Debugging Heterogeneous HPC Applications with Totalview
Cray Users Group 2013, Napa, CA

Chris Gottbrath, Product Manager
May 6th, 2013

TotalView

John Hollis

Agenda

- **Hour 1 – Basic Topics (1-2 pm)**
  - Lecture 30 minutes
  - Lab 30 minutes
- **Hour 2 – Intermediate Topics (2-2:30, 3-3:30)**
  - Lecture 30 minutes
  - Lab 30 minutes
- **Hour 3 – Advanced Topics (3:30-4:30)**
  - Lecture and Demo

Basic Topics

- **Introduction**
- **Startup**
- **UI Navigation and Process Control**
- **Action Points**
- **Data Monitoring and Visualization**
- **Lab 30 minutes**

4

Intermediate Topics

- Intermediate Debugging for Parallel Applications
- Asynchronous Thread Control
- Lab 30 minutes

5

Advanced Topics

- Reverse Debugging with ReplayEngine
- Comparative Debugging
- CUDA/OpenACC Debugging
- Xeon Phi Debugging
- Support and Documentation

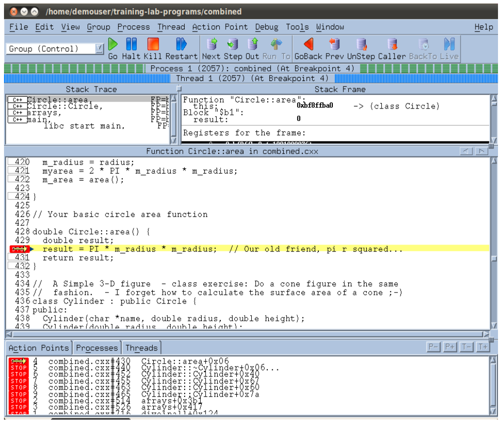
6

INTRODUCTION

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
- ReplayEngine reverse debugging
- Supports a Variety of Usage Models
  - Powerful and Easy GUI
    - Visualization
  - CLI for Scripting
  - Long Distance Remote Debugging
  - Unattended Batch Debugging



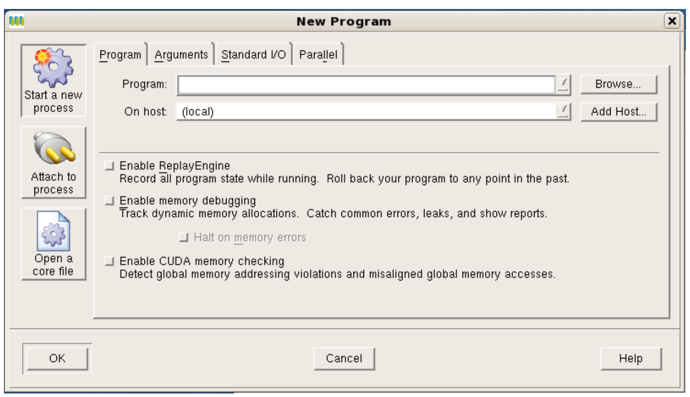
Supported Compilers and Architectures

- **Platform Support**
  - Linux x86, x86-64, ia64, Power
  - Mac Intel
  - Solaris Sparc and AMD64
  - AIX
  - Cray XT, XE, XK, XC, CS-3000AC
  - IBM BG/L, BG/P, BG/Q
- **Languages / Compilers**
  - C/C++, Fortran, UPC, Assembly
  - Many Commercial & Open Source Compilers
- **Parallel Environments**
  - MPI
    - MPICH1& 2, Open MPI, Intel MPI, SGI MPT & Propack, SLURM, poe, MPT, Quadrics, MVAPICH1 & 2, Bullx MPI, & many others
  - UPC

STARTUP

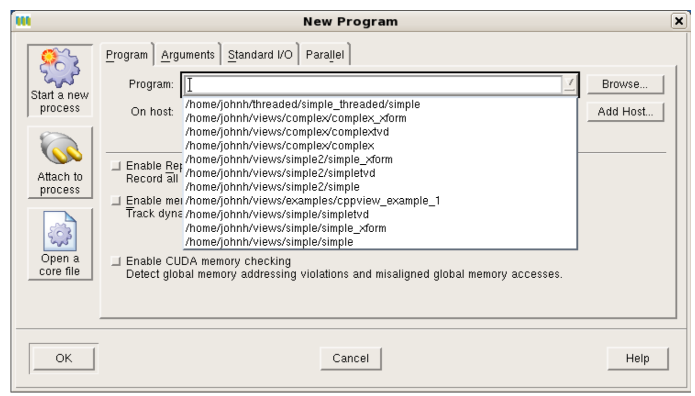
Starting TotalView

Start New Process



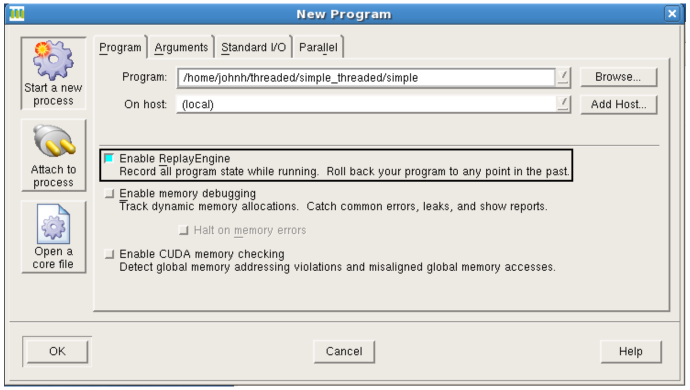
Starting TotalView

Start New Process – Select a recent process



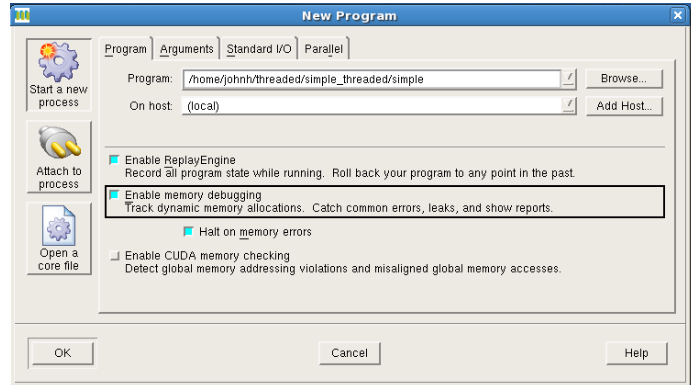
Starting TotalView

Start New Process – Enable ReplayEngine



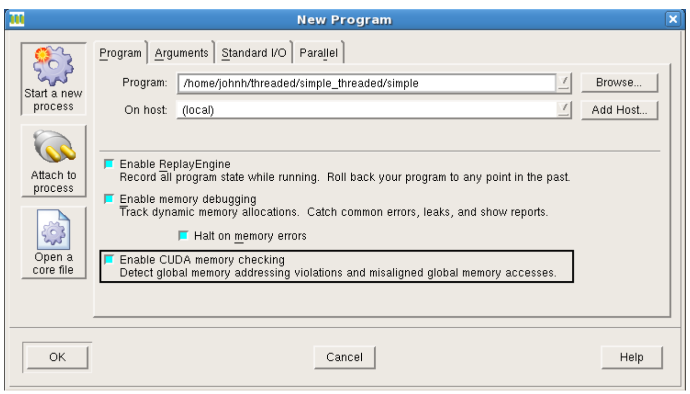
Starting TotalView

Start New Process – Memory Debugging



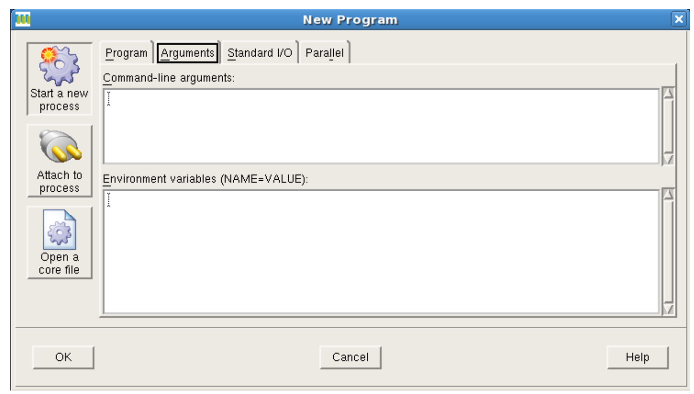
Starting TotalView

Start New Process – CUDA memory checking



Starting TotalView

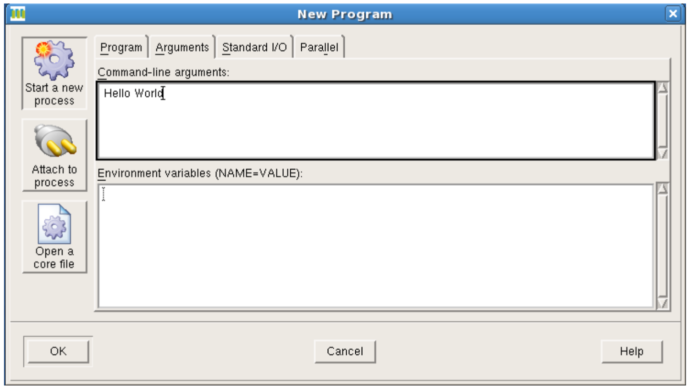
Start New Process – Arguments tab



Starting TotalView

ROGUE WAVE SOFTWARE

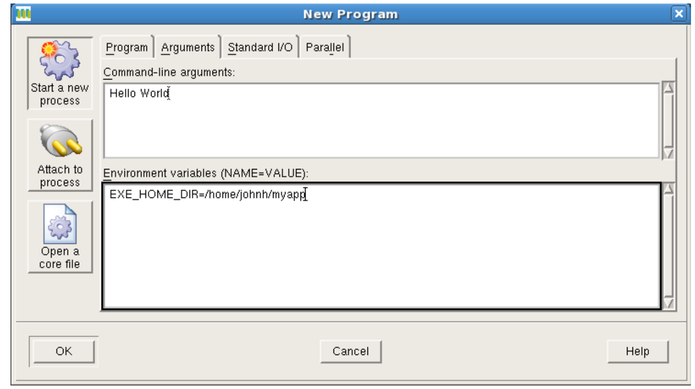
Start New Process – Command-line Args



Starting TotalView

ROGUE WAVE SOFTWARE

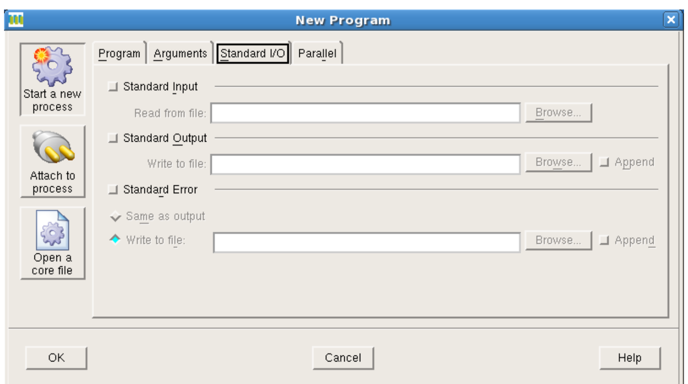
Start New Process – set environment variables



Starting TotalView

ROGUE WAVE SOFTWARE

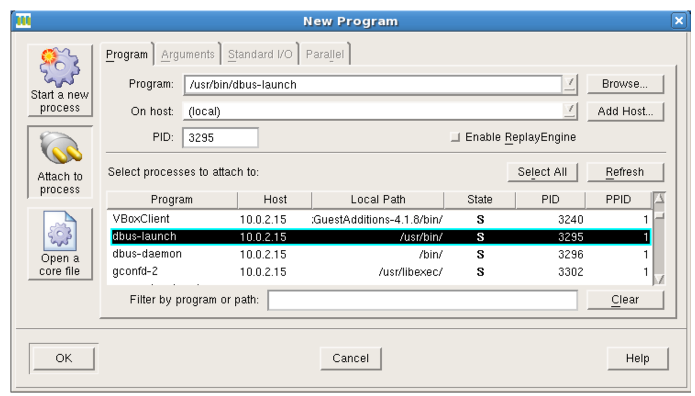
Start New Process – Standard I/O redirection



Starting TotalView

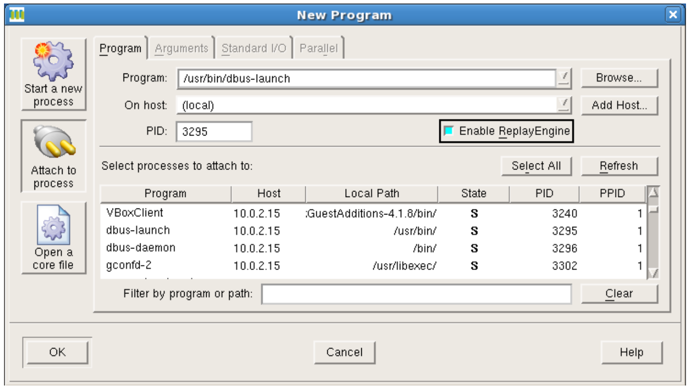
ROGUE WAVE SOFTWARE

Attach to Process



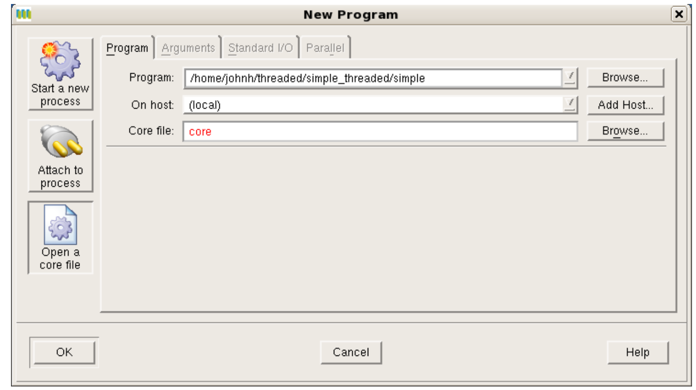
Starting TotalView

Attach to Process – Enable Replay Engine



Starting TotalView

Open a Core File



Via Command Line

Normal
totalview [tv_args] prog_name [–a prog_args]

Attach to running program
totalview [tv_args] prog_name –pid PID# [–a prog_args]

Attach to remote process
totalview [tv_args] prog_name –remote name [–a prog_args]

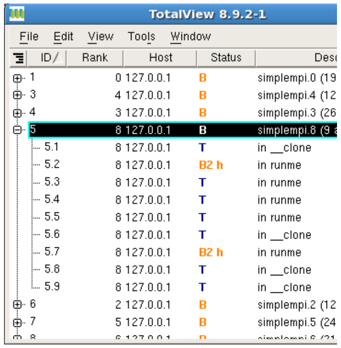
Attach to a core file
totalview [tv_args] prog_name corefile_name [–a prog_args]

UI NAVIGATION AND PROCESS CONTROL

Interface Concepts

Root Window

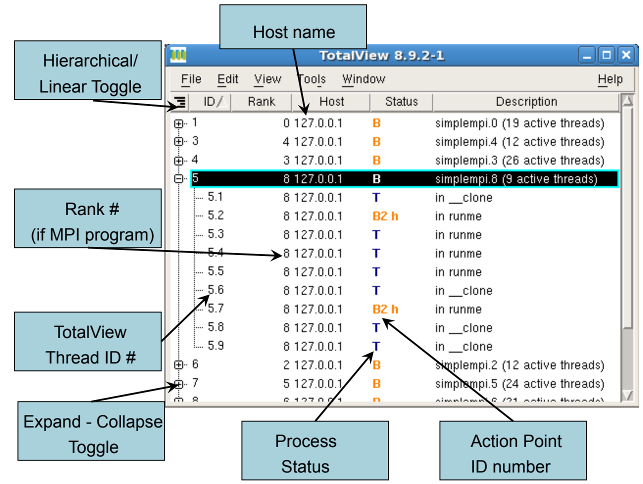
- State of all processes being debugged
- Process and Thread status
- Instant navigation access
- Sort and aggregate by status



➤Status Info

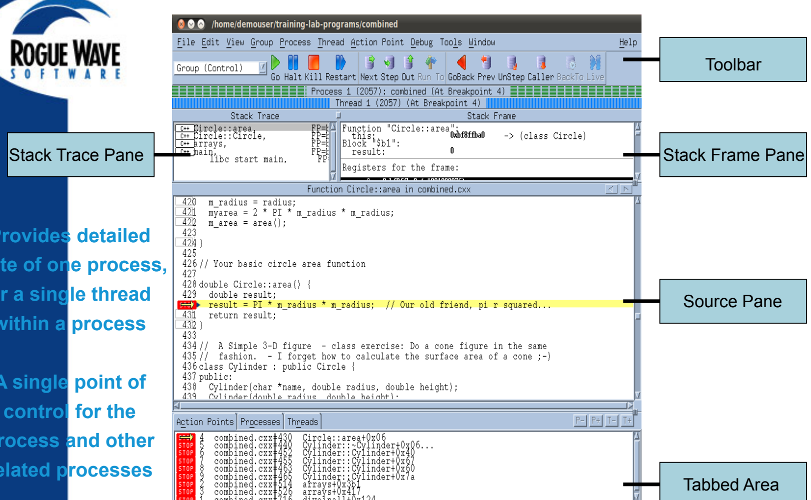
- •T = stopped
- •B = Breakpoint
- •E = Error
- •W = Watchpoint
- •R = Running
- •M = Mixed
- •H = Held

TotalView Root Window



- **Dive to refocus**
- ***Dive in new window* to get a second process window**

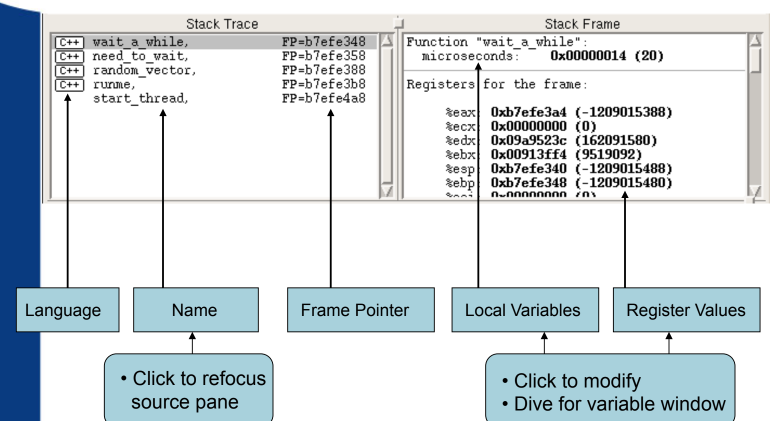
Process Window Overview



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

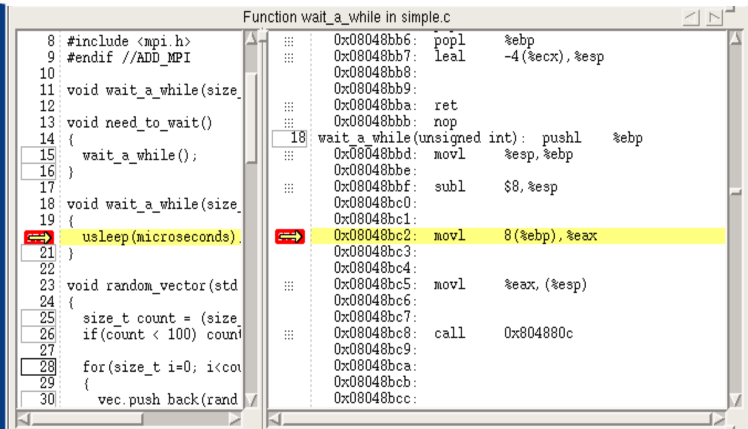
A single point of control for the process and other related processes

Stack Trace and Stack Frame Panes

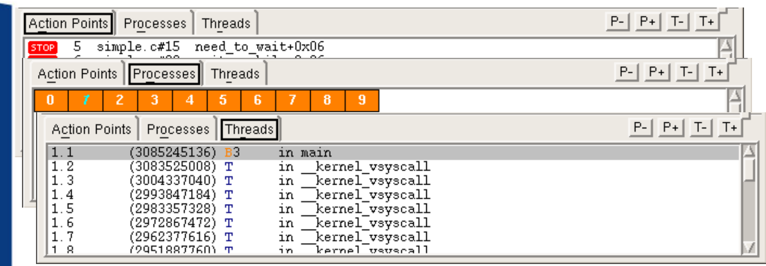


Source Code Pane

View as Source - or Assembly - or Both!



Tabbed Pane



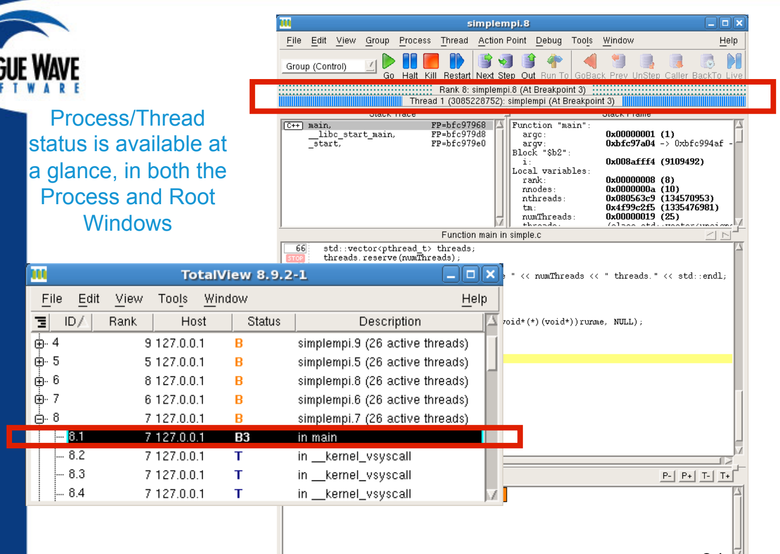
Action Points Tab
all currently defined action points

Processes Tab
all current processes

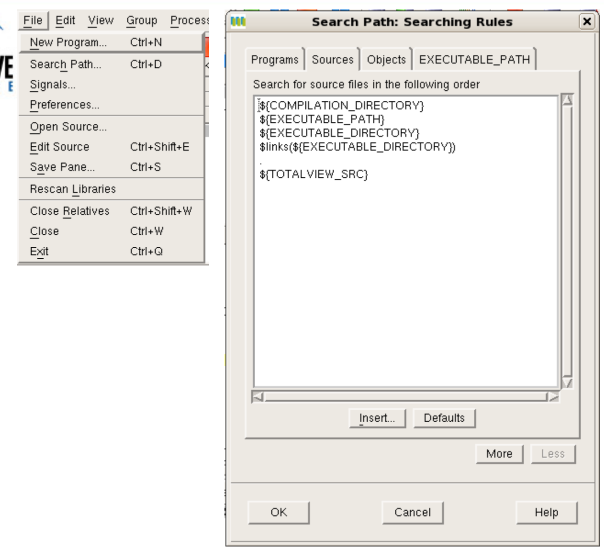
Threads Tab:
all current threads, ID's, Status

Process Status

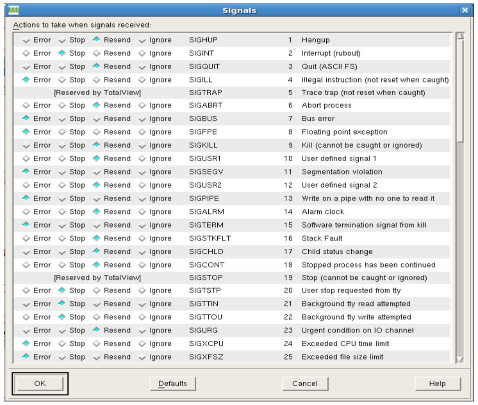
Process/Thread status is available at a glance, in both the Process and Root Windows



Search Paths

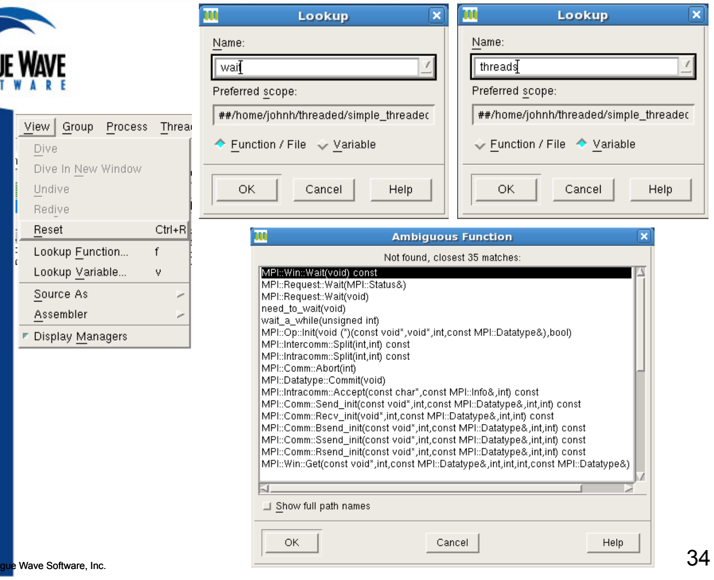


Managing Signals
File > Signals

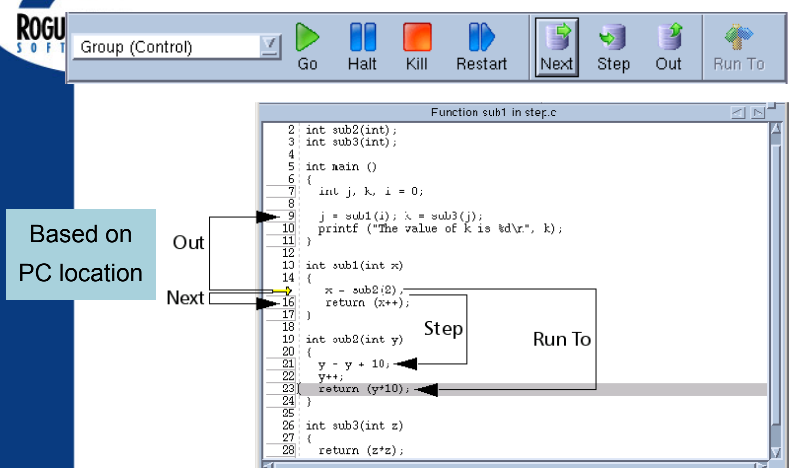


Error	Stop the process and flag as error
Stop	Stop the process
Resend	Pass the signal to the target and do nothing: use with signal handlers
Ignore	Discard the signal

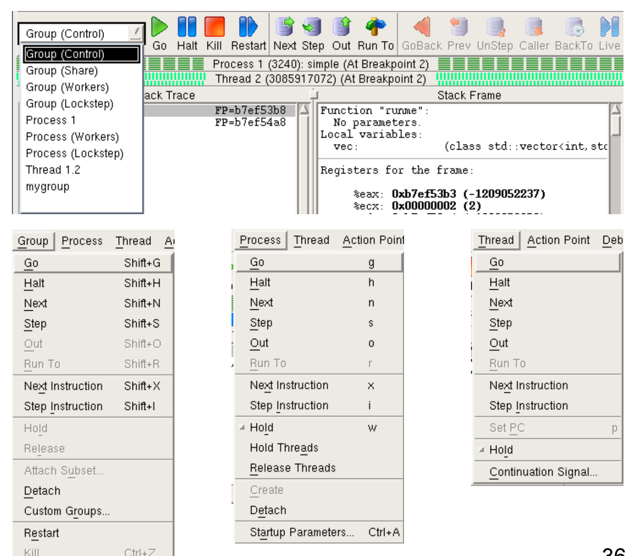
Finding Functions, Variables, and Source Files



Stepping Commands

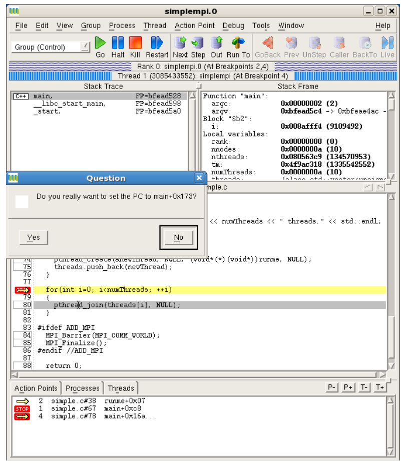


Stepping Commands



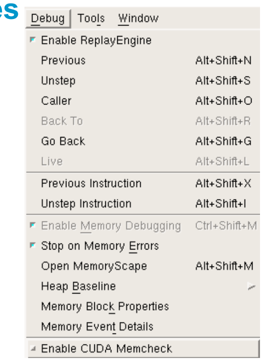
Using Set PC to resume execution at an arbitrary point

- Select the line
- Thread->Set PC
- Click Yes to set the PC



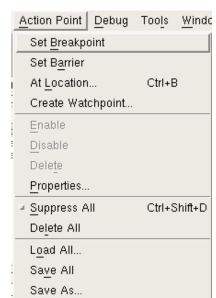
Debug Menu

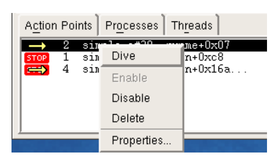
- Menu Items for extra features
  - Replay Engine
  - MemoryScape
  - CUDA Memcheck
- More on these features later



ACTION POINTS

Action Points





Breakpoints

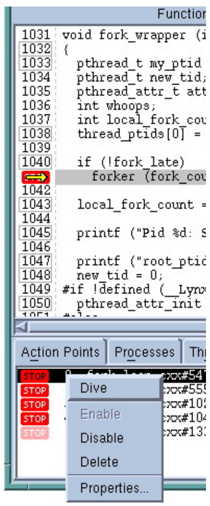
Barrier Points

Conditional Breakpoints

Evaluation Points

Watchpoints

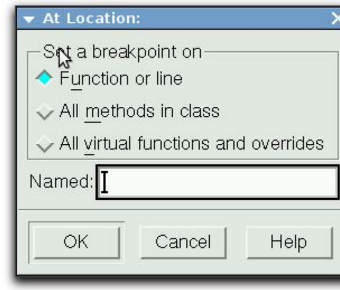
Setting Breakpoints



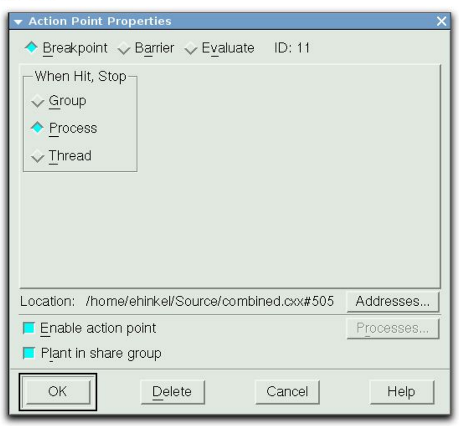
- Setting action points
  - Single-click line number
- Deleting action points
  - Single-click action point line
- Disabling action points
  - Single-click in Action Points Tab Pane
- Optional contextual menu access for all functions
- Action Points Tab
  - Lists all action points
  - Dive on an action point to focus it in source pane
- Action point properties
  - In Context menu
- Saving all action points
  - Action Point > Save All

Setting Breakpoints

- **Breakpoint->At Location…**
  - Specify function name or line #
  - Specify class name and break on all methods in class, optionally with virtuals and overrides

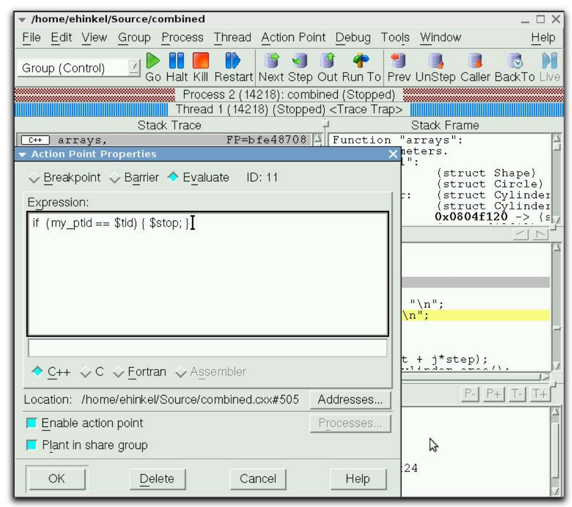


Setting Breakpoints



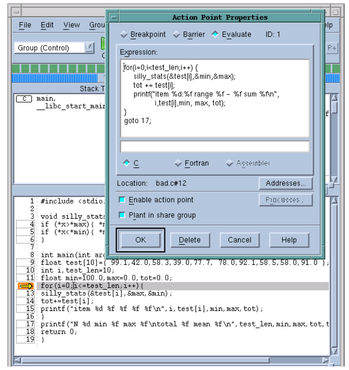
- **Breakpoint type**
- **What to stop**
- **Set conditions**
- **Enable/disable**
- **In 1 process or share group**

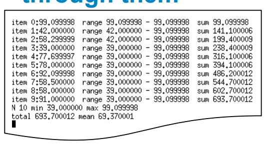
Conditional Breakpoint



Evaluation Breakpoint… Test Fixes on the Fly!

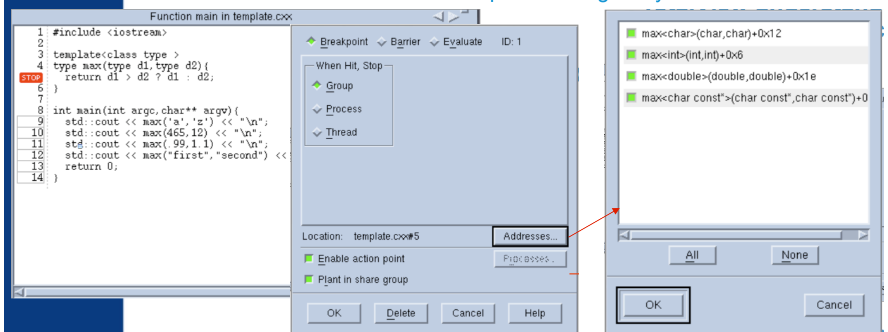
- Test small source code patches
- Call functions
- Set variables
- Test conditions
- C/C++ or Fortran
- Can' t use C++ constructors
- Use program variables
- ReplayEngine records changes but won' t step through them





Setting Breakpoints With C++ Templates

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

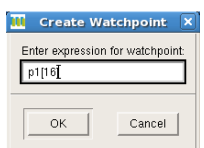


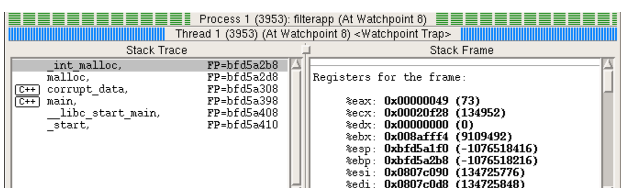
Boxes with solid lines around line numbers indicate code that exists at more than one location.

Watchpoints

- Watchpoints are set on a specific memory region
- Execution is stopped when that memory changes

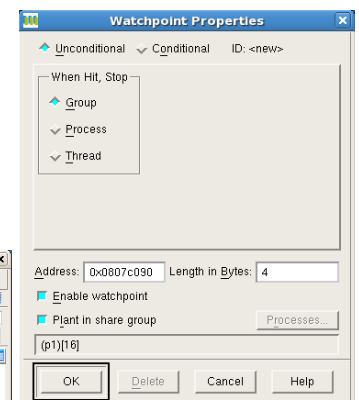
Action Point -> Create Watchpoint…

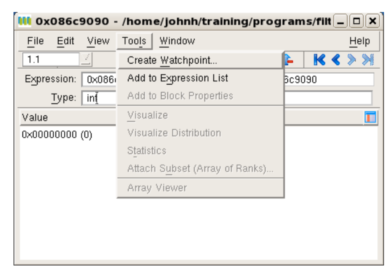




Watchpoints

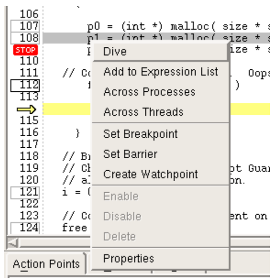
- Can create from a variable window using Tools -> Watchpoint





Watchpoints

- Can create from right-click on variable in Source pane



Watchpoints

- Watchpoints are set on a memory region, not a variable
- Watch the variable scope and disable watchpoints when a variable is out of scope
- Can be conditional, just like other action points
  - Use $newval and $oldval in your evaluation to find unexpected changes in value (such as a loop value changing by more than 1)

LAB 1: THE BASICS

DATA MONITORING AND VISUALIZATION

Diving on Variables

You can use Diving to:

… get more information

… open a variable in a Variable Window.

… chase pointers in complex data structures

… refocus the Process window Source Pane

You can Dive on:

… variable names to open a variable window

… function names to open the source in the Process Window.

… processes and threads in the Root Window.

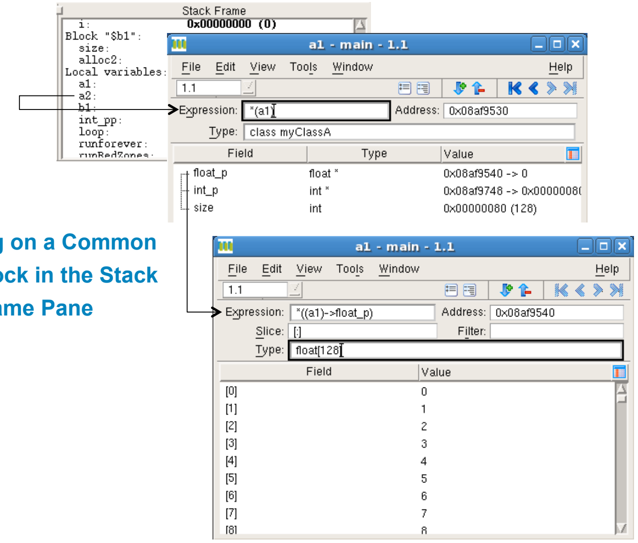
How do I dive?

•Double-click the left mouse button on selection

•Single-click the middle mouse button on selection.

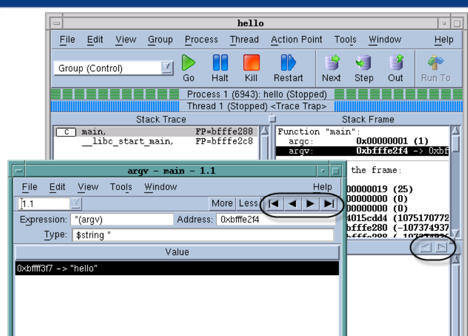
•Select Dive from context menu opened with the right mouse button

Diving



Diving on a Common Block in the Stack Frame Pane

Undiving

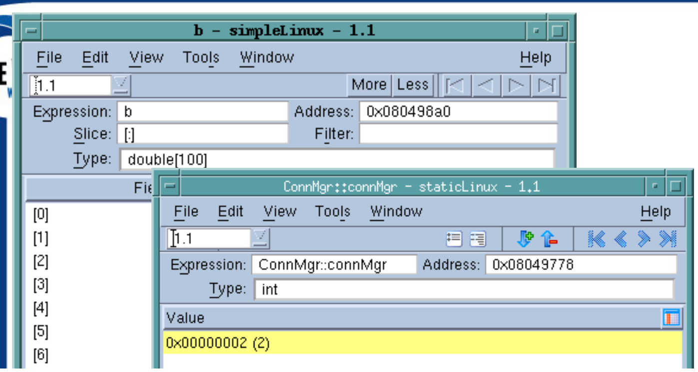


In a Process Window: retrace the path that has been explored with multiple dives.

In a Variable Window: replace contents with the previous contents.

You can also remove changes in the variable window with Edit > Reset Default.

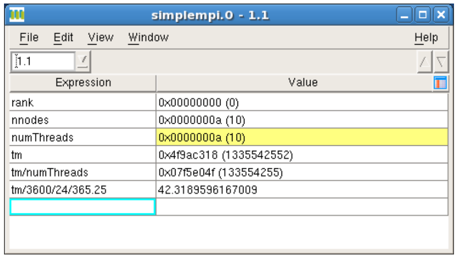
The Variable Window



Editing Variables

- Window contents are updated automatically
- Changed values are highlighted
- "Last Value" column is available
- Click once on the value
- Cursor switches into edit more
- Esc key cancels editing
- Enter key commits a change
- Editing values changes the memory of the program

Expression List Window

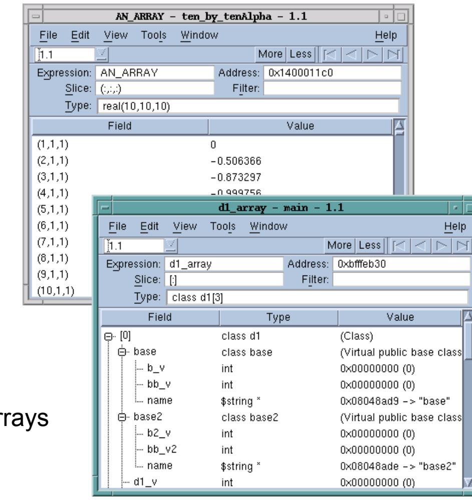


Add to the expression list using contextual menu with right-click on a variable, or by typing an expression directly in the window

- 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

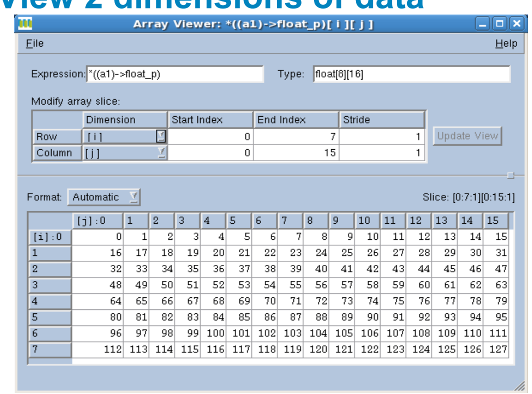
Data Arrays



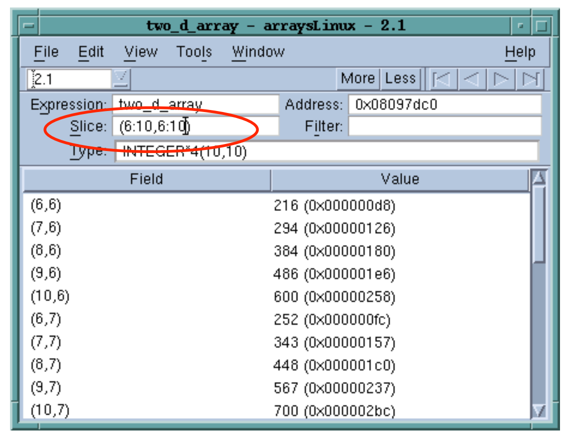
Structure Arrays

Array Viewer

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

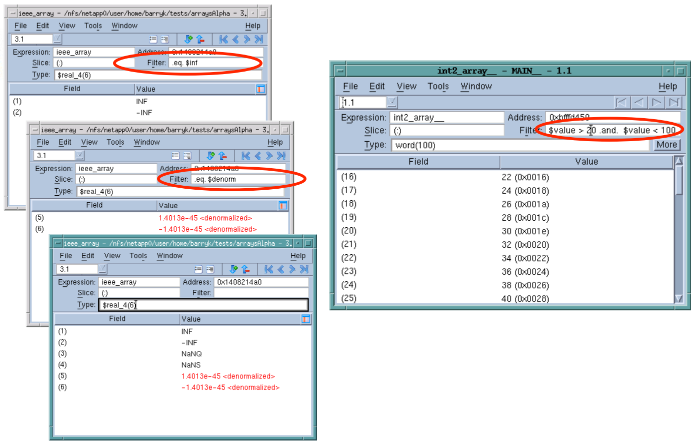


Slicing Arrays

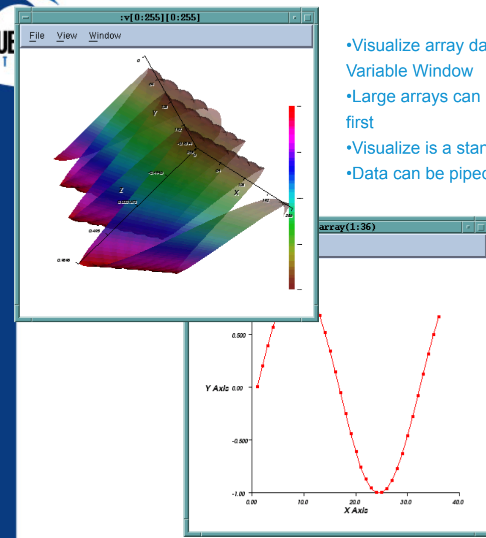


Slice notation is `[start:end:stride]`

Filtering Arrays



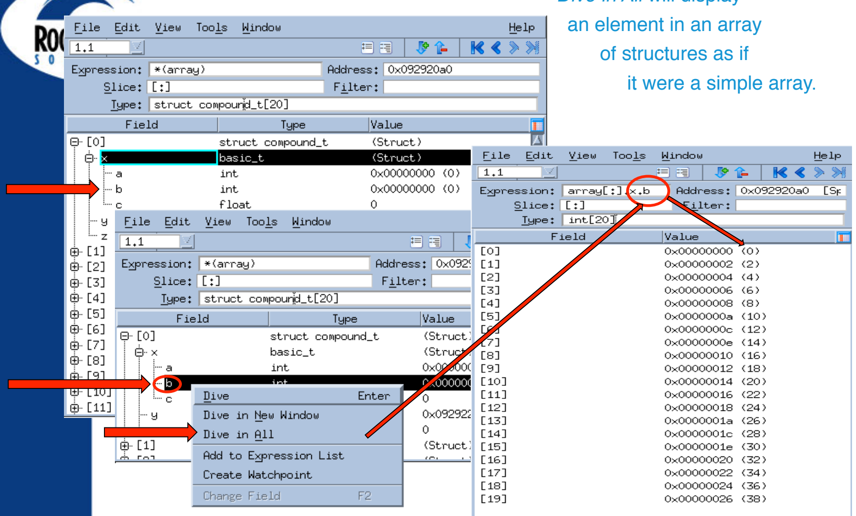
Visualizing Arrays



- 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.

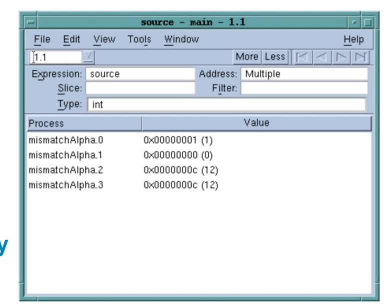
Dive in All

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



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.**



Typecasting Variables

- Edit the type of a variable
- View data as type…
- Often used with pointers

Type Casts Read from Right to Left

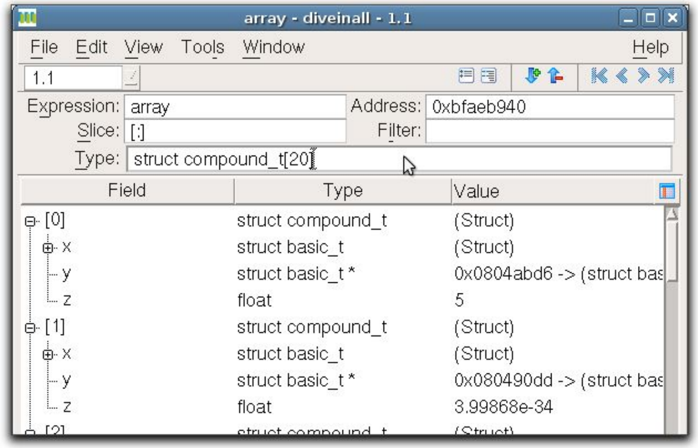
- int[10]* Pointer to an array of 10 int
- int*[10] Array of 10 pointers to int

- **Cast** float * **to** float [100]* **to see a dynamic array's values**
- **Cast to built-in types like** $string **to view a variable as a null-terminated string**
- **Cast to** $void **for no type interpretation or for displaying regions of memory**

The Bottom Line

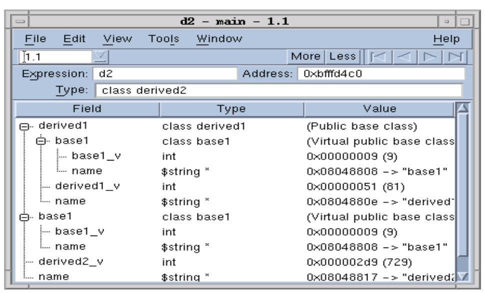
Give TotalView a starting memory address and you can tell TotalView how to interpret your memory from that starting location.

Typecasting a Dynamic Array



C++ Class Hierarchies

Variable Window shows class hierarchy using indentation



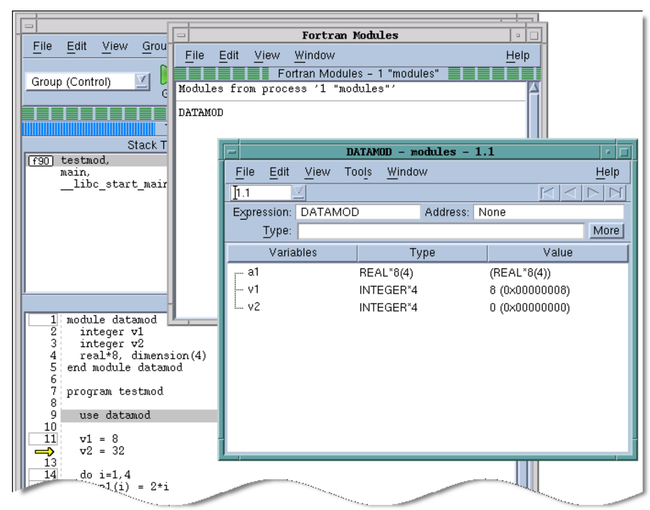
Example:

- derived2 inherits from base1 and derived1
- derived1 inherits from base1

Note:

- Virtual public base classes appear each time they are referenced
- The vtable entry here is part of the C++ implementation but can provide useful information

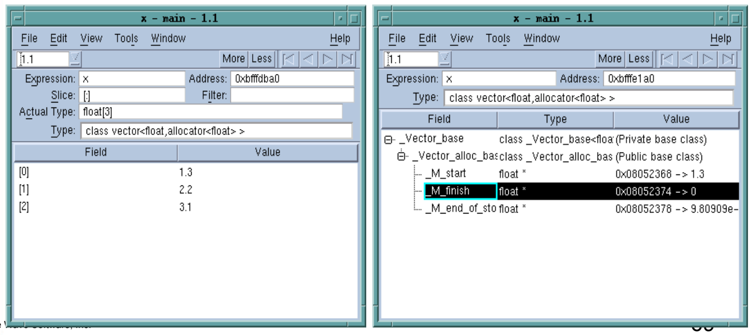
Fortran 90 Modules
Tools > Fortran Modules



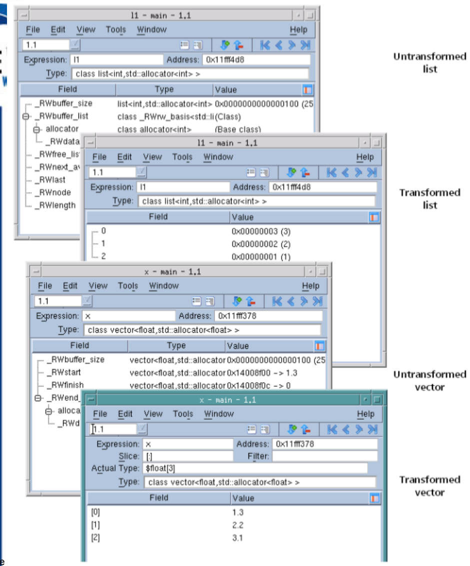
STLView

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



STLView



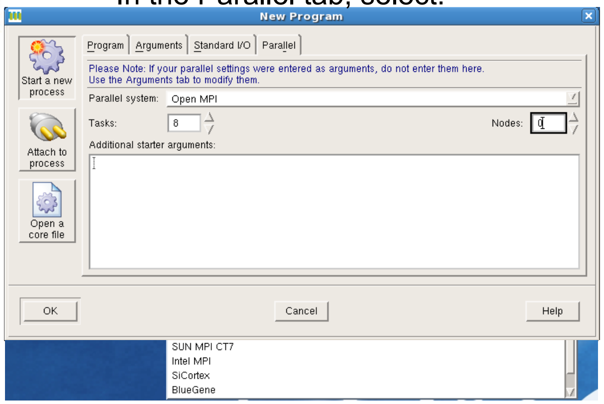
STLView transforms templates into readable and understandable information

LAB 2: VIEWING, EXAMING,WATCHING AND EDITING DATA

DEBUGGING FOR PARALLEL APPLICATIONS

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

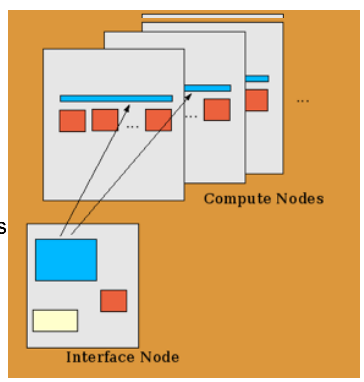
TotalView Startup with MPI

IBM	`totalview poe –a myprog –procs 4 –rmpool 0`
QUADRICS Intel Linux under SLURM	`totalview srun –a –n 16 –p pdebug myprog`
MVAPICH Opteron Linux under SLURM	`totalview srun –a –n 16 –p pdebug myprog`
SGI	`totalview mpirun –a myprog –np 16`
Sun	`totalview mprun –a myprog –np 16`
MPICH	`mpirun –np 16 –tv myprog`
MPICH2 Intel MPI	`Totalview python –a 'which mpiexec' –tvsu –np 16 myprog`

The order of arguments and executables is important, and differs between platforms.

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

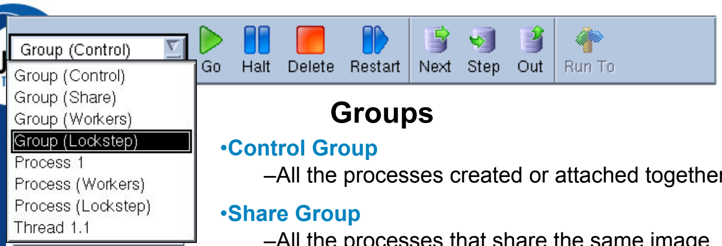


Provides Robust, Scalable and efficient operation with Minimal Program Impact

Process Control Concepts

- **Each process window is always focused on a specific process.**
- **Process focus can be easily switched**
  - **P+/P-, Dive in Root window and Process tab**
- **Processes can be 'held' - they will not run till unheld.**
  - **Process > Hold**
- **Breakpoints can be set to stop the process or the group**
- **Breakpoint and command scope can be simply controlled**

Basic Process Control

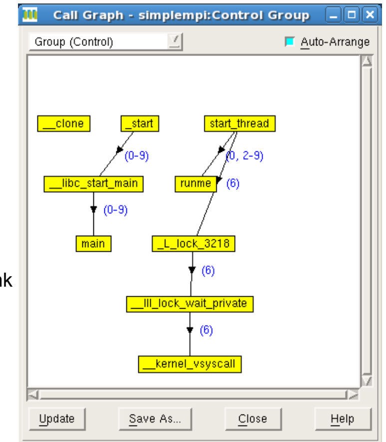


Groups

- **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

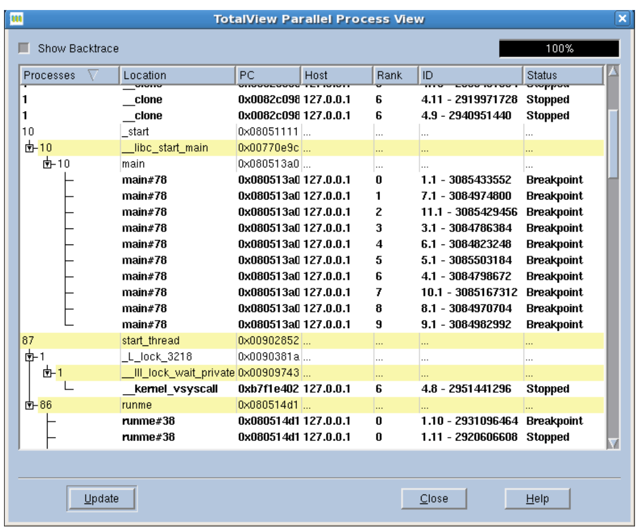
Call Graph

- **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.

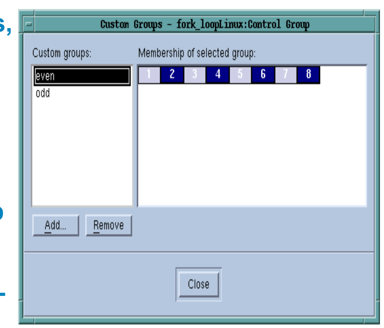
Parallel Back Trace



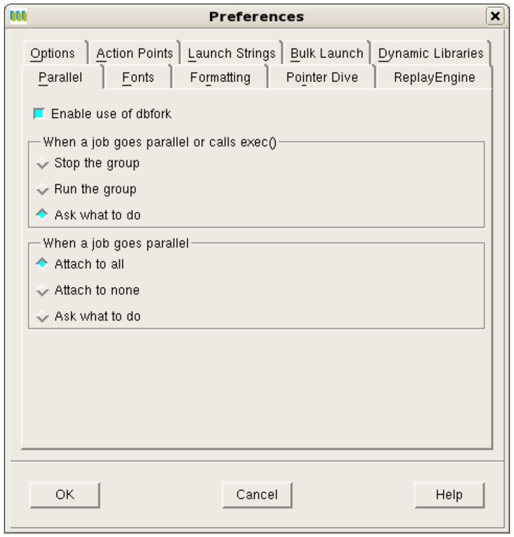
Processes	Location	PC	Host	Rank	ID	Status
1	__clone	0x0082c098	127.0.0.1	6	4.11 - 2919971728	Stopped
1	__clone	0x0082c098	127.0.0.1	6	4.9 - 2940951440	Stopped
10	_start	0x08051111	...	...	...	...
10	__libc_start_main	0x00770e9c	...	...	...	...
10	main	0x080513a0	...	...	...	...
	main#78	0x080513a0	127.0.0.1	0	1.1 - 3085433552	Breakpoint
	main#78	0x080513a0	127.0.0.1	1	7.1 - 3084974800	Breakpoint
	main#78	0x080513a0	127.0.0.1	2	11.1 - 3085429456	Breakpoint
	main#78	0x080513a0	127.0.0.1	3	3.1 - 3084786384	Breakpoint
	main#78	0x080513a0	127.0.0.1	4	6.1 - 3084823248	Breakpoint
	main#78	0x080513a0	127.0.0.1	5	5.1 - 3085503184	Breakpoint
	main#78	0x080513a0	127.0.0.1	6	4.1 - 3084798672	Breakpoint
	main#78	0x080513a0	127.0.0.1	7	10.1 - 3085167312	Breakpoint
	main#78	0x080513a0	127.0.0.1	8	8.1 - 3084970704	Breakpoint
	main#78	0x080513a0	127.0.0.1	9	9.1 - 3084982992	Breakpoint
87	start_thread	0x00902852	...	...	...	...
1	_L_lock_3218	0x0090381a	...	...	...	...
1	__lll_lock_wait_private	0x00909743	...	...	...	...
	__kernel_vsyscall	0xb7f1e402	127.0.0.1	6	4.8 - 2951441296	Stopped
86	runme	0x080514d1	...	...	...	...
	runme#38	0x080514d1	127.0.0.1	0	1.10 - 2931096464	Breakpoint
	runme#38	0x080514d1	127.0.0.1	0	1.11 - 2920606608	Stopped

User Defined Groups

- **Group > Custom Groups, to create a process group of some other specification**
- **Group Membership shown in Processes Tab**
- **User defined groups appear in the "Go" drop-down menu**

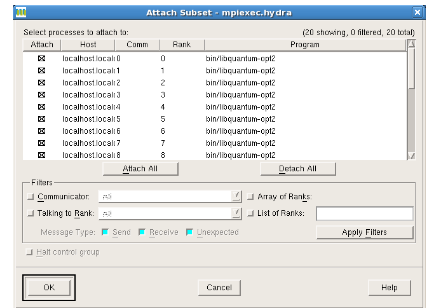


Preferences



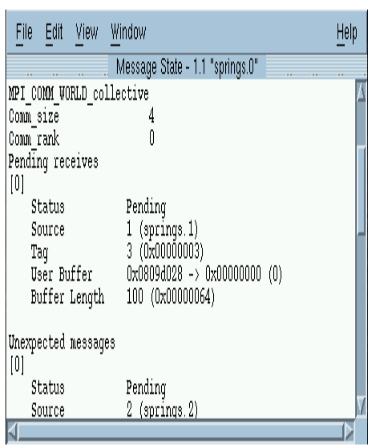
Subset Attach

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



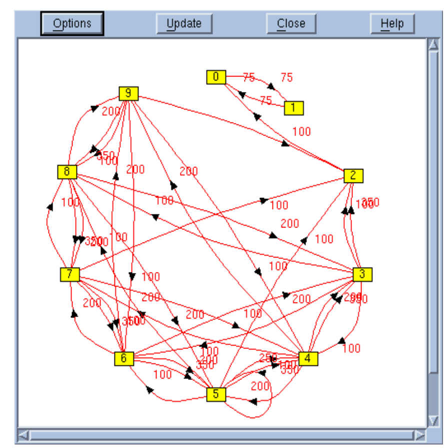
View MPI Message Queues

- Information visible whenever MPI rank processes are halted
- Provides information from the MPI layer
  - Unexpected messages
  - Pending Sends
  - Pending Receives
- Use this info to debug
  - Deadlock situations
  - Load balancing
- May need to be enabled in the MPI library
  - --enable-debug



Message Queue Graph

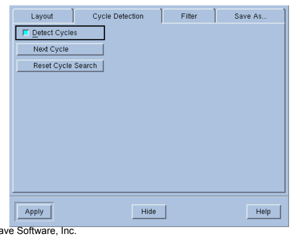
- Hangs & Deadlocks
- Pending Messages
  - Receives
  - Sends
  - Unexpected
- Inspect
  - Individual entries
- Patterns

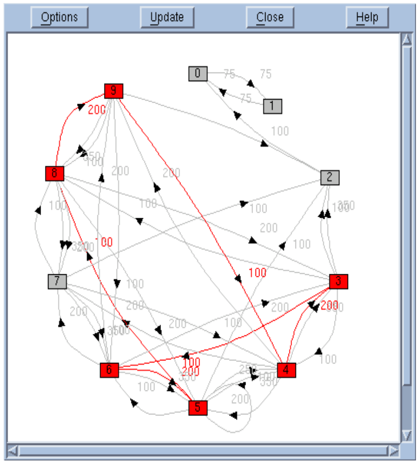


Message Queue Graph

Message Queue Debugging

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





Strategies for Large Jobs

- **Reduce N**
  - **Problem:** Each process added requires overhead
  - **Strategy:** Reduce the number of processes TotalView is attached to
    - Simply reducing N is best, however data or algorithm may require large N
  - **Technique:** subset attach mechanism
- **Focus Effort**
  - **Problem:** Some debugger operations are much more intensive than others, and when multiplied by N this could be significant
  - **Strategy:** Reduce the interaction between the debugger and the processes
  - **Technique:** Use TotalView's process control features to
    - Avoid single stepping
    - Focus on one or a small set of processes

LAB 3: EXAMINING AND CONTROLLING A PARALLEL APPLICATION

ADVANCED ASYNCHRONOUS CONTROL

Why Asynchronous Control

- Parallel codes are very difficult to debug
- Breaking down the problem to smaller pieces helps narrow down issues
- Stepping individual processes, threads, or groups can help narrow down a problem

TotalView Asynchronous Control Features

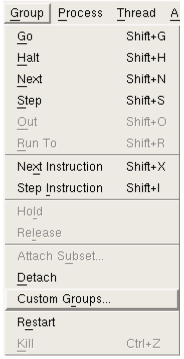
- Built in control groups
- User-defined control groups
- Action points can target threads, processes or groups
- Typical debugging commands can target groups or individual processes and threads (Next, Step, etc.)

Groups

- By default, TotalView defines the following groups:
  - Control Group: everything
  - Share Group: all processes and their threads with same image
  - Workers Group: all threads in all control group processes
  - Lockstep Group: all threads at the same breakpoint
  - Process: current process with debugger focus
  - Process Workers: all threads in the process
  - Process Lockstep: all threads at the same breakpoint in one process
  - Thread: current thread with focus
- Only the Workers group can be modified by the user
  - CLI, use dworker 0 to remove from the workers group or dworker 1 to add

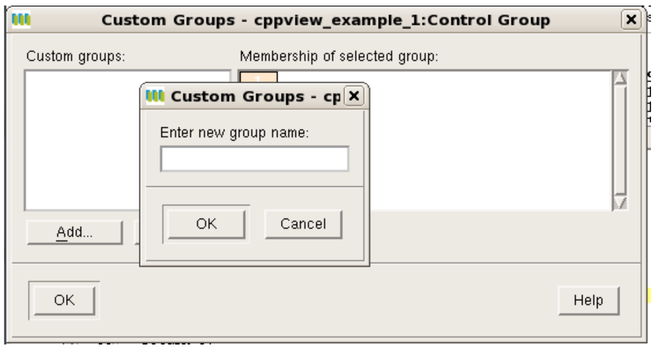
Customizing Groups

- Only the Workers group can be modified by the user
  - CLI, use dworker 0 to remove from the workers group or dworker 1 to add
- Create a Custom Group from the Group menu



Creating a Custom Group

- Enter the group name
- Select processes to be members of the group
- Add… button to create more groups



Custom Groups in the CLI

- In the CLI, use the dgroups command to create & modify groups

  dgroups –new *t|p* [–g *groupname]* [*id_list]*

  dgroups –add [–g *groupname] [id_list]*

  dgroups –remove [–g *groupname] [id_list]*

  dgroups –intersect [–g *groupname id_list]*

  dgroups –delete [–g *groupname]*

  t or p – can also use thread or process, is it a thread or process group

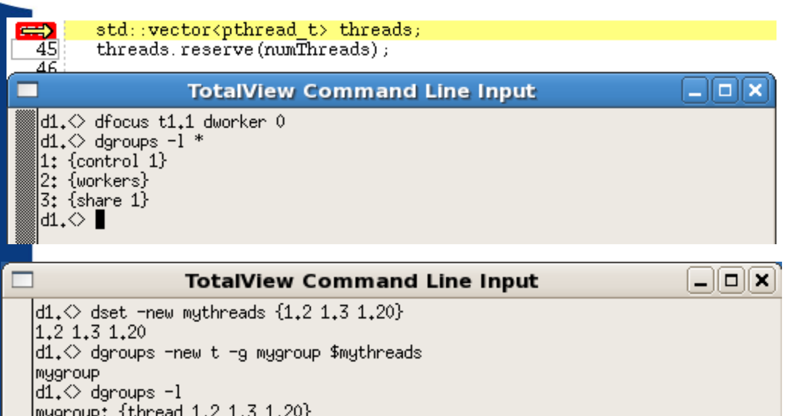
  groupname is your name for the new group

  id_list is a TCL list of ids to add to the new group

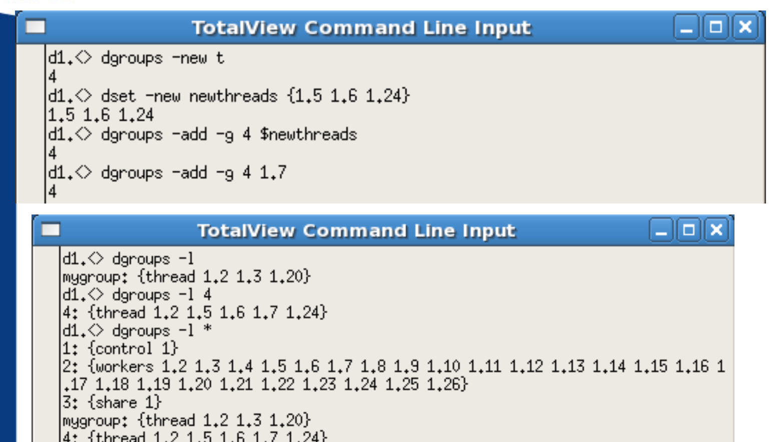
- You can also use dworker to add/remove threads from the process workers group

  dfocus t1.1 dworker 0

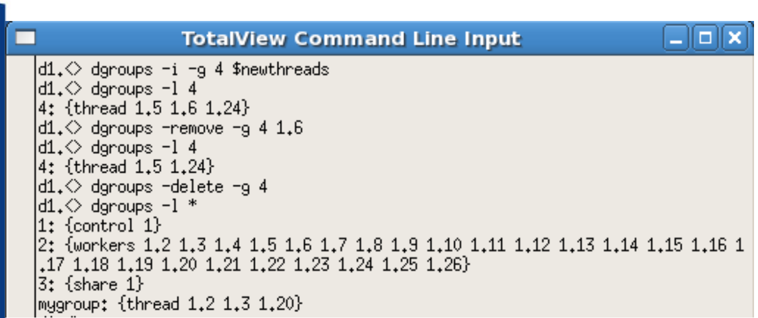
Custom Groups in CLI



Custom Groups in CLI



Custom Groups in CLI



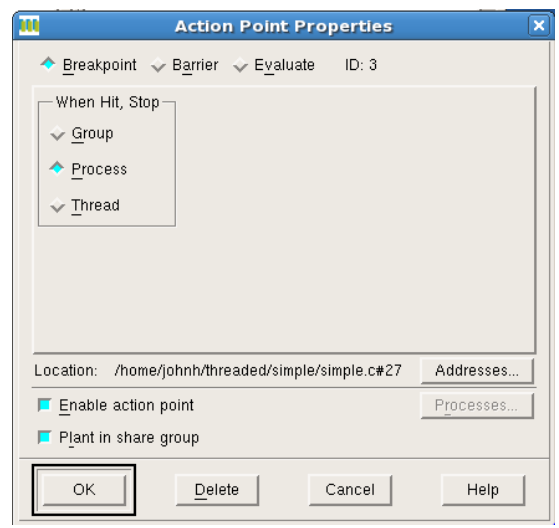
Breakpoints

- **Control where they are planted, defaults to the Share Group**
  - Uses the SHARE_ACTION_POINT variable, true plants in the Share Group, false plants in the focus process only
- **Control what is stopped by hitting the breakpoint, the group, the process, or just the thread**
  - Uses the STOP_ALL variable set to: group, process, or thread
  - Use the –g, -p, or –t flag to dbreak in the CLI to override

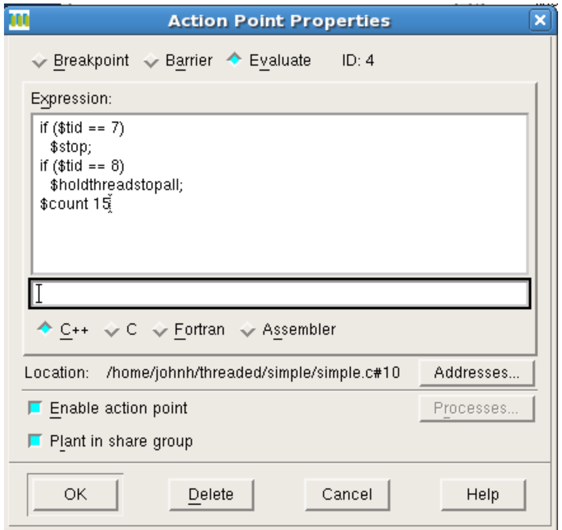
Breakpoints

- **Control what is stopped and finer control over when it is stopped by using eval option and writing test code**
  - Code can be C, C++, FORTRAN 77, Fortran 9x, or assembler
  - Can use TotalView-specific values and commands like $tid, $pid, $stop
  - Use –lang and –e flags to dbreak in CLI

Breakpoints in UI



Eval Breakpoints in UI



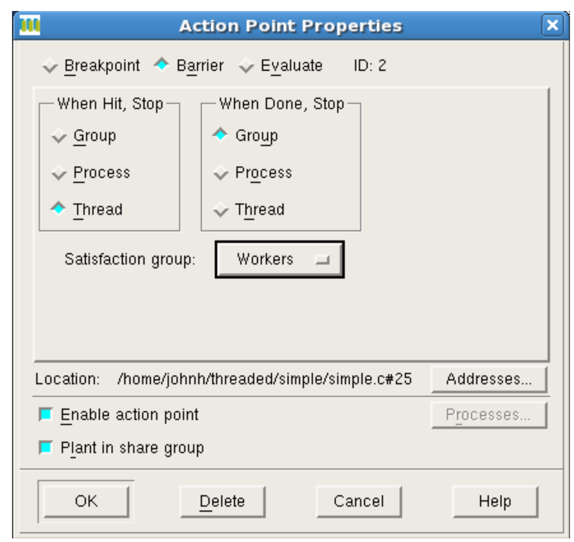
Barriers

- Control where they are planted, defaults to the Share Group
  - Uses the SHARE_ACTION_POINT variable, true plants in the Share Group, false plants in the focus process only
- Control what is stopped by hitting the breakpoint, the group, the process, or just the thread
  - Uses the BARRIER_STOP_ALL variable set to: group, process, or none
  - Use –stop_when_hit flag in CLI to override default

Barriers

- Control what is stopped when the barrier is satisfied, the group or the process
  - Uses the BARRIER_STOP_WHEN_DONE variable set to: group, process, or none (same as process for a process barrier)
  - Use –stop_when_done flag in CLI to override default

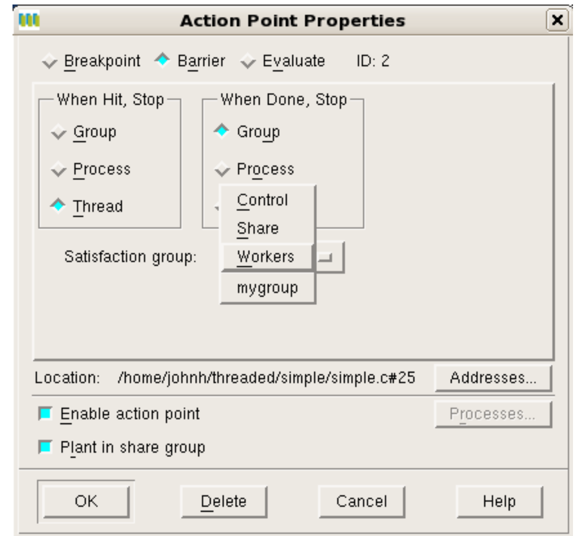
Barriers Satisfaction Group in UI



Barriers – Satisfaction Group

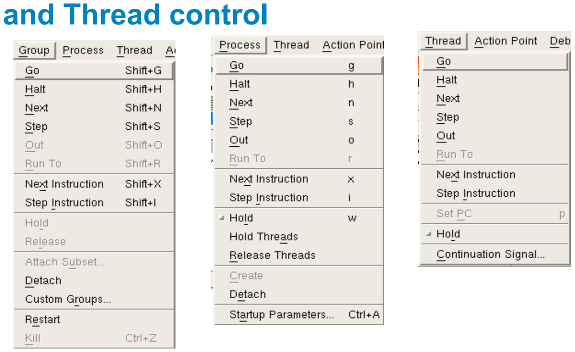
- **Satisfaction Group determines how many times barrier needs to be reached before it is satisfied and can release all threads that have reached it.**
  - In the UI, you can select from Control group, Process, or Workers
  - If you have created custom groups, they should also appear in the drop down list in the UI
  - CLI uses the intersection of the current focus and the share group to determine the satisfaction group
  - BE SURE YOUR ENTIRE SATISFACTION GROUP CAN REACH THE BARRIER OR YOU CAN BE DEADLOCKED
  - Barriers can also create deadlocks if a thread held by the barrier is holding a lock or another thread is dependent on a held thread's output, etc.

Barriers – Select Satisfaction Group UI



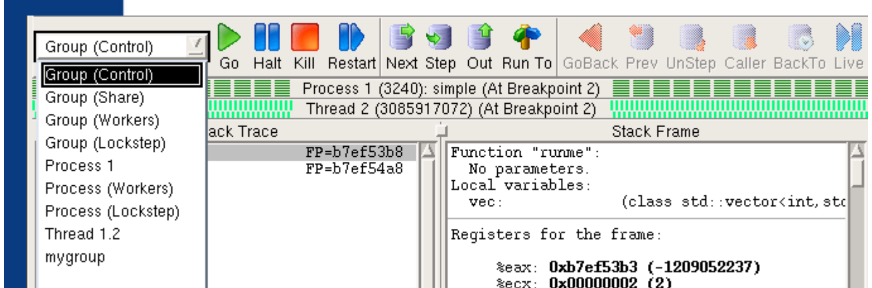
Asynchronous Controls

- **Once things are stopped, now what?**
- **CLI commands operate on the current focus, so you can step, next, go, etc. based on your focus of a group, process, or thread**
- **UI has separate menus for Group, Process, and Thread control**



Asynchronous Controls

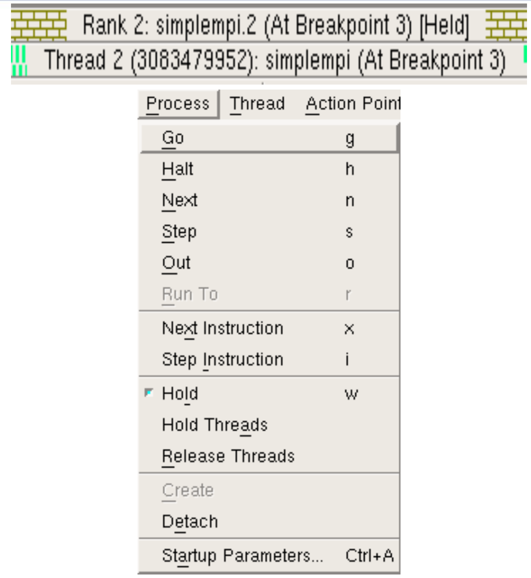
- **UI also has a drop down list control to control what the buttons will affect**



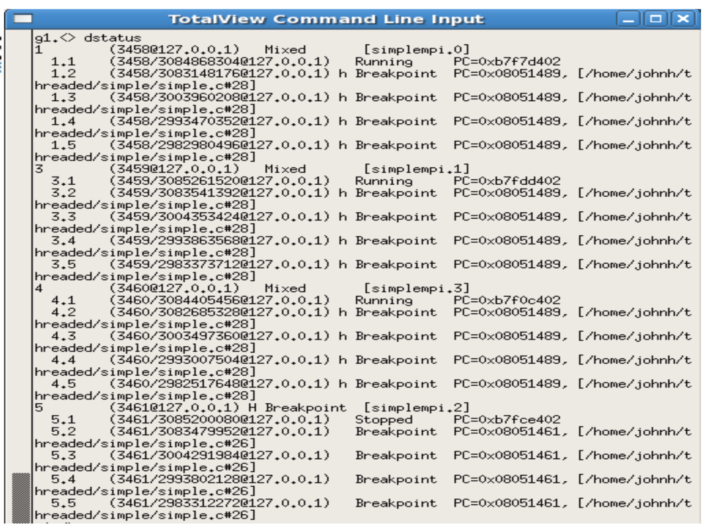
Holds

- Group, Process, Thread can all be held
- Anything that is held won't run or step again until it is unheld
- Hold status is indicated in dstatus, in the Process Window, and also under the toolbar in the UI
- Hold status also applies to anything that is held at a barrier prior to the satisfaction group completing the barrier

Holds



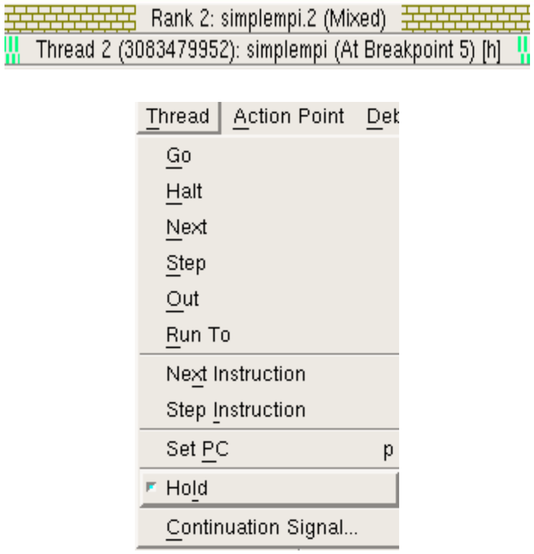
Holds - CLI



Holds

- When something is held, you must "un-hold" it
- Set focus to the held thread/process, then release the hold

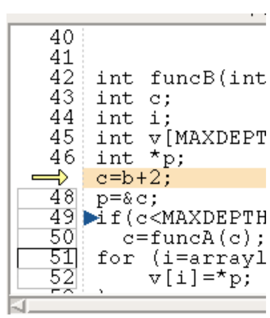
Holds - releasing

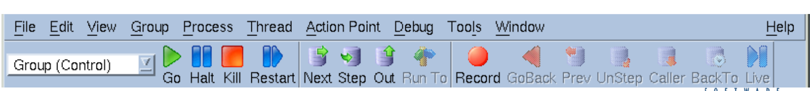


LAB 9: ASYNCHRONOUS CONTROL

ReplayEngine

- **Reverse Debugging: Radically simplify your debugging**
  - Captures and Deterministically Replays Execution
    - Not just logging or "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





ReplayEngine modes

Record Mode

- Captures Input
  - Function calls
  - Network and file IO
- Captures Non-Determinism
  - Forces single thread execution at a time
  - Records context switches
- Stores "images" of memory contents throughout runtime
- Can be used with the TotalView Memory Debugger.
- Can be activated during the middle of the run

Replay Mode

- Provides you with the ability to review any part of the program execution (see all variables) from the beginning of the run to the current time
- Like a "rewind" button on a DVR
- Use breakpoints, watchpoints, and some conditional breakpoints when running forward or backwards in replay mode
- Searches for relevant events behind the scenes but provides a streamlined "step backwards" experience
- Provides Determinism within a debugging session

ReplayEngine controls

Replay Engine – The right way to debug

Next Step forward over functions

Prev Step *backward* over functions

Step Step forward into functions

UnStep Step *backward* into functions

Out Advance forward out of current Function, after the call

Caller Advance backward out of current Function, to before the call

Run To Advance forward to selected line

BackTo Advance backward to selected line

Go Run forward

GoBack Run *backward*

Live Advance forward to "live" session

 |

Demo

 |

Comparative Debugging

 |

Comparative Debugging with TotalView

- Two options
  - Separate TV sessions, one for A and the other for B
  - Single TotalView session attached to both A and B
- Separate sessions
  - On different architectures
  - Separate batch submissions
  - Drive as two separate parallel jobs
  - Some tricks for comparing data which we will discuss later

 |

Debugging two programs in one session of TotalView

- TotalView handles Multiple Program Multiple Data
  - TotalView does not assume that all the parts of a parallel job are identical
  - Part of the same control group if they are launched from the same mpiexec
- TotalView can also launch a second process or parallel job while attached to the first one
  - These two are part of separate control groups
  - They can be placed in the same control group after the fact though
  - Once in the same control group you can issue single commands that apply to both processes or sets of processes.
- This can be augmented by using ReplayEngine in both jobs ..
  - Follow difference back to root causes

 |

Techniques for comparative debugging with TotalView

- Use background color setting to distinguish the two debug sessions
  - Requires two instances of the debugger
- Use the ability to save breakpoints to share a breakpoint set between the two instances
  - TotalView will try to be smart about restoring breakpoints ... it can deal with small code changes such as line number offsets due to adding lines to functions earlier in the program.
- Consider using scripted commands for any complex operations

 |

Case Study: Physical Simulation

- Semi-Automated Parallel Program Debugging

Jeff Keasler
LLNL

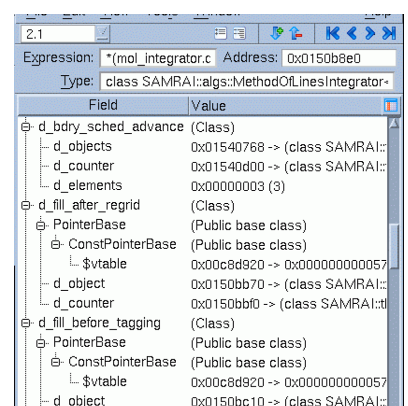
Alejandro Hernandez
UC Santa Barbara, LLNL Intern

The following materials are adapted from a BOF talk at SC11 and are used with the authors permission

 |

Debugging of Large-Scale OO Programs is Increasingly Challenging

- Rich OO design patterns are already here
- Debugging through an object hierarchy has proven to be difficult
  - Object inheritance
  - Objects composed of other objects
- Most debuggers display objects as a collection of atomic types
  - Often displays irrelevant data to the code developer
- Need a more automated way to isolate bugs in rich object environments



 |

Some Common Debugging Issues in Science Codes are Difficult to Address

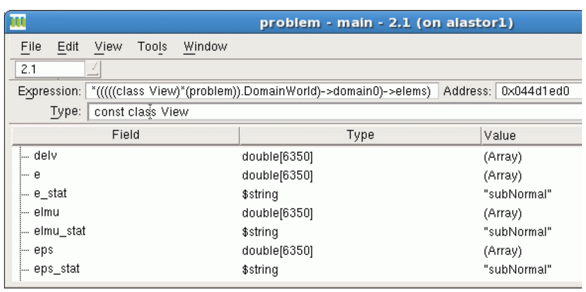
- Need a way to debug halo-layer issues.
- Need an efficient way to compare two versions of the same code
  - Algorithm changes (New algorithms, updates, etc.)
  - Compiler porting (new flags/version, icpc, g++, etc.)
  - Platform porting (x86 cluster, BG/P, etc.)

C++View Interface Provides Custom Debugging Support

- TV_display_type(const class X *obj)
  - A user defined function that can be overloaded for each specific class/ struct/union type of interest.
  - In TV, diving into an object of type *class X* invokes the associated TV_display_type() function, if present.
  - Unrestricted use of C++ within these functions.
- TV_add_row(char *name, char *type, void *ptrToType)
  - This function is called from within a TV_display_type() function to display a row of data in Totalview's data display window. Example:

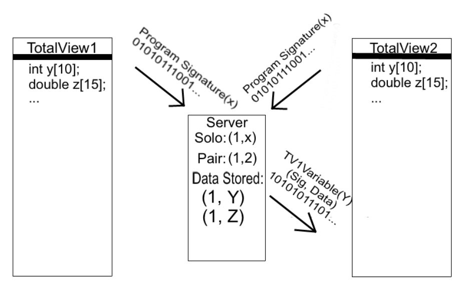
```
TV_add_row("count", "int", &obj->count) ;
```


Advanced Example – NaN/Inf/etc.



This example shows how fields containing bogus values can be flagged in the output. Here, whenever bad values are detected, an extra line of output is created describing the nature of the numerical errors.

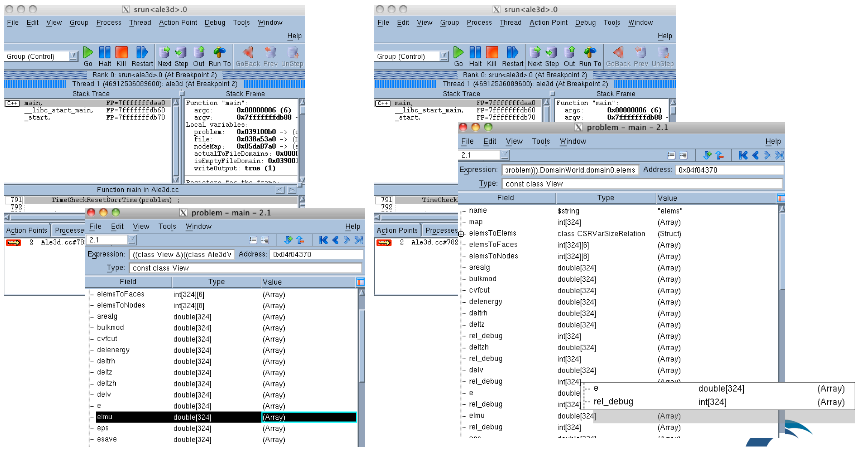
Debugging Large-Scale OO Programs Can Be Simplified



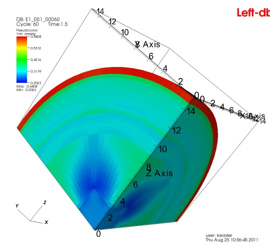
Semi-Automated Data Comparison Debugging

- Semi-Automated Data Comparison Debugging:
  - Compare two different versions of the same code to search for data differences
  - Data compared within the program
  - Comparison information displayed within the TotalView debugger

Differences Displayed as Integrated Part of Debugger

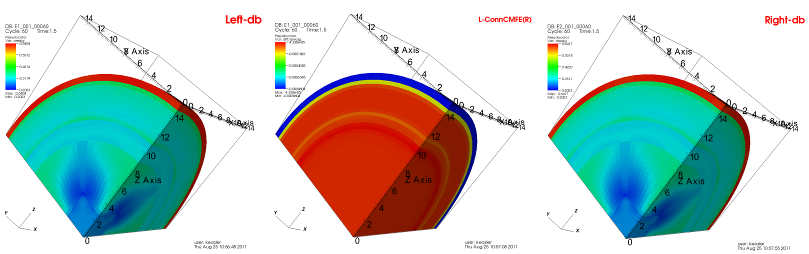


The Next Step is To Visualize Field Data…



Diving on a Field will display the field with respect to a submesh context

…and Visualize Computed Differences between code Versions



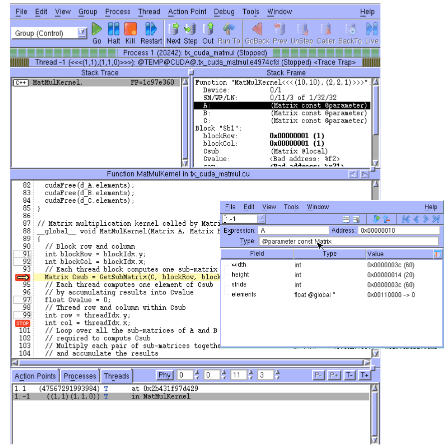
Semi-Automated Data-Comparison Debugging Provides an Additional Tool for Finding Bugs

- Time saver
  - Comparison of a collection of data, possibly very large, is done quickly for the user
- Easily integrated into pre-existing programs
  - Does not interfere with pre-existing code
- Implemented once and used through the entire development cycle of the application

CUDA/OpenACC Debugging

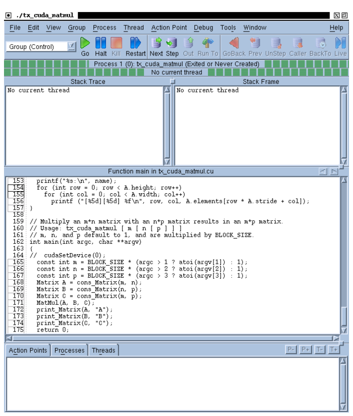
- Some of the slides here are marked with
  – These contain content developed by Sandra Wienke and are used with permission.

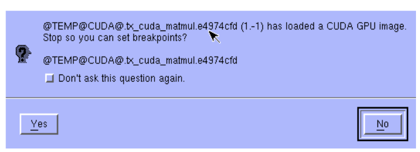
TotalView for CUDA



- Characteristics
  – Full visibility of both Linux threads and GPU device threads
  – Fully represent the hierarchical memory
  – Supports Unified Virtual Addressing and GPUDirect
  – Thread and Block Coordinates
  – Device thread control
  – Handles CUDA function inlining and CUDA stacks
  – Support for C++ and inline PTX
  – Reports memory access errors
  – Handles CUDA exceptions
  – Multi-Device Support
  – Can be used with MPI

Starting TotalView

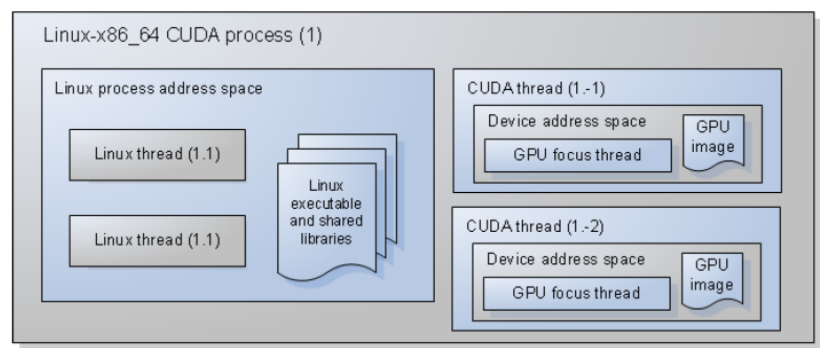




When a new kernel is loaded you get the option of setting breakpoints

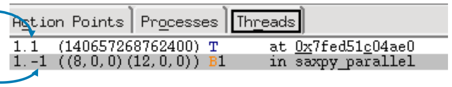
- You can debug the CUDA host code using the normal TotalView commands and procedures

TotalView CUDA Debugging Model



CUDA Debugging

- **Debugger thread IDs in Linux CUDA process**
  - → **Host thread: positive no.**
  - → **CUDA thread: negative no.**



- **GPU thread navigation**
  - → **Logical coordinates: blocks (3 dimensions), threads (3 dimensions)**

  - → **Physical coordinates: device, SM, warp, core/lane**

  - → **Only valid selections are permitted**

 |

CUDA Debugging

- **Warp: group of 32 threads**
  - → **Share one PC**
  - → **Advance synchronously**

Problem: Diverging threads

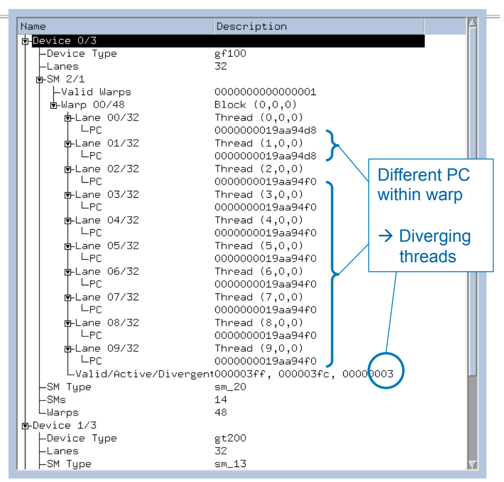
```
if (threadIdx.x > 2)
{...} else {...}
```

- **Single Stepping**
  - → **Advances all GPU hardware threads within same warp**
  - → **Stepping over a __syncthreads() call advances *all* threads within the block**
- **Advancing more than just one warp**
  - → **"Run To" a selected line number in the source pane**
  - → **Set a breakpoint and "Continue" the process**
- **Halt**
  - → **Stops all the host and device threads**

 |

CUDA Debugging

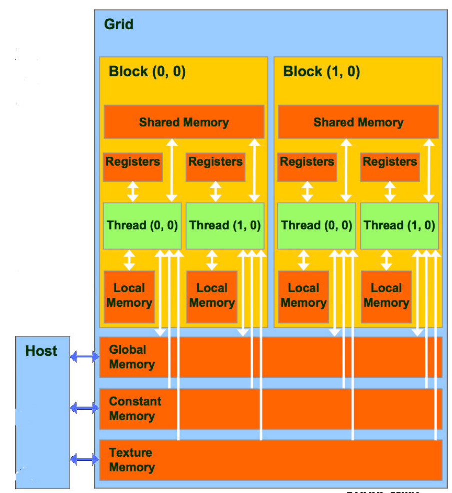
- **Displaying CUDA device properties**
  - → **"Tools" - "CUDA Devices"**
  - → **Helps mapping between logical & physical coordinates**
- **PCs across SMs, warps, lanes**
  - → **GPU thread divergence?**



 |

GPU Memory Hierarchy

- **Hierarchical memory**
  - **Local (thread)**
    - **Local**
    - **Register**
  - **Shared (block)**
  - **Global (GPU)**
    - **Global**
    - **Constant**
    - **Texture**
  - **System (host)**



 |

TotalView Type Storage Qualifiers

@parameter Address is an offset within parameter storage.

@local Address is an offset within local storage.

@shared Address is an offset within shared storage.

@constant Address is an offset within constant storage.

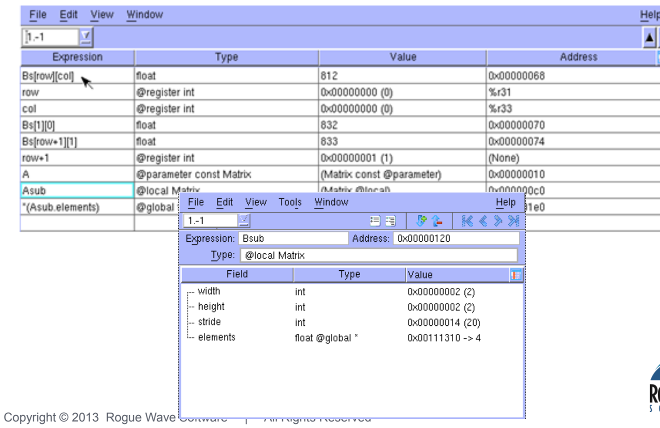
@global Address is an offset within global storage.

@register Address is a PTX register name.

 |

CUDA Variables

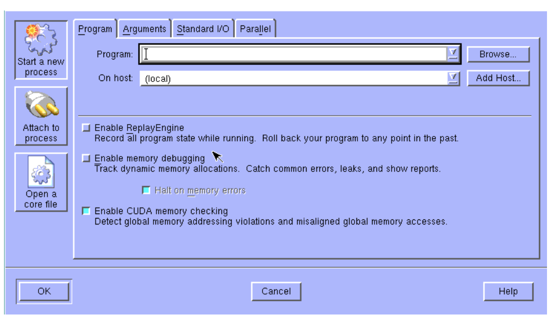
- **Storage qualifiers appear in the data type**



 |

CUDA Segmentation Faults

- **TotalView displays segmentation faults as expected**
  - **Must enable CUDA memory checking**



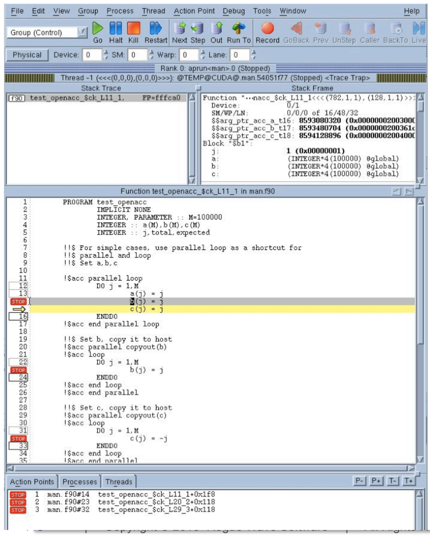
 |

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;**

 |

TotalView for OpenACC



OpenACC
DIRECTIVES FOR ACCELERATORS

- Step host & device
- View variables
- Set breakpoints

- Compatibility with Cray CCE 8 OpenACC now
- Investigating PGI and CAPS support

CUDA Debugging - Tips

- Check CUDA API calls
  - → All CUDA API routines return error code (`cudaError_t`)
  - → Or `cudaGetLastError()` returns last error from a CUDA runtime call
  - → `cudaGetErrorString(cudaError_t)` returns corresponding message
  1. Write a macro to check CUDA API return codes or use *SafeCall* and *CheckError* macros from *cutil.h* (NVIDIA GPU Computing SDK)
  2. Use TotalView to examine the return code
     - → Evaluate the CUDA API call in the expression list
     - → If needed, dive on the error value and typecast it to an `cudaError_t` type
     - → You can also surround the API call by `cudaGetErrorString()` in the expression field and typecast it to `char[XX]*`

 |

CUDA Debugging - Tips

- Check + use available hardware features
  - → `printf` statements are possible within kernels (since Fermi)
  - → Use double precision floating point operations (since GT200)
  - → Enable ECC and check whether single or double bit errors occurred using `nvidia-smi -q` (since Fermi)
- Check final numerical results on host
  - → While porting, it is recommended to compare all computed GPU results with host results
  1. Compute check sums of GPU and host array values
  2. If not sufficient, compare arrays element-wise
  - → See TotalView's comparative debugging approach (Lab 3), e.g. statistics view

 |

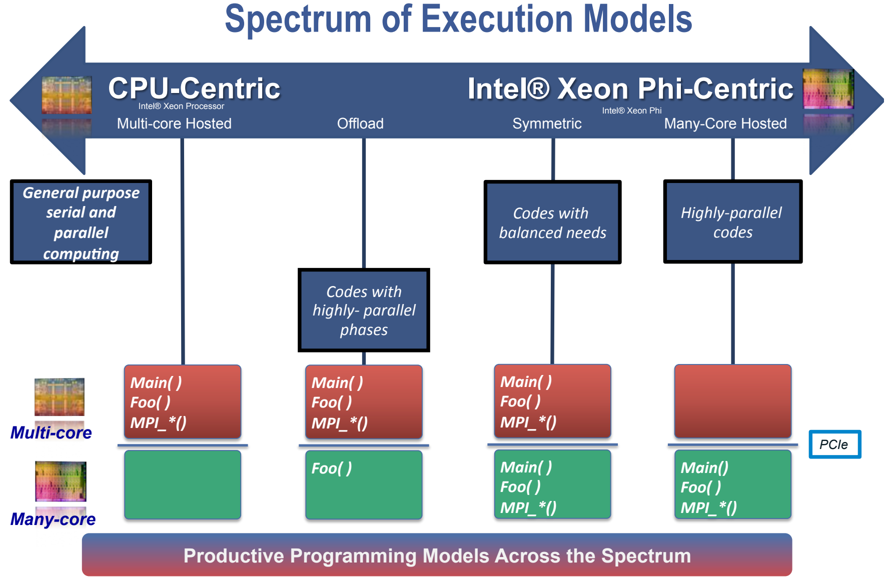
CUDA Debugging - Tips

- Check intermediate results
  - → If results are directly stored in global memory: dive on result array
  - → If results are stored in on-chip memory (e.g. registers) → tedious debugging
    - → TotalView: View of variables across CUDA threads not possible yet
    1. Create additional array on host for intermediate results with size `#threads * #results * sizeof(result)`
       Use array on GPU: each thread stores its result at unique index
       Transfer array back to host and examine the results
    2. If having a limited number of thread blocks: create additional array in shared memory within kernel function: `__shared__ myarray[size]`
       Use defines to exchange access to on-chip variable with array access
       Examine results by diving on array and switching between blocks
  - → Use filter, array statistics, freeze, duplicate, last values and watch points (see Lab 2)

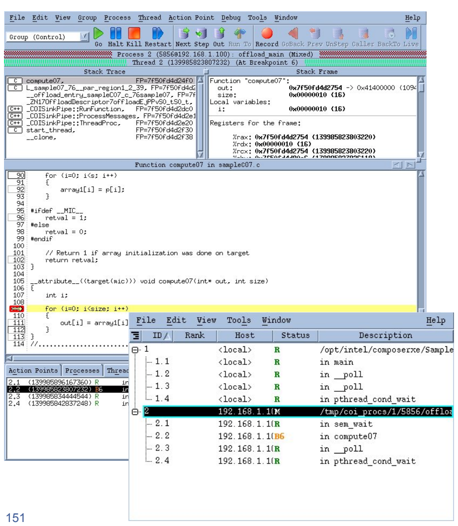
 |

Xeon Phi

Spectrum of Execution Models



Xeon Phi Port of TotalView



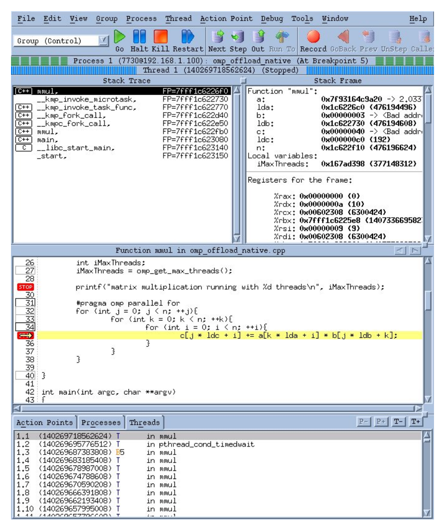
Key to Success:

Working closely with Intel Development Team

Key Features:

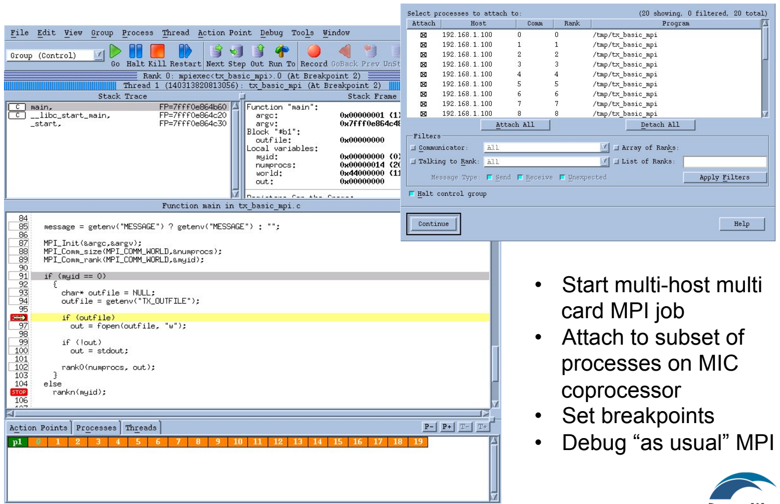
- Full visibility of both host and coprocessor threads
- Full support of MPI programs
- Symmetric debugging of heterogeneous applications with offloaded code
- Remote debugging of Xeon Phi-native applications
- Asynchronous thread control on both Xeon and Xeon Phi

Remote Debugging of Applications on Xeon Phi



- Just run as
  totalview –r hostN-micM <program>
- Attach to running application
- See thread private data
- Investigate individual threads
- Analyze core crashes on Xeon Phi

Debugging MPI Applications

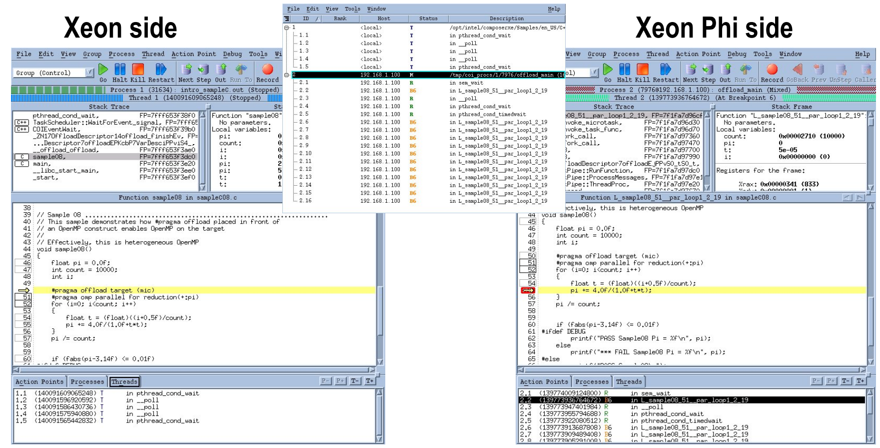


- Start multi-host multi card MPI job
- Attach to subset of processes on MIC coprocessor
- Set breakpoints
- Debug “as usual” MPI

 |

Debugging Applications with Offloaded Code

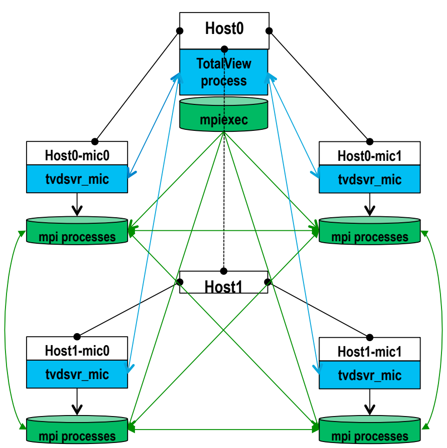
Xeon side **Xeon Phi side**



One debugging session for Xeon Phi-accelerated code

 |

Multi-host, Multi-card Phi-native MPI Debugging in TotalView 8.12



Conditions:

1. Each card has its own IP address and is accessible from front host node, running TotalView.

2a. TotalView is installed in global area and is accessible from each card in allocation, so that you can start tvd mic-server on each mic-card from the partition

OR

2b. You can copy tvdsvr using mic_native_server_launch_string

 |

Multi-host, Multi-card MPI Debugging in TotalView 8.12

Single server launch (default)

- totalview -args mpiexec -np 240 -hosts host1-mic0,host1-mic1,host2-mic0,host2-mic1 ./ tx_basic_mpi
- set env TVDSVRLAUNCHCOMMAND=<your ssh command to card> (ssh,micssh)
- Set TV::server_launch_string preference

MIC Native Launch

- totalview –mmic -args mpiexec -np 240 -hosts host1-mic0,host1-mic1,host2-mic0,host2-mic1 ./tx_basic_mp
- Set: dset TV::mic_native_server_launch_string {
  ssh -n %R "/bin/rm -f /tmp/tvdsvrmain%K"; //1
  scp %B/tvdsvrmain%K %R:/tmp/tvdsvrmain_mic; //2
  ssh -n %R -n "/tmp/tvdsvrmain%K -callback %L -set_pw %P -verbosity %V %F“ //3 }

1. Removes your previous tvdsvrmain_mic
2. Copies it from the installation directory to the /tmp/ directory on the coprocessor
3. Starts the server on the Xeon Phi coprocessor.

 |

Demo

SUPPORT AND QUESTIONS

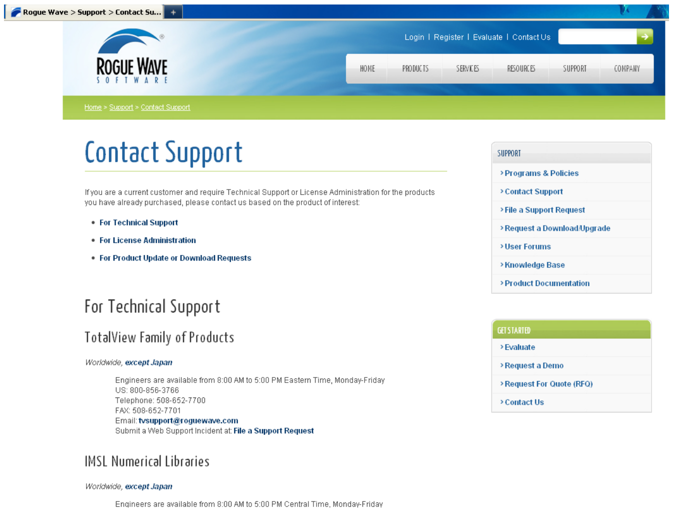
TotalView Customer Support

- Email: tvsupport@roguewave.com
- Use our web site for documentation, demos, FAQs and to contact support

TotalView Customer Support

http://www.roguewave.com

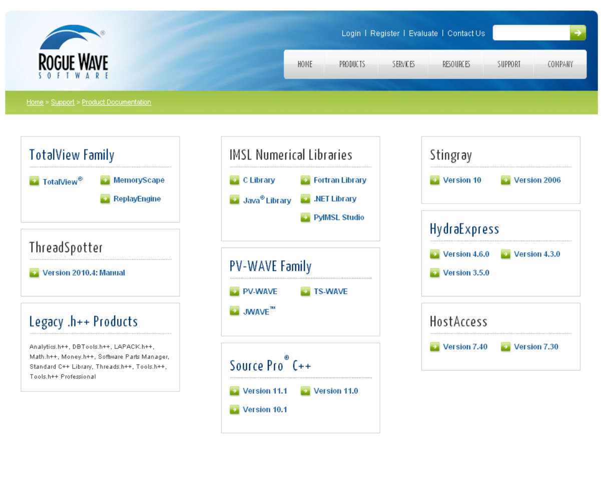
TotalView Customer Support



TotalView Documentation

162

TotalView Documentation



Thanks!

QUESTIONS?