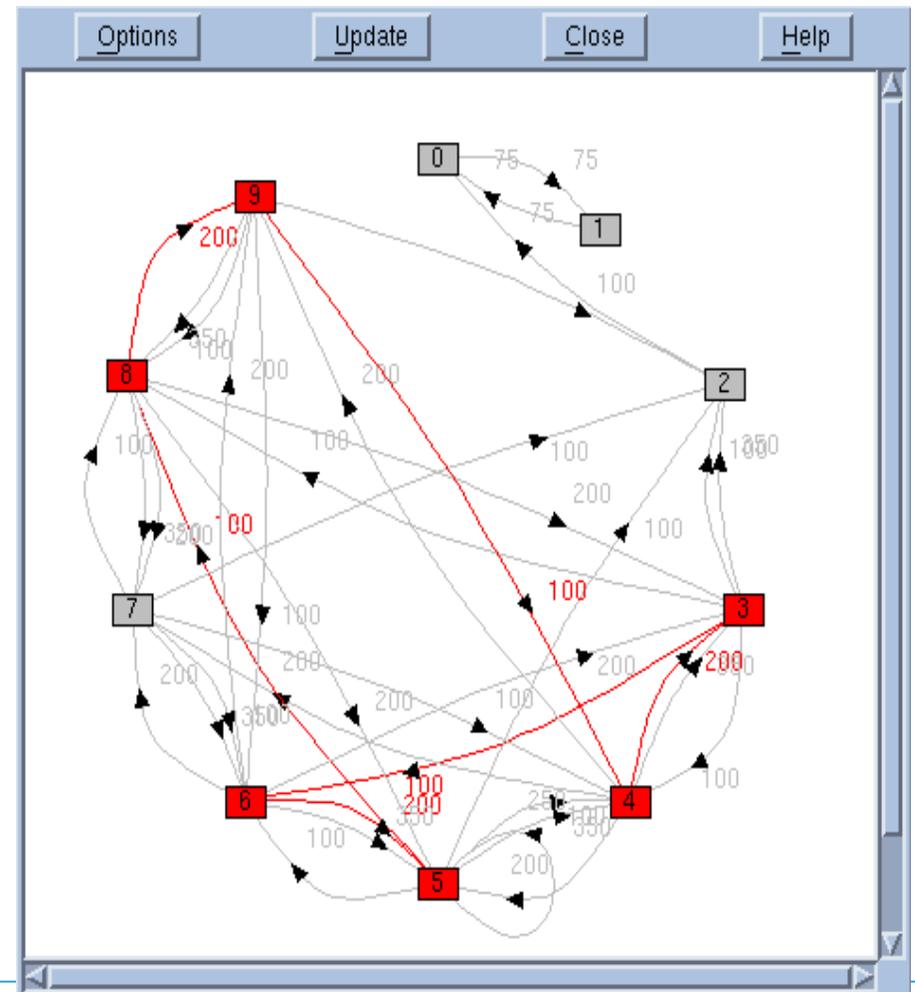
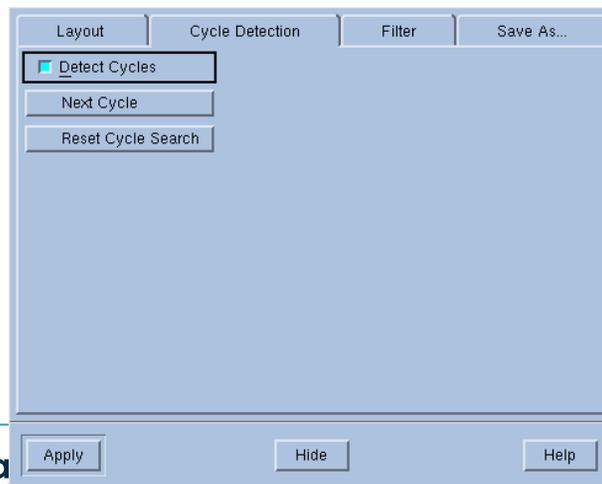
- Hangs & Deadlocks
- Pending Messages
 - Receives
 - Sends
 - Unexpected
- Inspect
 - Individual entries
- Patterns



Message Queue Graph

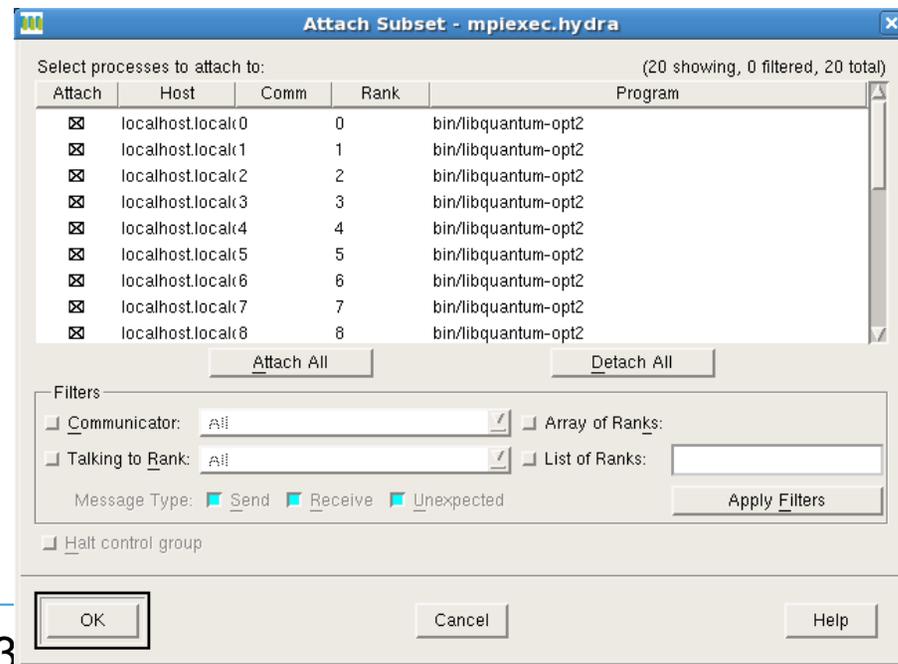
Message Queue Debugging

- **Filtering**
 - Tags
 - MPI Communicators
- **Cycle detection**
 - Find deadlocks



Subset Attach

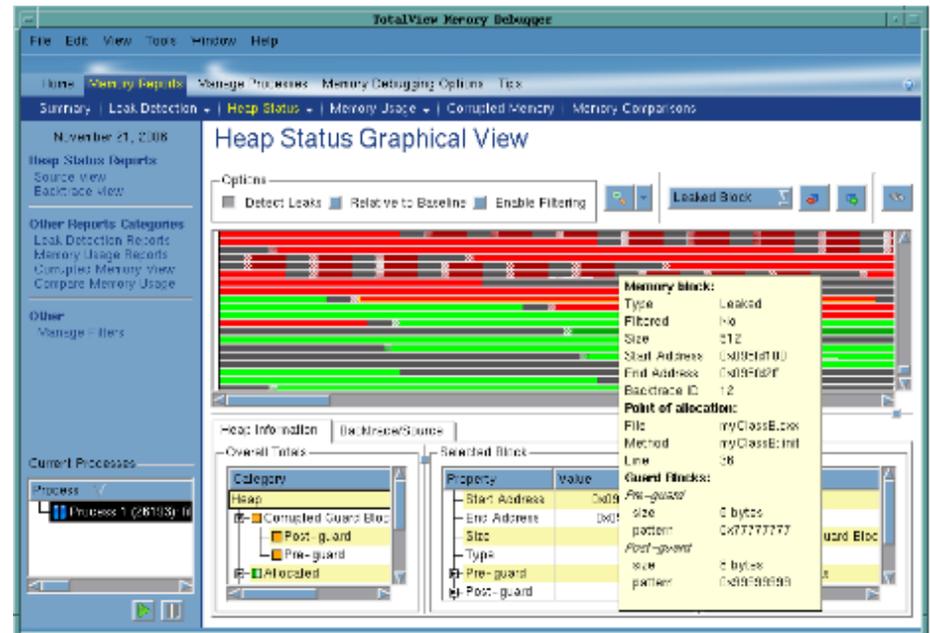
- Connecting to a subset of a job reduces tokens and overhead
- Can change this during a run
- Groups->Subset Attach



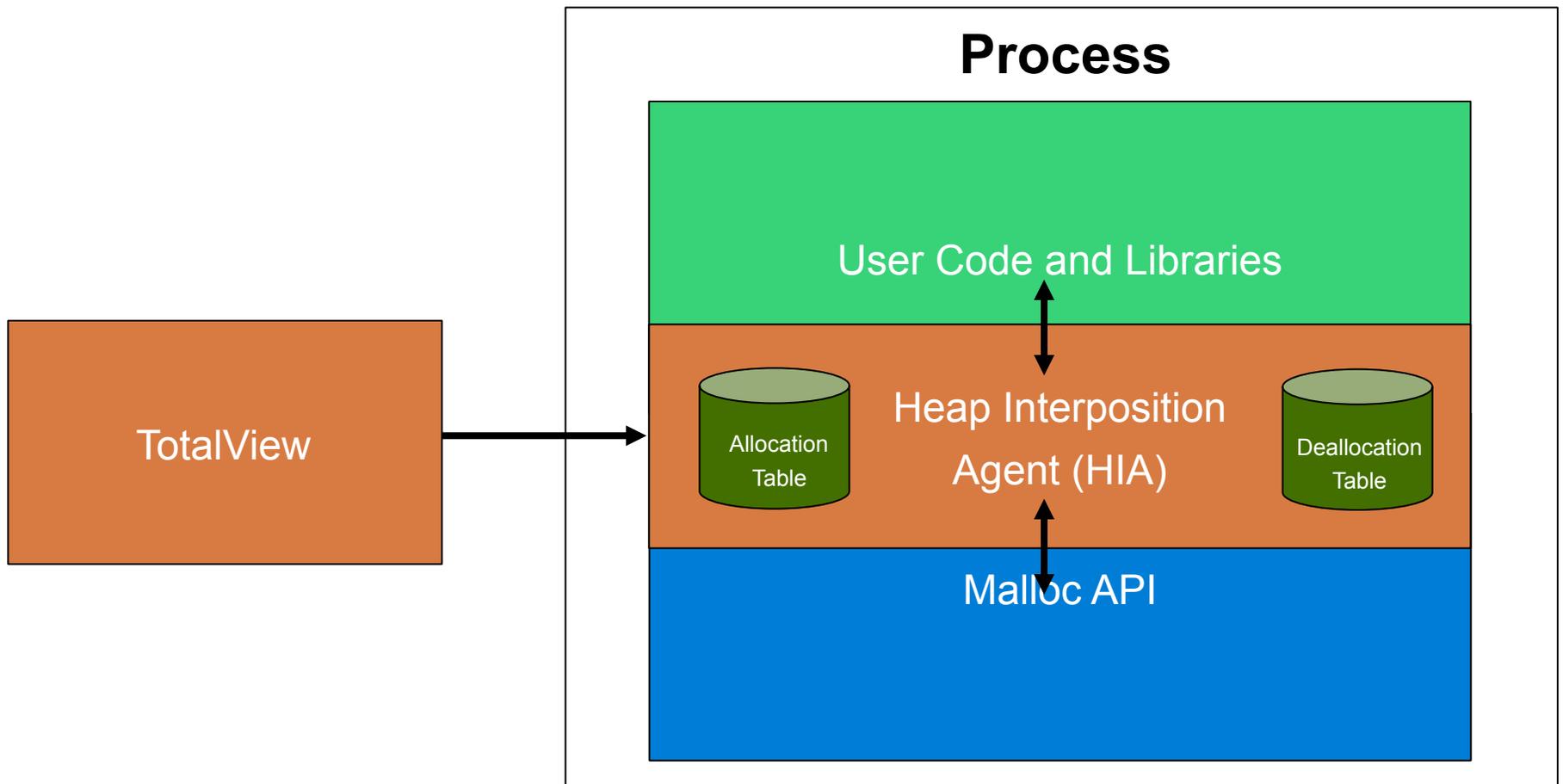
Memory Debugging

What Is MemoryScape®?

- Runtime Memory Analysis : Eliminate Memory Errors
 - Detects memory leaks *before* they are a problem
 - Explore heap memory usage with powerful analytical tools
 - Use for validation as part of a quality software development process
- Major Features
 - Included in TotalView, or Standalone
 - Detects
 - Malloc API misuse
 - Memory leaks
 - Buffer overflows
 - Supports
 - C, C++, Fortran
 - Linux, Unix, and Mac OS X
 - Intel® Xeon Phi™
 - MPI, pthreads, OMP, and remote apps
 - Low runtime overhead
 - Easy to use
 - Works with vendor libraries
 - No recompilation or instrumentation

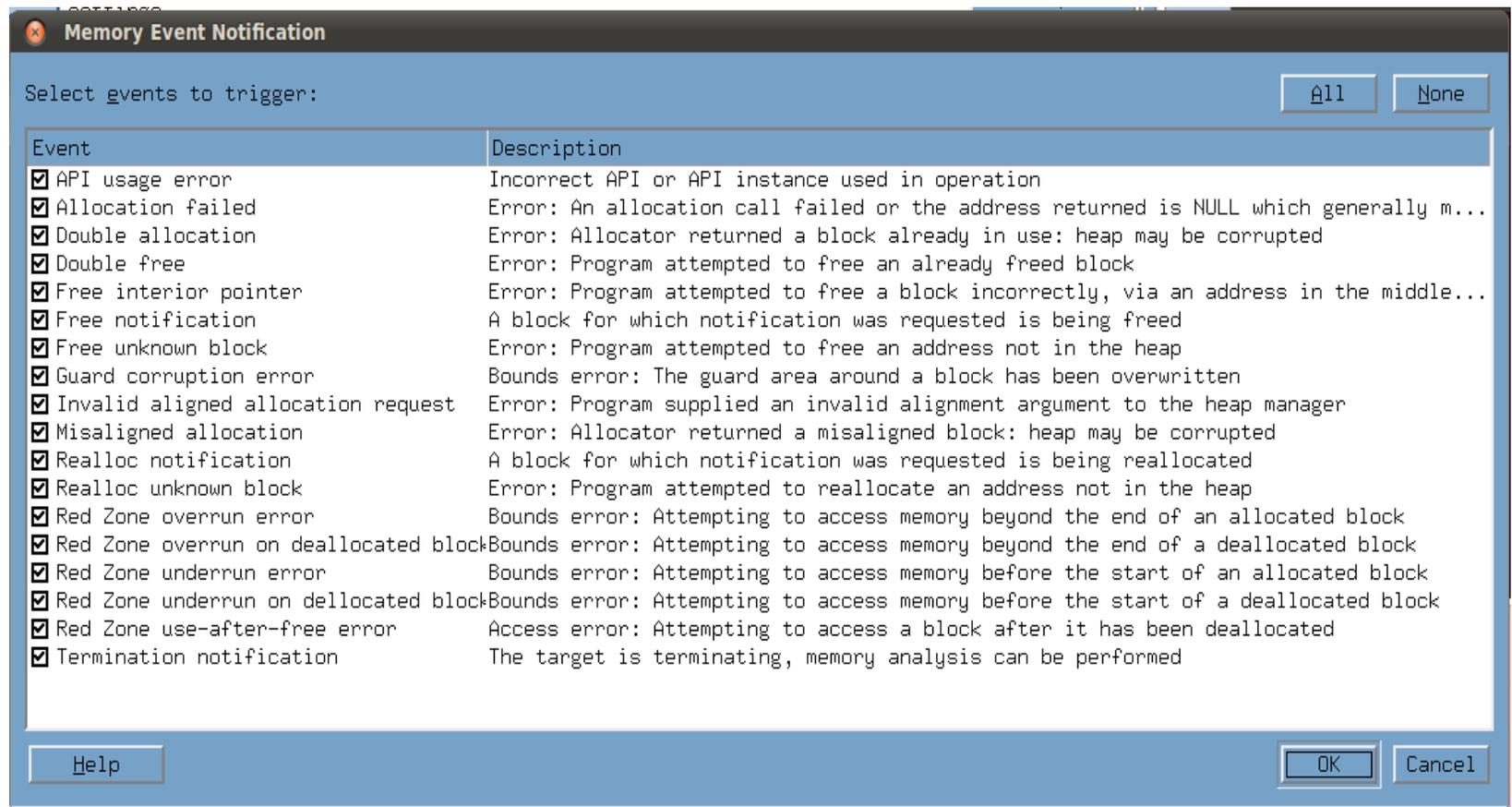


The Agent and Interposition



Enabling Memory Debugging

Memory Event Notification



Memory Event Details Window

Memory Event Details - Process 1: filterapp-mpi.1 - 1

Process 1: filterapp-mpi.1 - 1 Time: 00:40:18

Event: Double free - Error: Program attempted to free an already freed block

Event Location Allocation Location Deallocation Location Block Details

Backtrace

ID	Function	Line #	Source Information
100	free	184	malloc_wrappers_dlopen.c
	double_free	74	main.cxx
	main	287	main.cxx
	__libc_start_mair		libc.so.6
	_start		filterapp-mpi

Source /home/demouser/tv-src/main.cxx

```
73 // Show that the deallocation stack is available now
74 junk = 0;
75
76 // Now release the memory the second time - illegal
77 #ifdef USEMPI
78 if( rank == 1 )
79 #endif
80     free ( p );
81
```

Generate Memory File

Close View in Block Properties window Help

Memory Block Properties

Memory Blocks

+ 0x0949ea88 - 0x0949eb32 D !

Point of Allocation Point of Deallocation Memory Cont

Backtrace

ID	Function	Line #	Source Information
99	malloc	166	malloc_wrappers_dlopen.c
	double_free	60	main.cxx
	main	287	main.cxx
	__libc_start_mair		libc.so.6
	_start		filterapp-mpi

Source /home/demouser/tv-src/main.cxx

```
59 int junk = 0;
60 p = (int*) malloc( length );
61 printf ( "malloced %d (%#6x) bytes at %p\n", leng
62
63 // Breakpoint here
64 // Show allocated annotation
```

Close Hide Backtrace/Content Help

Heap Graphical View

MemoryScape 3.2.3-0

File Tools Window Help

Home **Memory Reports** Manage Processes Memory Debugging Options Tips 1 New Event

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

May 11, 2012

Heap Status Graphical Report

Options: Detect Leaks Relative to Baseline Enable Filtering

Leaked Block

Process 1: filterapp-mpi.1

0x0949d058 - 0x094d2c00 (214.91KB)

Heap Information | Backtrace/Source | Memory Content

Overall Totals	
Category	Bytes
Heap	
Allocated	81.55KB
Deallocated	129.88KB
Hoarded	0
Leaked	Unknown
Red Zones	0

Selected Block	
Property	Value
Start Address	0x0949d098
End Address	0x0949d0bb
Size	36
Type	Allocated
Filtered	No
Backtrace ID	3
Allocator	C
Owner	C

Related Blocks	
Category	
Backtrace ID 3	
Allocated	
Corrupted Guard Block	
Deallocated	
Guard Blocks	
Hoarded	
Leaked	
Red Zones	

Process Selection

- Process
 - Parallel Job file
 - MPI_COMM_WORLD
 - filterapp-mpi
 - filterapp-mpi
 - filterapp-mpi
 - filterapp-mpi

Leak Detection

MemoryScape 3.2.3-0

File Tools Window Help

Home Memory Reports Manage Processes Memory Debugging Options Tips 1 New Event

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

May 11, 2012

Save Data
Export Memory Data

Heap Status Reports
Source Report
Backtrace Report

Other Reports Categories
Leak Detection Report
Memory Usage Report
Corrupted Memory Report
Compare Memory Usage

Other Tasks
Manage Filters

Process Selection
Process
Parallel Job filter
MPI_COMM_WORLD
filterapp-mpi
filterapp-mpi
filterapp-mpi
filterapp-mpi

Heap Status Graphical Report

Options
 Detect Leaks Relative to Baseline Enable Filtering Leaked Block

Process 1: filterapp-mpi.1

0x0949d058 - 0x094d2c00 (214.91KB)

Heap Information | Backtrace/Source | Memory Content

Category	Bytes
Heap	
Allocated	14.53KB
Deallocated	129.88KB
Hoarded	0
Leaked	67.02KB
Red Zones	0

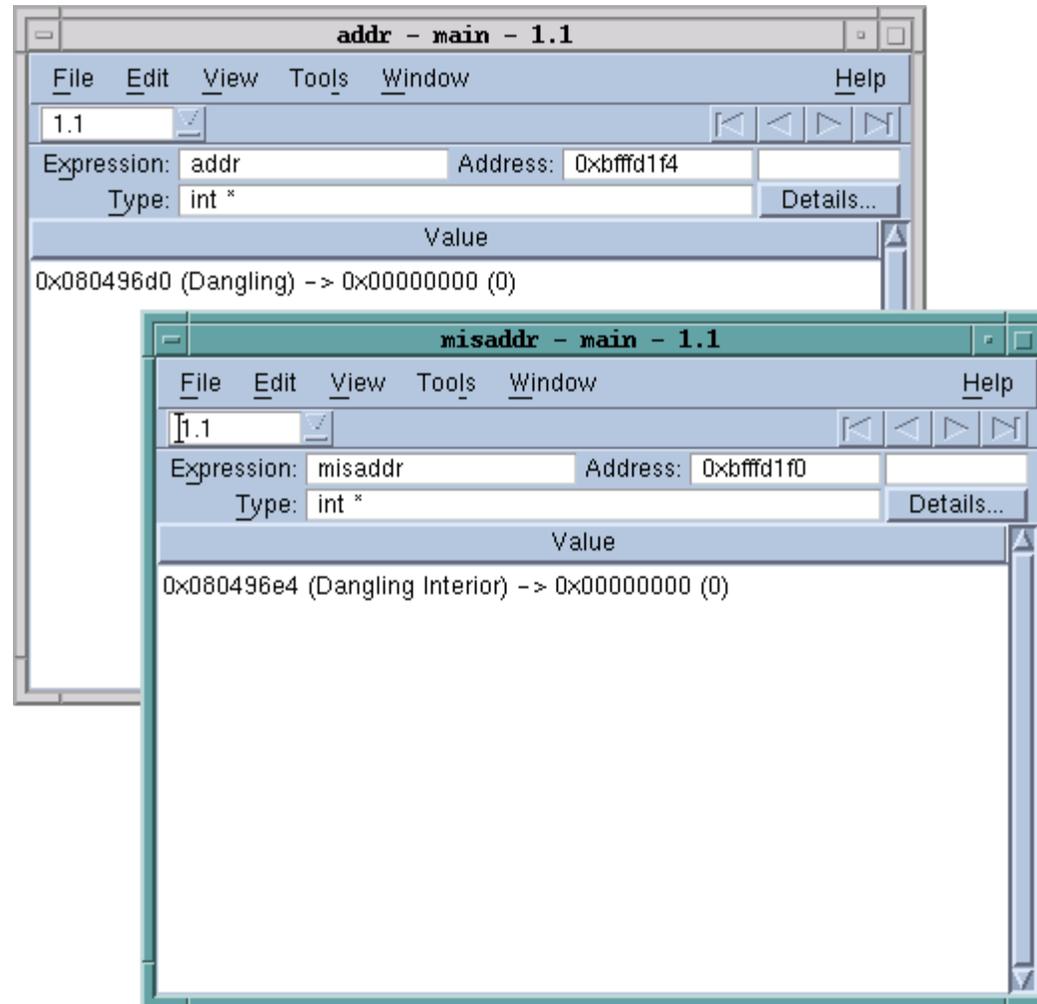
Property	Value
Start Address	0x0949e990
End Address	0x0949e99b
Size	12
Type	Leaked
Filtered	No
Backtrace ID	125
Allocator	C
Owner	C

Category
Backtrace ID 125
Allocated
Corrupted Guard Block
Leaked
Deallocated
Guard Blocks
Hoarded
Red Zones

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**

Dangling Pointer Detection



Memory Corruption Report

The screenshot displays a software interface for memory corruption analysis. The main window is titled "Process Set" and shows a list of processes, including "filterapp (25212)". The interface is divided into several sections:

- Configuration:** Includes tabs for "Leak Detection", "Heap Status", "Memory Usage", and "Memory Compare".
- Corrupted Block:** A table showing the details of a corrupted memory block. The table has three columns: "Preceding Block", "Corrupted Block", and "Following Block".
- Backtrace/Source:** A section with two sub-panels: "Backtrace" and "Source".

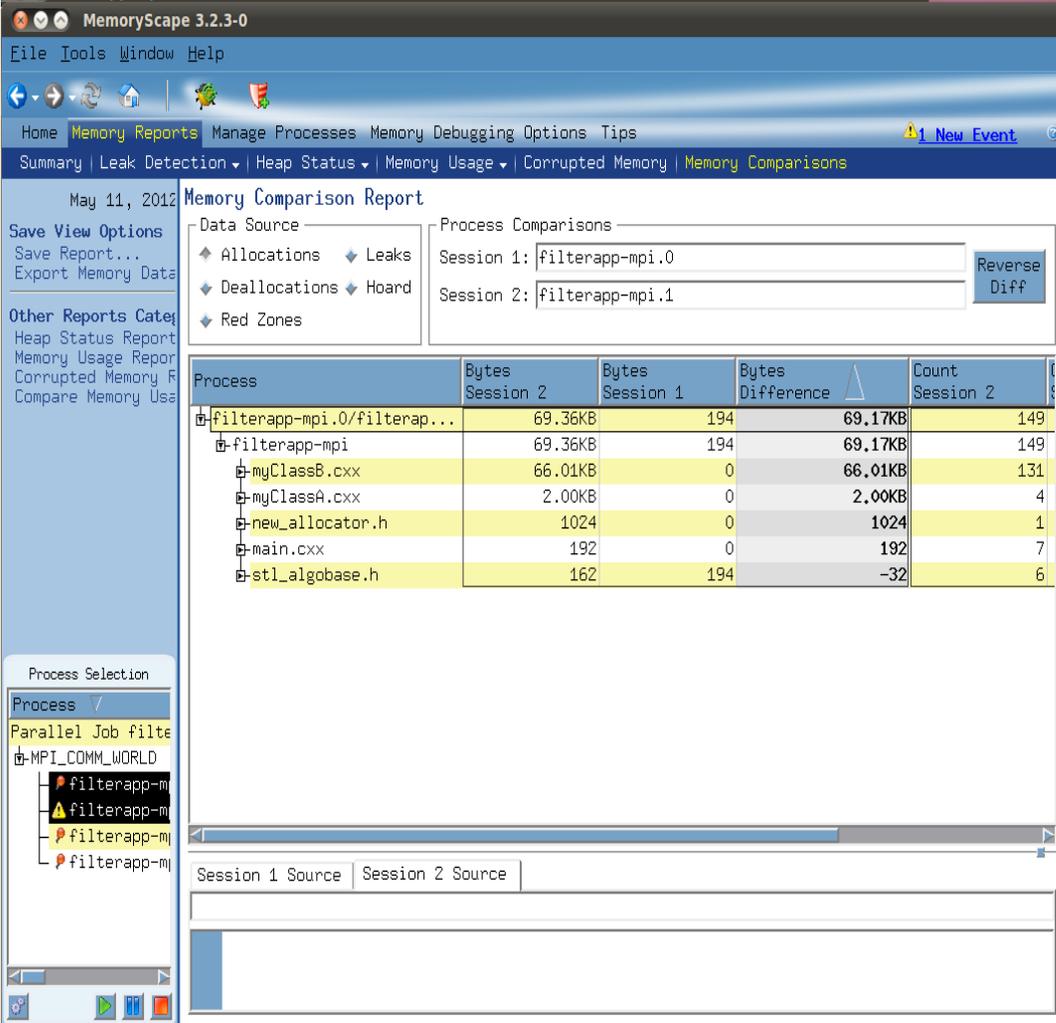
Preceding Block	Corrupted Block	Following Block
0x022b0020 - 64 bytes - 0x022b005f	0x022b0090 - 64 bytes - 0x022b00cf	0x022b0100 - 64 bytes - 0x022b013f

Process	Function	Line #	Source Information
3			
	malloc	166	malloc_wrappers_dlopen.c
	corrupt_data	76	main.cxx
	main	126	main.cxx
	_libc_start_main		libc.so.6
	_start		filterapp


```
irisg/temp/webcast_demo_files/memory/main.cxx
72 // Use 8 byte pre and post guard size.
73 size = 16;
74
75 // Allocate some arrays.
76 p0 = (int *) malloc( size * sizeof( int ) );
77 p1 = (int *) malloc( size * sizeof( int ) );
```

Memory Comparisons

- “Diff” live processes
 - Compare processes across cluster
- Compare with baseline
 - See changes between point A and point B
- Compare with saved session
 - Provides memory usage change from last run



MemoryScape 3.2.3-0

File Tools Window Help

Home Memory Reports Manage Processes Memory Debugging Options Tips

Summary Leak Detection Heap Status Memory Usage Corrupted Memory Memory Comparisons

May 11, 2012

Save View Options
Save Report...
Export Memory Data

Other Reports Categories
Heap Status Report
Memory Usage Report
Corrupted Memory Report
Compare Memory Usage

Memory Comparison Report

Data Source

- ◆ Allocations ◆ Leaks
- ◆ Deallocations ◆ Hoard
- ◆ Red Zones

Process Comparisons

Session 1: filterapp-mpi.0

Session 2: filterapp-mpi.1

Reverse Diff

Process	Bytes Session 2	Bytes Session 1	Bytes Difference	Count Session 2
filterapp-mpi.0/filterapp...	69.36KB	194	69.17KB	149
filterapp-mpi	69.36KB	194	69.17KB	149
myClassB.cxx	66.01KB	0	66.01KB	131
myClassA.cxx	2.00KB	0	2.00KB	4
new_allocator.h	1024	0	1024	1
main.cxx	192	0	192	7
stl_algobase.h	162	194	-32	6

Process Selection

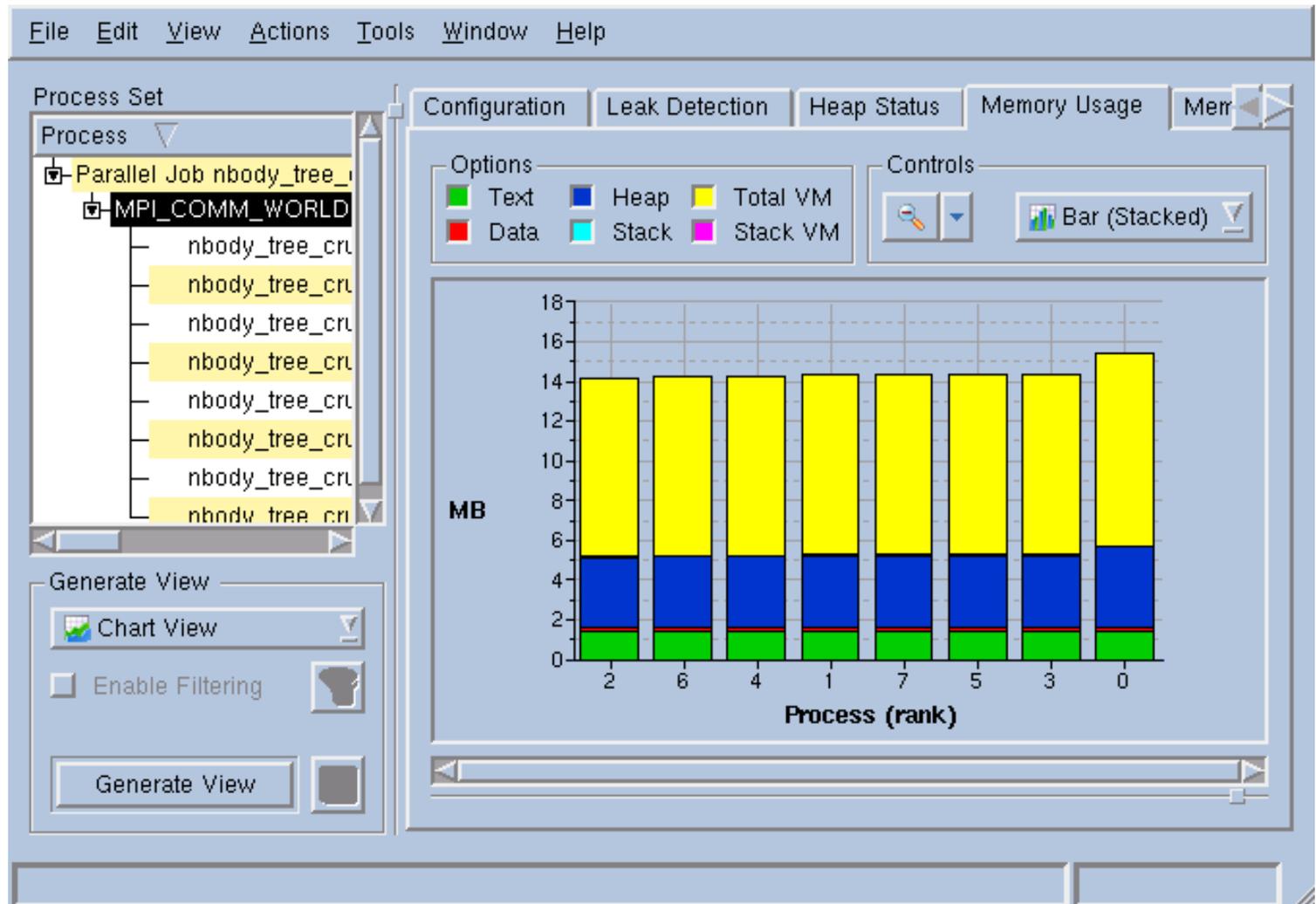
Process

- Parallel Job filterapp-mpi
- MPI_COMM_WORLD
 - filterapp-mpi
 - filterapp-mpi
 - filterapp-mpi
 - filterapp-mpi

Session 1 Source

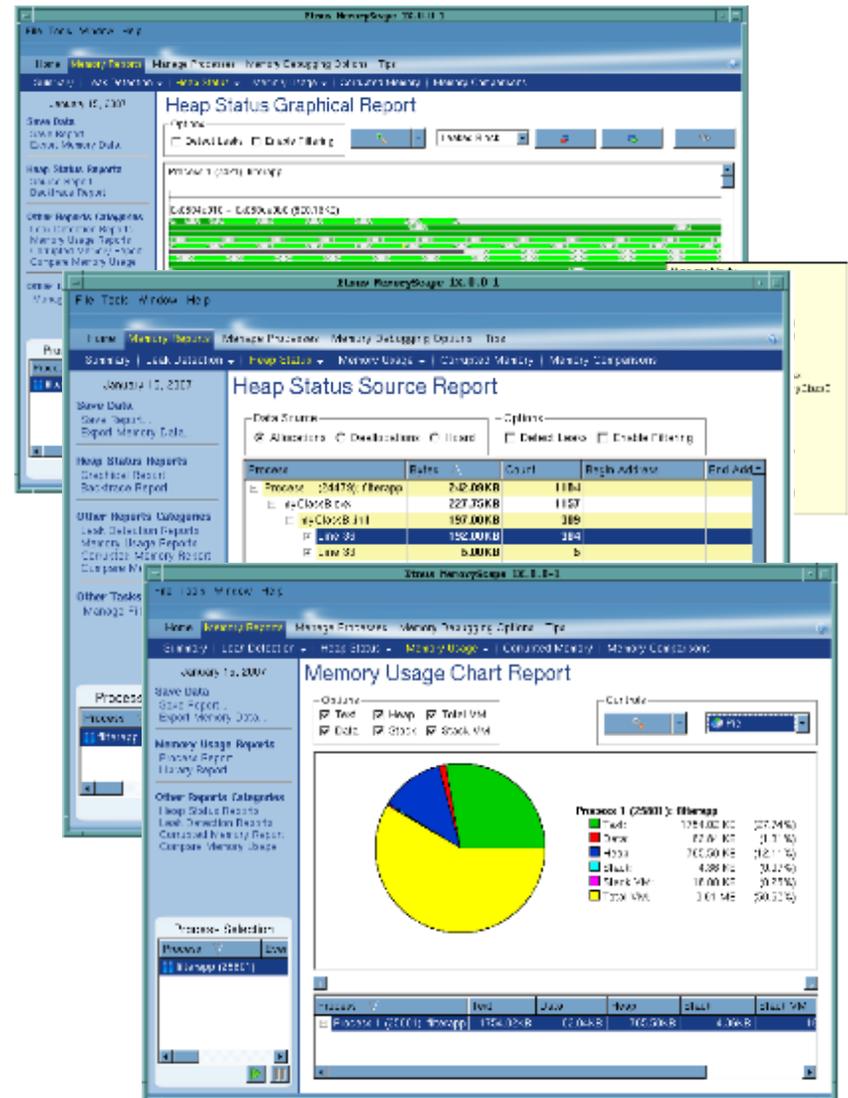
Session 2 Source

Memory Usage Statistics



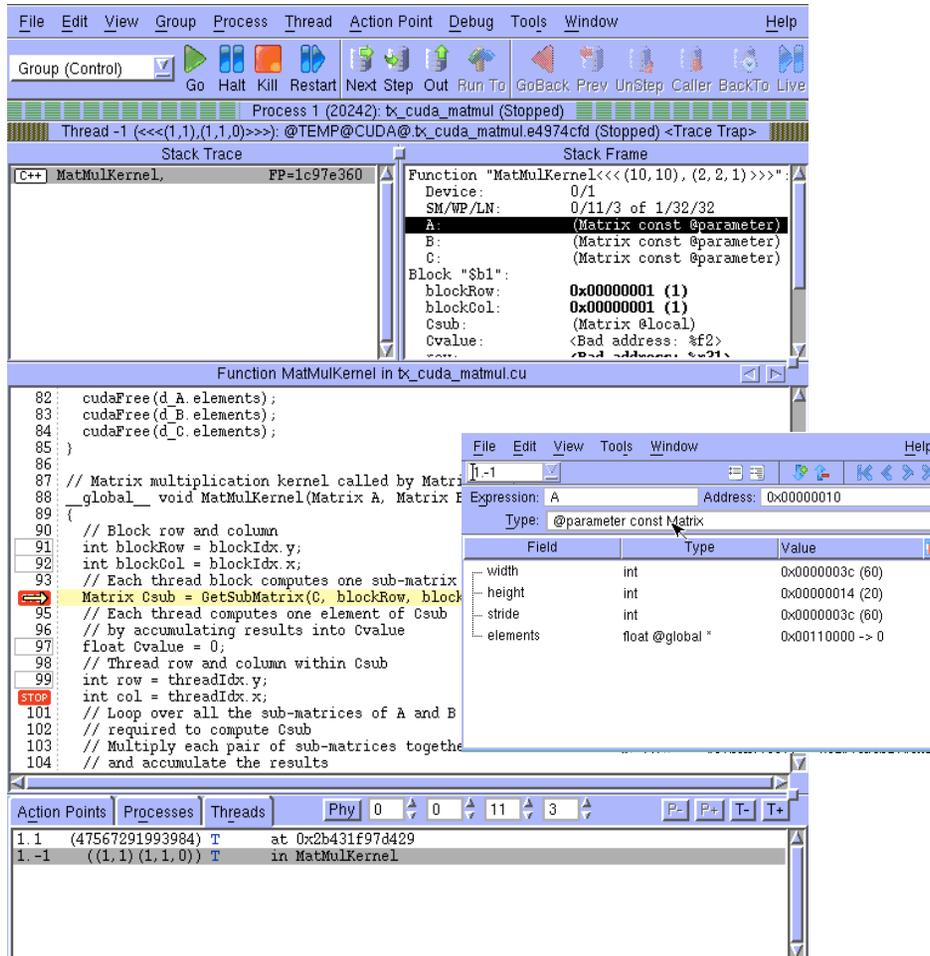
Memory Reports

- 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



Debugging Accelerators and Coprocessors

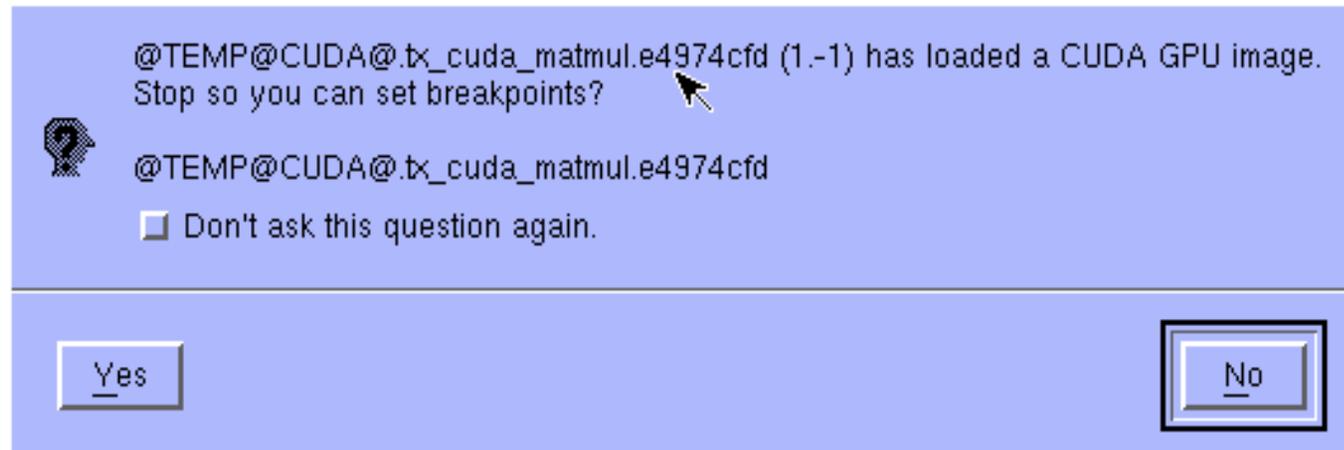
TotalView for the NVIDIA® GPU Accelerator



- NVIDIA Kepler
- NVIDIA CUDA 5.0, 5.5, and 6.0 (New in 8.14)
 - With support for Unified Memory
- Cray CCE OpenACC
- 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 types and separate memory address spaces
 - Leverages CUDA memcheck

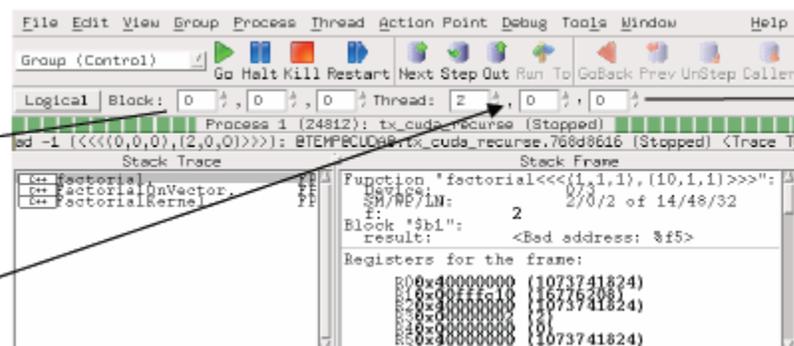
Debugging CUDA in TotalView

- When a new kernel is loaded, you get the option of setting breakpoints
- Once breakpoints are set, you can turn off the dialog and say no



Debugging CUDA in TotalView

- CUDA threads are considered part of the initiating process
- CUDA threads are given a negative TotalView thread id to distinguish them
- Normal TotalView controls work on CUDA code
- Underneath Toolbar is a GPU focus thread selector for changing block and thread indices



Block (x,y,z)

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

Control of Threads and Warps

- Warps advance synchronously
 - They share a PC
- Single step operation advances all GPU threads in the same warp
- Stepping over a `__syncthreads()` call will advance all relevant threads
- To advance more than one warp
 - Continue, possibly after setting a new breakpoint
 - Select a line and “Run To”

CUDA Built-in Runtime Variables

- Supported built-in runtime variables are:
 - `struct dim3_16 threadIdx;`
 - `struct dim2_16 blockIdx;`
 - `struct dim3_16 blockDim;`
 - `struct dim2_16 gridDim;`
 - `int warpSize;`

GPU Device Status

- Display of PCs across SMs, Warps and Lanes
- Updates as you step
- Shows what hardware is in use
- Helps you map between logical and hardware coordinates

The screenshot shows a tree view of GPU hardware coordinates and thread states. The tree is expanded to show SM 2/1, which contains 48 warps. Warp 00/48 is selected, showing 32 lanes. Each lane is associated with a thread ID and a Program Counter (PC) value. The PC values for lanes 01/32 through 09/32 are different, indicating divergent threads. The state of lanes is shown as 'Valid/Active/Divergent'.

Name	Description
Device 0/3	
Device Type	gf100
Lanes	32
SM 2/1	
Valid Warps	0000000000000001
Warp 00/48	Block (0,0,0)
Lane 00/32	Thread (0,0,0)
Lane 01/32	Thread (1,0,0)
Lane 02/32	Thread (2,0,0)
Lane 03/32	Thread (3,0,0)
Lane 04/32	Thread (4,0,0)
Lane 05/32	Thread (5,0,0)
Lane 06/32	Thread (6,0,0)
Lane 07/32	Thread (7,0,0)
Lane 08/32	Thread (8,0,0)
Lane 09/32	Thread (9,0,0)
Valid/Active/Divergent	000003ff, 000003fc, 00000003
SM Type	sm_20
SMS	14
Warps	48
Device 1/3	
Device Type	gt200
Lanes	32
SM Type	sm_13

Example of divergent GPU threads

Different PC for two groups of lanes

State of lanes inside warp

TotalView for the Intel® Xeon Phi™ coprocessor

Supports All Major Intel Xeon Phi Coprocessor Configurations

- Native Mode
 - With or without MPI
- Offload Directives
 - Incremental adoption, similar to GPU
- Symmetric Mode
 - Host and Coprocessor
- Multi-device, Multi-node
- Clusters

User Interface

- MPI Debugging Features
 - Process Control, View Across, Shared Breakpoints
- Heterogeneous Debugging
 - Debug Both Xeon and Intel Xeon Phi Processes

Memory Debugging

- Both native and symmetric mode

ID / Rank	Host	Status	Description
1	<local>	R	/opt/intel/composerxe/Sample
1.1	<local>	R	in main
1.2	<local>	R	in __poll
1.3	<local>	R	in __poll
1.4	<local>	R	in pthread_cond_wait
2	192.168.1.10M	R	/tmp/col_procs/1/5856/offload
2.1	192.168.1.10R	R	in sem_wait
2.2	192.168.1.10B6	R	in compute07
2.3	192.168.1.10R	R	in __poll
2.4	192.168.1.10R	R	in pthread_cond_wait

Stack Trace

Function	FP
compute07	FP=7F50F44D24F0
L_sample07_76_pan_region1_2_39	FP=7F50F44D24C0
_offload_entry_sample07_c_76sample07	FP=7F7424170FFloadDescriptor7offloadE_PV0_150_t
_COISinkPipe::RunFunction	FP=7F50F44D24C0
_COISinkPipe::ProcessMessages	FP=7F50F44D24E0
_COISinkPipe::ThreadProc	FP=7F50F44D2420
start_thread	FP=7F50F44D2F30
_clone	FP=7F50F44D2F38

Stack Frame

Function "compute07":

out: 0x7F50F44D2754 -> 0x41400000 (1092)

size: 0x00000010 (16)

Local variables:

i: 0x00000010 (16)

Registers for the frame:

Xrax: 0x7F50F44D2754 (139985823803220)

Xrdi: 0x00000010 (16)

Xrcx: 0x7F50F44D2754 (139985823803220)

Xrbx: 0x7F50F44D2754 (139985823803220)

```
30 for (i=0; i<: i++)
31 {
32     array[i] = p[i];
33 }
34
35 #ifdef __MIC__
36     retval = 1;
37 #else
38     retval = 0;
39 #endif
40
41 // Return 1 if array initialization was done on target
42 return retval;
43 }
44
45 __attribute__((target(mic))) void compute07(int* out, int size)
46 {
47     int i;
48
49     for (i=0; i<size; i++)
50     {
51         out[i] = array[i]*2;
52     }
53 }
54 //.....07
```

Action Points

Process	Thread
2.1 (139985836167360) R	in sem_wait
2.2 (13998583607232) B6	in compute07
2.3 (139985834444544) R	in __poll
2.4 (139985842637248) R	in pthread_cond_wait

Batch Debugging

TVScript Overview

- Gives you non-interactive access to TotalView's capabilities
- Useful for
 - Debugging in batch environments
 - Watching for intermittent faults
 - Parametric studies
 - Automated testing and validation
- TVScript is a script (not a scripting language)
 - It runs your program to completion and performs debugger actions on it as you request
 - Results are written to an output file
 - No GUI
 - No interactive command line prompt
- A “better” printf()

Sample Output

- Simple interface to create an action point
 - create_actionpoint "#85=>print foreign_addr"
- Sample output with all information

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! Print
!
! Process:
! ./TVscript_demo (Debugger Process ID: 5, System ID: 2457@127.0.1.1)
! Thread:
! Debugger ID: 5.1, System ID: 3077191888
! Rank:
! 0
! Time Stamp:
! 05-14-2012 17:11:24
! Triggered from event:
! actionpoint
! Results:
! err_detail = {
!   intervals = 0x0000000a (10)
!   almost_pi = 3.1424259850011
!   delta = 0.000833243988525023
! }
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

Events

- General
 - any_event
- Source code debugging events
 - actionpoint
 - error
- Memory events (just a few, all are listed in Chapter 4 of TotalView Reference Guide)
 - any_memory_event
 - free_not_allocated
 - guard_corruption
 - rz_overrun, rz_underrun, rz_use_after_free

Actions

- Source code
 - `display_backtrace [-level num] [numlevels] [options]`
 - `print [-slice {exp}] {variable | exp}`
- Memory
 - `check_guard_blocks`
 - `list_allocations`
 - `list_leaks`
 - `save_html_heap_status_source_view`
 - `save_memory_debugging_file`
 - `save_text_heap_status_source_view`

Command syntax

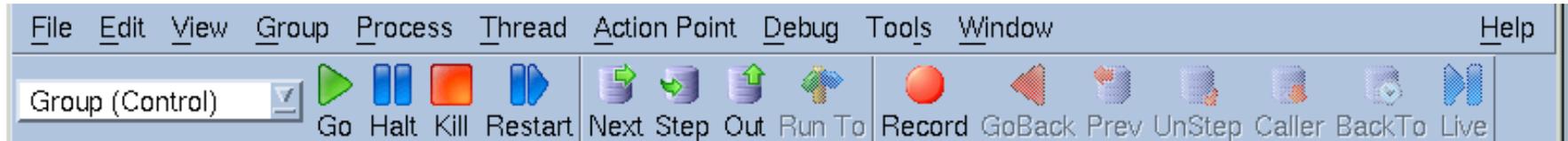
- General syntax
 - `tvscript [options] [filename] -a [program_args]`
- MPI Options
 - `-mpi starter` starter comes from Parallel tab dropdown
 - `-starter_args` “args for starter program”
 - `-nodes`
 - `-np` or `-procs` or `-tasks`

Command syntax

- Action options
 - -create_actionpoint “src_expr[=>action1[,action2] ...]”
 - Repeat on command line for each actionpoint
 - -event_action “event_action_list”
 - event1=action1,event2=action2 or event1=>action1,action2
 - Can repeat on command line for multiple actions
- General options
 - -display_specifiers “display_specifiers_list”
 - -maxruntime “hh:mm:ss”
 - -script_file scriptFile
 - -script_log_filename logFilename
 - -script_summary_log_filename summaryLogFilename

Reverse Debugging

Deterministic Replay Debugging



- Reverse Debugging: Radically simplify your debugging
 - Captures and Deterministically Replays Execution
 - Not just “checkpoint and restart”
 - Eliminate the Restart Cycle and Hard-to-Reproduce Bugs
 - Step Back and Forward by Function, Line, or Instruction
- Specifications
 - A feature included in TotalView on Linux x86 and x86-64
 - No recompilation or instrumentation
 - Explore data and state in the past just like in a live process, including C++View transformations
 - Replay on Demand: enable it when you want it
 - Supports MPI on Ethernet, Infiniband, Cray XE Gemini
 - Supports Pthreads, and OpenMP
 - **New: Save / Load Replay Information (CLI only)**

```
40
41
42 int funcB(int
43 int c;
44 int i;
45 int v[MAXDEPT
46 int *p;
47 → c=b+2;
48 p=&c;
49 ▶ if(c<MAXDEPTH
50     c=funcA(c);
51 for (i=array1
52     v[i]=*p;
```

Running on ALCF systems

Debugging on BG/Q with Totalview 8.14.0

Load `.totalview` in your `.soft`

Use the remote display client

Just add `totalview --args` before `runjob`

- `totalview -args runjob --block $COBALT_PARTNAME -p 16 : demoMpi`
- Add options from `~/chrisg/ATPESC/example.tvdrc` to your `.totalview/.tvdrc` to use the MRNet early access
- For memory debugging (from documentation):
 - Link statically as `-L<path> -ltvheap -Wl,rpath,<path>`
 - Link dynamically as `-L<path> -Wl,@<path>/tvheap_bgqs_ld`
- TotalView 8.14 will be available on Mira, Vesta, Cetus and Tukey for the duration of the training.

Thanks!

- To learn more / sign up for the Scalability Early Experience Program please contact me: chris.gottbrath@roguewave.com
- Visit the website
 - <http://www.roguewave.com/products/totalview.aspx>
 - Videos
 - Documentation
 - Sign up for an evaluation
 - Contact customer support & post on the user forum