



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

Chris Gottbrath, Product Manager
May 6th, 2013



TotalView

John Hollis
2

©Copyright 2013 Rogue Wave Software, Inc.

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

©Copyright 2013 Rogue Wave Software, Inc.

Basic Topics



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

©Copyright 2013 Rogue Wave Software, Inc.

Intermediate Topics



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

5

©Copyright 2013 Rogue Wave Software, Inc.

5

Advanced Topics



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

6

©Copyright 2013 Rogue Wave Software, Inc.

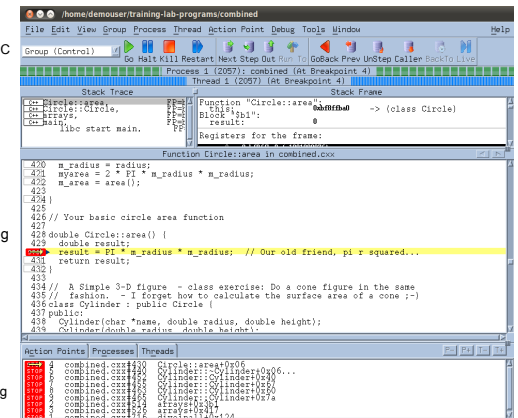
6

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



©Copyright 2013 Rogue Wave Software, Inc.

7

©Copyright 2013 Rogue Wave Software, Inc.

8

INTRODUCTION

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, MVA PICH1 & 2, Bullx MPI, & many others
 - UPC

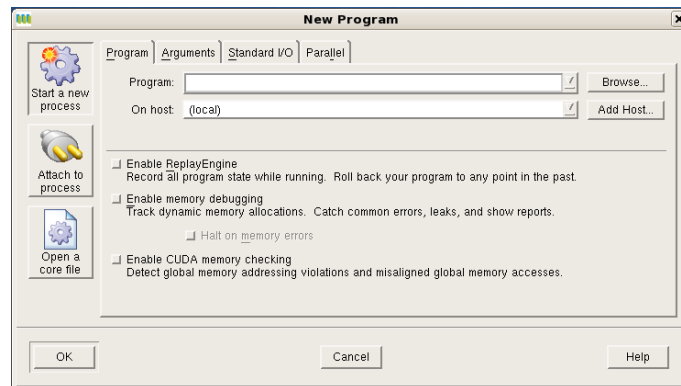


STARTUP

Starting TotalView



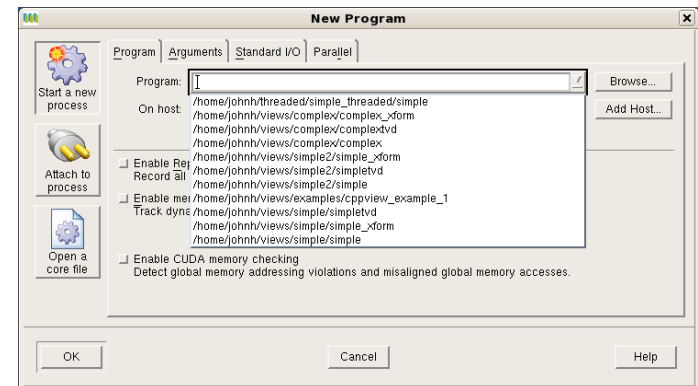
Start New Process



Starting TotalView



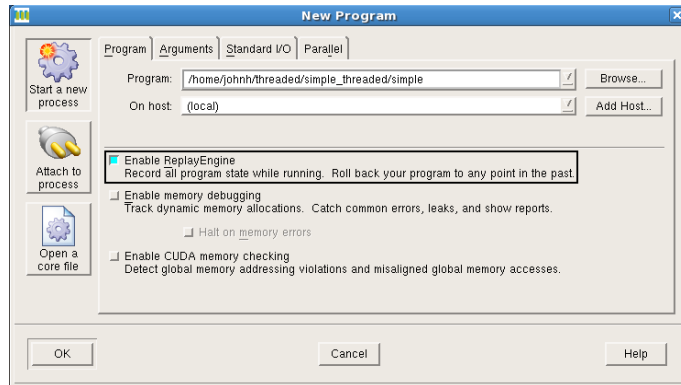
Start New Process – Select a recent process



Starting TotalView



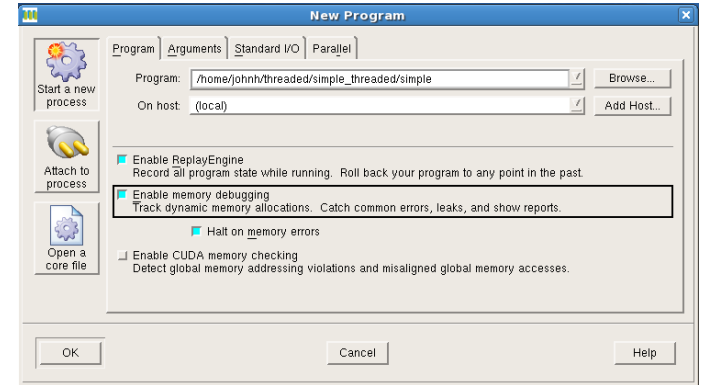
Start New Process – Enable ReplayEngine



Starting TotalView



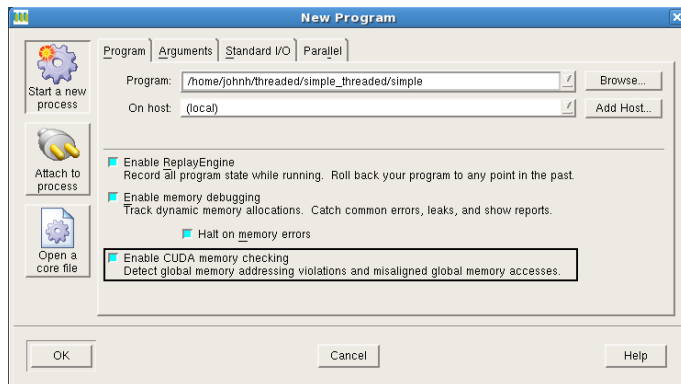
Start New Process – Memory Debugging



Starting TotalView



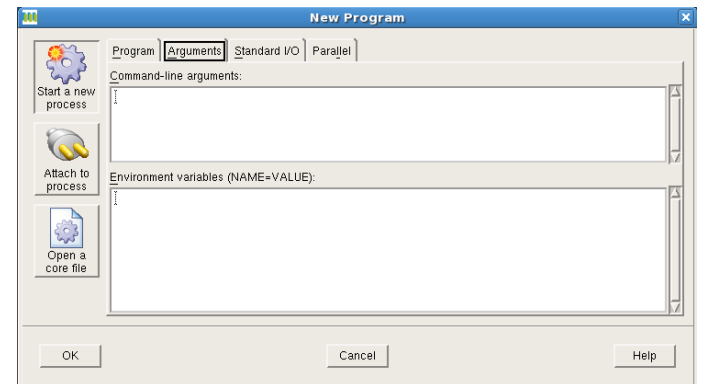
Start New Process – CUDA memory checking



Starting TotalView



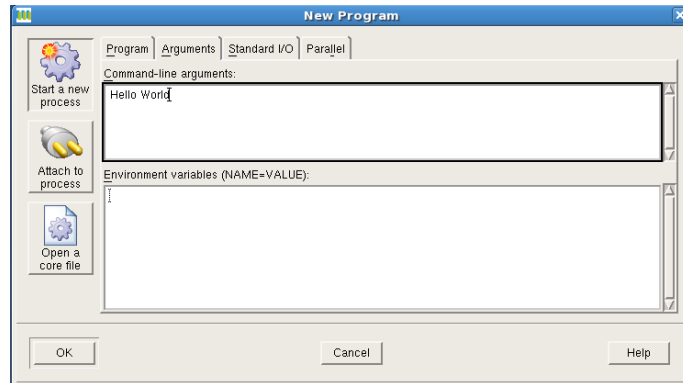
Start New Process – Arguments tab



Starting TotalView



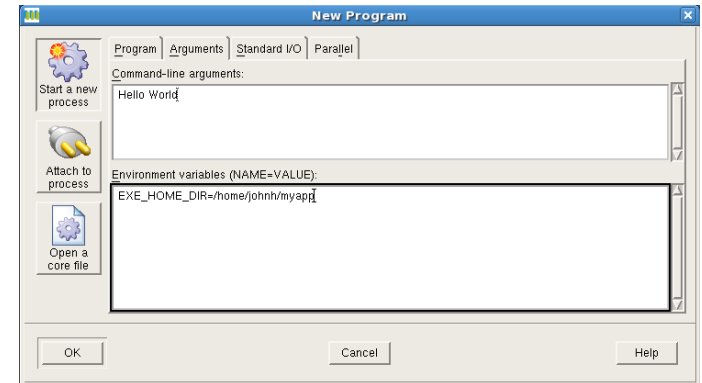
Start New Process – Command-line Args



Starting TotalView



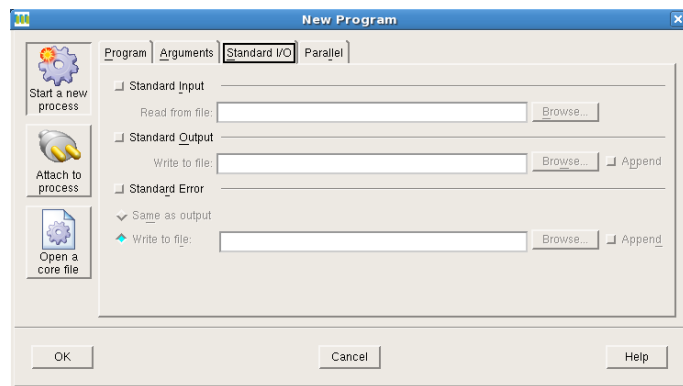
Start New Process – set environment variables



Starting TotalView



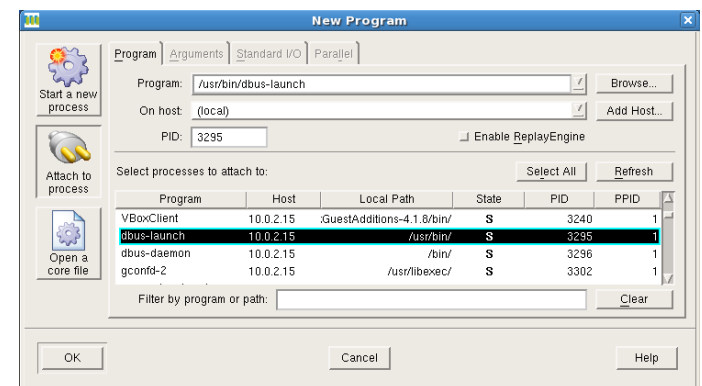
Start New Process – Standard I/O redirection



Starting TotalView



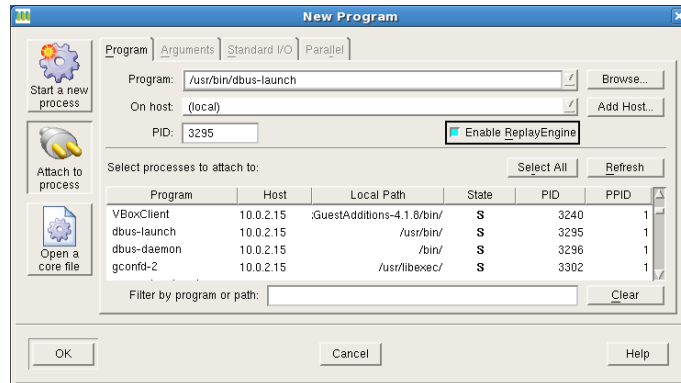
Attach to Process



Starting TotalView



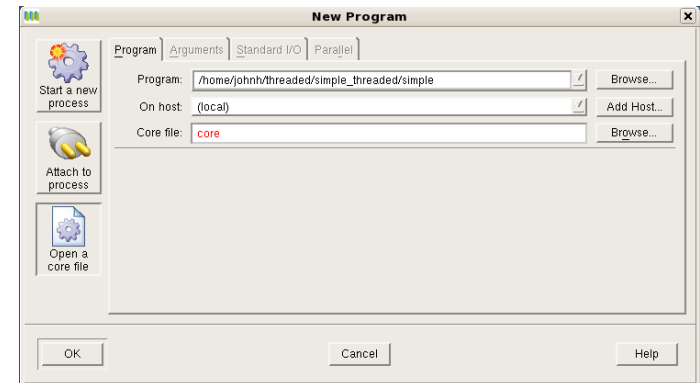
Attach to Process – Enable Replay Engine



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

ID/	Rank	Host	Status	Description
1	0	127.0.0.1	B	simplempi.0 (19 active threads)
3	4	127.0.0.1	B	simplempi.4 (12 active threads)
4	3	127.0.0.1	B	simplempi.3 (26 active threads)
5	8	127.0.0.1	B	simplempi.8 (9 active threads)
5.1	8	127.0.0.1	T	in __clone
5.2	8	127.0.0.1	B2 h	in runme
5.3	8	127.0.0.1	T	in runme
5.4	8	127.0.0.1	T	in runme
5.5	8	127.0.0.1	T	in runme
5.6	8	127.0.0.1	T	in __clone
5.7	8	127.0.0.1	B2 h	in runme
5.8	8	127.0.0.1	T	in __clone
5.9	8	127.0.0.1	T	in __clone
6	2	127.0.0.1	B	simplempi.2 (12 active threads)
7	5	127.0.0.1	B	simplempi.5 (24 active threads)
8	8	127.0.0.1	B	simplempi.6 (24 active threads)

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

TotalView Root Window



Host name

Hierarchical/Linear Toggle

Rank # (if MPI program)

TotalView Thread ID #

Expand - Collapse Toggle

Process Status

Action Point ID number

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

Process Window Overview



Stack Trace Pane

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

Toolbar

Stack Frame Pane

Source Pane

Tabbed Area

Stack Trace and Stack Frame Panes



Language

Name

Frame Pointer

Local Variables

Register Values

Click to refocus source pane

Click to modify

Dive for variable window

Source Code Pane



View as Source - or Assembly - or Both!

```

Function wait_a_while in simple.c
8 #include <mpi.h>          ::: 0x08048bb6: popl   %ebp
9 #endif //ADD_MPI         ::: 0x08048bb7: leal  -4(%ecx), %esp
10                          ::: 0x08048bb8:
11 void wait_a_while(size,  ::: 0x08048bb9:
12                          ::: 0x08048bba: ret
13 void need_to_wait()      ::: 0x08048bbb: nop
14 {                          18 wait_a_while(unsigned int): pushl %ebp
15     wait_a_while();        ::: 0x08048bbd: movl  %esp, %ebp
16                          ::: 0x08048bbe:
17                          ::: 0x08048bbf: subl  $8, %esp
18 void wait_a_while(size,  ::: 0x08048bc0:
19 {                          ::: 0x08048bc1:
20     usleep(microseconds);  ::: 0x08048bc2: movl  8(%ebp), %eax
21                          ::: 0x08048bc3:
22                          ::: 0x08048bc4:
23 void random_vector(std   ::: 0x08048bc5: movl  %eax, (%esp)
24 {                          ::: 0x08048bc6:
25     size_t count = (size  ::: 0x08048bc7:
26     if(count < 100) count ::: 0x08048bc8: call  0x0804880c
27                          ::: 0x08048bc9:
28     for(size_t i=0; i<co  ::: 0x08048bca:
29     {                       ::: 0x08048bcb:
30         vec.push back(rand  ::: 0x08048bcc:
    
```

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



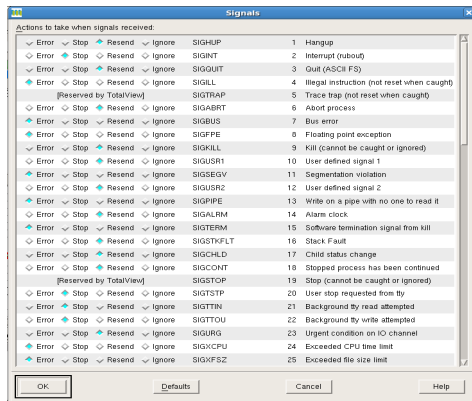
Process/Thread
status is available at
a glance, in both the
Process and Root
Windows

ID/	Rank	Host	Status	Description
4	9	127.0.0.1	B	simplempi.9 (26 active threads)
5	5	127.0.0.1	B	simplempi.5 (26 active threads)
6	8	127.0.0.1	B	simplempi.6 (26 active threads)
7	6	127.0.0.1	B	simplempi.7 (26 active threads)
8	7	127.0.0.1	B	simplempi.8 (26 active threads)
8.1	7	127.0.0.1	B3	in main
8.2	7	127.0.0.1	T	in __kernel_vsyscall
8.3	7	127.0.0.1	T	in __kernel_vsyscall
8.4	7	127.0.0.1	T	in __kernel_vsyscall

Search Paths

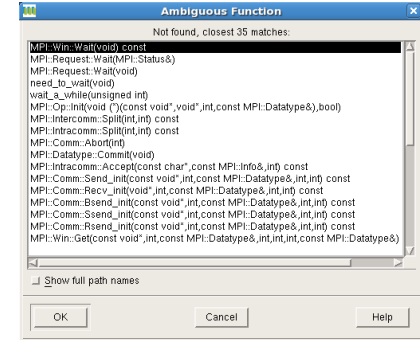
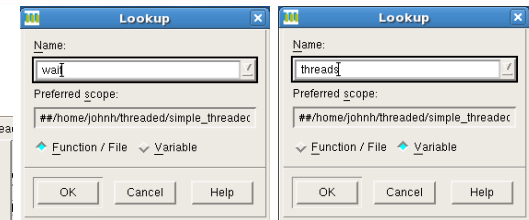
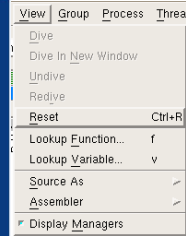


Managing Signals File > Signals

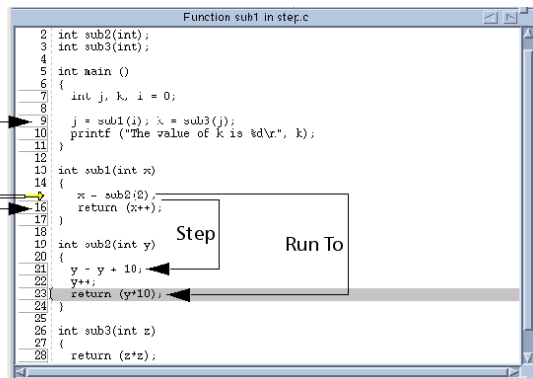
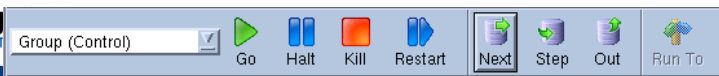


- Error** Stop the process and flag as error
- Stop** Stop the process
- Resend** Pass the signal to the target and do nothing; use with signal handlers
- Ignore** Discard the signal

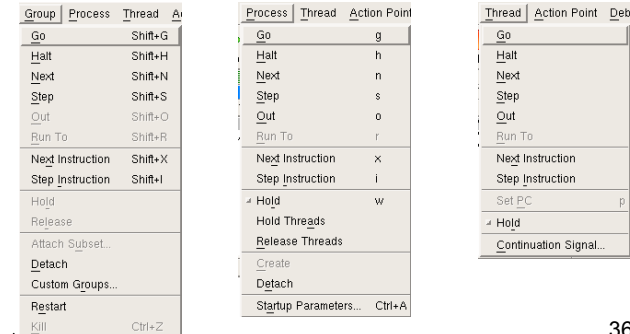
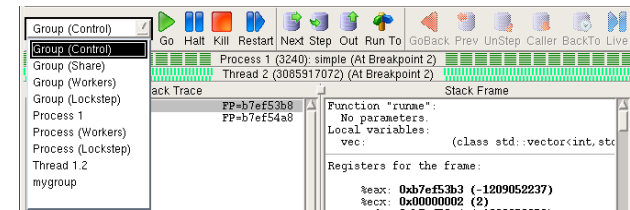
Finding Functions, Variables, and Source Files



Stepping Commands



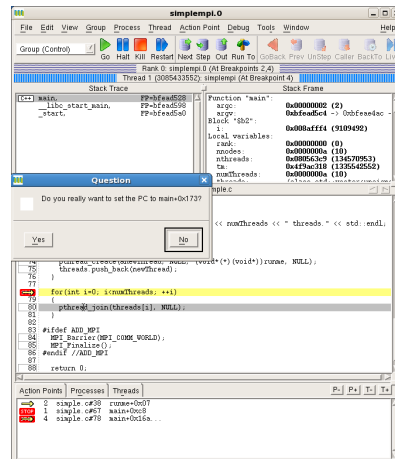
Stepping Commands



Using Set PC to resume execution at an arbitrary point



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



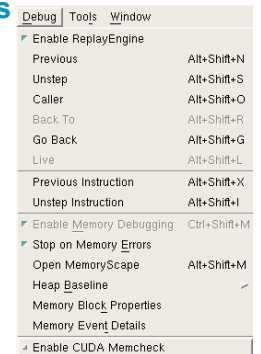
37

Debug Menu



• Menu Items for extra features

- Replay Engine
- MemoryScope
- CUDA Memcheck
- More on these features later



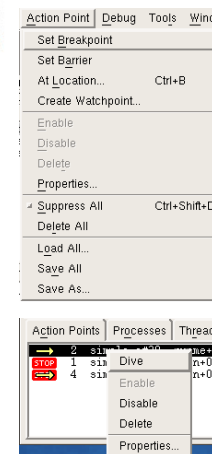
38

ACTION POINTS



39

Action Points



Breakpoints

Barrier Points

**Conditional
Breakpoints**

Evaluation Points

Watchpoints

40

Setting Breakpoints

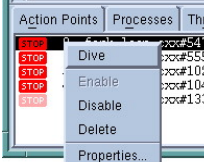


```

Function
1031 void fork_wrapper (i
1032 {
1033     pthread_t my_ptid;
1034     pthread_t new_tid;
1035     pthread_attr_t att
1036     int whoops;
1037     int local_fork_cou
1038     thread_ptids[0] =
1039
1040     if (!fork_late)
1041         forker (fork_cou
1042
1043     local_fork_count =
1044     printf ("Pid %d: s
1045
1046     printf ("root_ptid
1047     new_tid = 0;
1048     #if !defined (_Lynx
1049     pthread_attr_init
1050

```

- 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

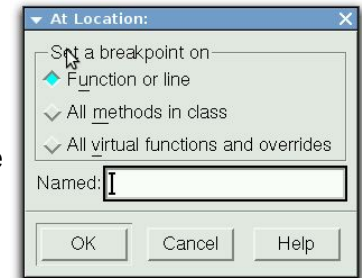


Setting Breakpoints

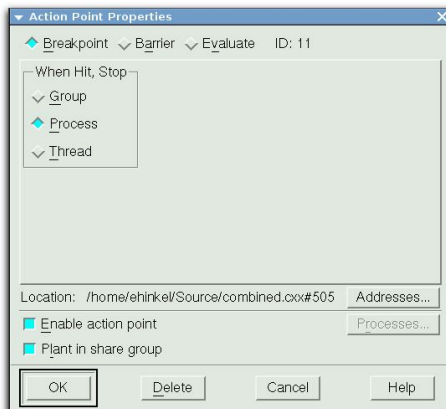


• Breakpoint->At Location...

- Specify function name or line #
- Specify class name and break on all methods in class, optionally with virtuals and overrides

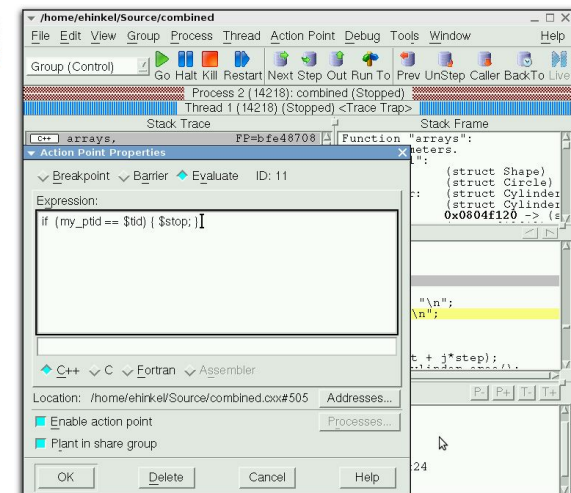


Setting Breakpoints



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

Conditional Breakpoint

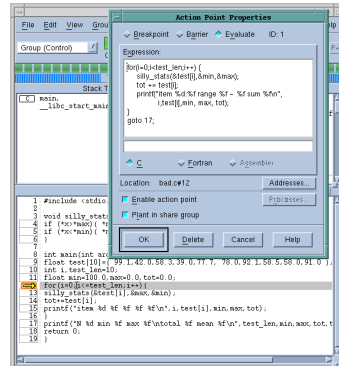


Evaluation Breakpoint... Test Fixes on the Fly!



- 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

item	0x90,099998	range	95,099998	-	95,099998	sum	95,099998
item	1142,000000	range	42,000000	-	95,099998	sum	141,100000
item	2150,000000	range	42,000000	-	95,099998	sum	150,400000
item	3139,000000	range	35,000000	-	95,099998	sum	228,400000
item	4172,000000	range	35,000000	-	95,099998	sum	316,100000
item	5178,000000	range	35,000000	-	95,099998	sum	384,100000
item	6162,000000	range	35,000000	-	95,099998	sum	465,200000
item	7158,000000	range	35,000000	-	95,099998	sum	544,700000
item	8156,000000	range	35,000000	-	95,099998	sum	632,700000
item	9151,000000	range	35,000000	-	95,099998	sum	685,700000
N 10 min	95,000000	max	95,099998				
total	455,700000	mean	95,370000				

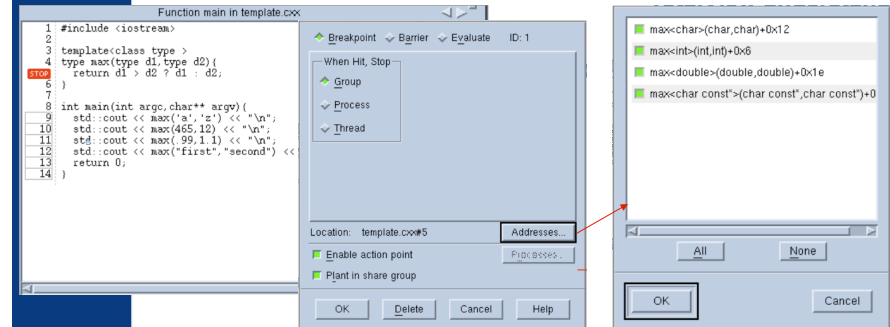


45

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.

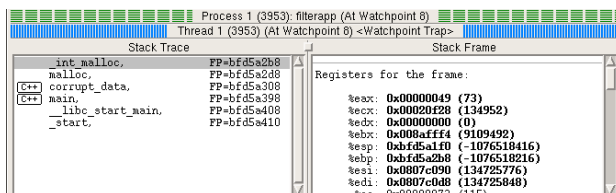
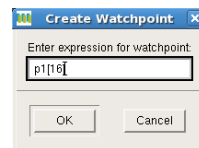
46

Watchpoints



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

Action Point ->
Create
Watchpoint...

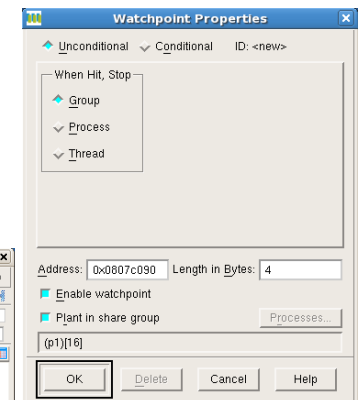
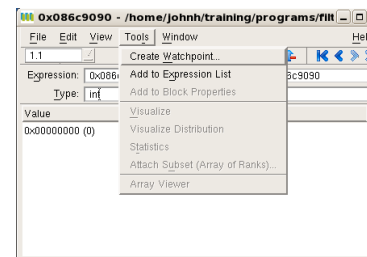


47

Watchpoints



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

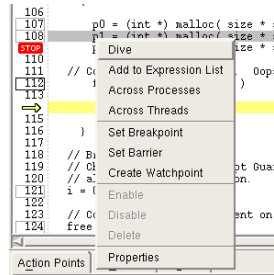


48

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

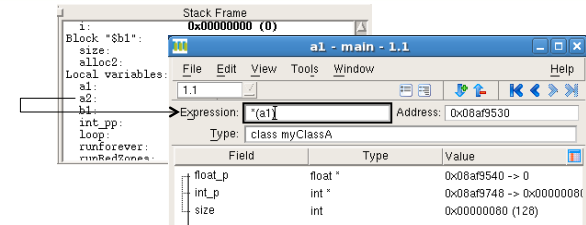
You can Dive on:

- ... variable names to open a variable window
- ... function names to open the source in the Process Window.
- ... processes and threads in the Root Window.

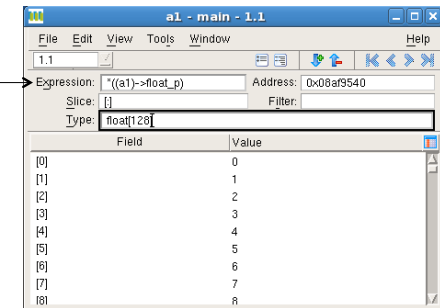
How do I dive?

- 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

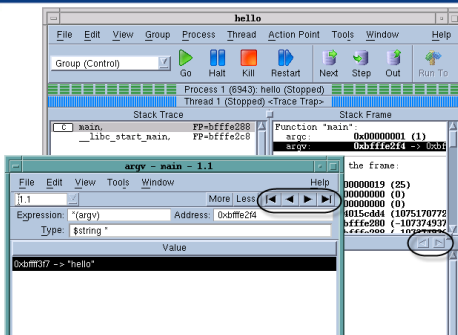
Diving



Diving on a Common Block in the Stack Frame Pane



Undiving

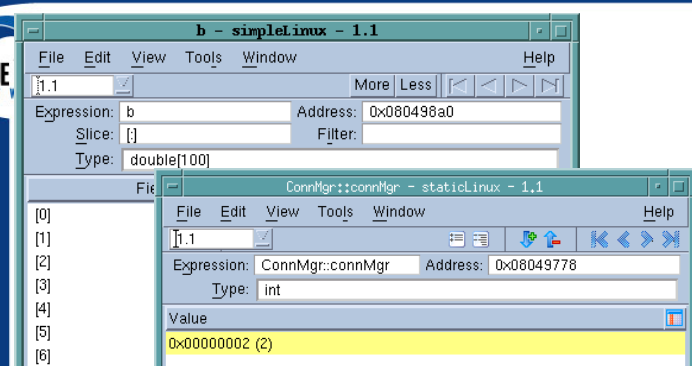


In a Process Window: retrace the path that has been explored with multiple dives.

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 mode
- Esc key cancels editing
- Enter key commits a change
- Editing values changes the memory of the program

Expression List Window



Expression	Value
rank	0x00000000 (0)
nnodes	0x0000000a (10)
numThreads	0x0000000a (10)
tm	0x4f9ac318 (1335542552)
tm/numThreads	0x075e04f (133554255)
tm/3600/24/365.25	42.3189596167009

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

Field	Value
(1,1,1)	0
(2,1,1)	-0.506366
(3,1,1)	-0.873297
(4,1,1)	-0.999756
(5,1,1)	
(6,1,1)	
(7,1,1)	
(8,1,1)	
(9,1,1)	
(10,1,1)	

Structure Arrays

Field	Type	Value
[0]	class d1	(Class)
base	class base	(Virtual public base class)
b_v	int	0x00000000 (0)
bb_v	int	0x00000000 (0)
name	\$string "	0x00048ad9 -> "base"
base2	class base2	(Virtual public base class)
b2_v	int	0x00000000 (0)
bb2_v	int	0x00000000 (0)
name	\$string "	0x00048ade -> "base2"
d1_v	int	0x00000000 (0)

Structure Arrays

Array Viewer



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

Array Viewer: *((a1)->float_p)[i][j]

Expressions: *((a1)->float_p) Type: float[8][16]

Modify array slice:

Dimension	Start Index	End Index	Stride
Row [i]	0	7	1
Column [j]	0	15	1

Format: Automatic Slice: [0:7][0:15:1]

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

Slicing Arrays



two_d_array - arraysLinux - 2.1

Expression: two_d_array Address: 0x08097dc0

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

Type: INTEGER*4(10,10)

Field	Value
(6,6)	216 (0x000000d8)
(7,6)	294 (0x00000126)
(8,6)	364 (0x00000180)
(9,6)	466 (0x000001e6)
(10,6)	600 (0x00000258)
(6,7)	252 (0x000000fc)
(7,7)	343 (0x00000157)
(8,7)	448 (0x000001c0)
(9,7)	567 (0x00000237)
(10,7)	700 (0x000002bc)

Slice notation is [start:end:stride]

Filtering Arrays

The first screenshot shows a window titled 'int2_array - MAIN - 1.1' with the expression 'int2_array' and address '0x0004450'. The filter is set to 'is int'. The second screenshot shows the same window with the filter changed to '\$value > 30 and \$value < 100'. The third screenshot shows the resulting filtered array elements in a table:

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

Visualizing Arrays

The 3D plot shows a surface with a color gradient from blue to red. The 2D graph shows a parabolic curve with red data points. The Y-axis ranges from -1.00 to 0.00, and the X-axis ranges from 0.00 to 40.0.

- 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

The screenshot shows a structure 'struct compound_t' with fields 'a', 'b', and 'c'. The 'int' field is selected, and a context menu is open with 'Dive in All' selected. The text 'Dive in All will display an element in an array of structures as if it were a simple array.' is overlaid on the image.

Looking at Variables across Processes

The screenshot shows a window titled 'source - main - 1.1' with the expression 'source' and address 'Multiple'. The filter is set to 'int'. The resulting table shows the values for 'mismatchAlpha.0' through 'mismatchAlpha.3':

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

- 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



Field	Type	Value
[0]	struct compound_t	(Struct)
x	struct basic_t	(Struct)
y	struct basic_t *	0x0804abd6 -> (struct bas...
z	float	5
[1]	struct compound_t	(Struct)
x	struct basic_t	(Struct)
y	struct basic_t *	0x080490dd -> (struct bas...
z	float	3.99868e-34
[2]	struct compound_t	(Struct)

C++ Class Hierarchies



Variable Window shows class hierarchy using indentation

Field	Type	Value
derived1	class derived1	(Public base class)
base1	class base1	(Virtual public base class)
base1_v	int	0x00000009 (9)
name	\$string *	0x08048808 -> "base1"
derived1_v	int	0x00000051 (81)
name	\$string *	0x0804880e -> "derived1"
base1	class base1	(Virtual public base class)
base1_v	int	0x00000009 (9)
name	\$string *	0x08048808 -> "base1"
derived2_v	int	0x000002d9 (729)
name	\$string *	0x08048817 -> "derived2"

- 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



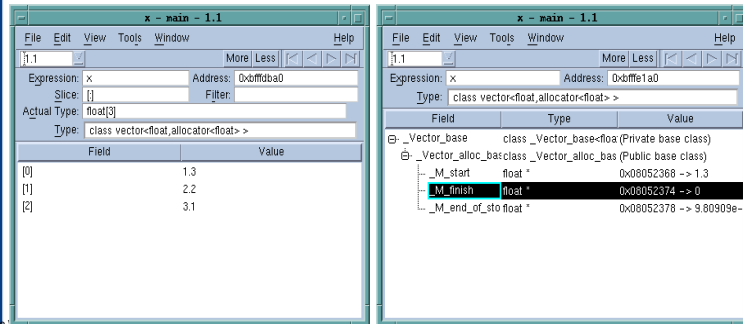
Variables	Type	Value
.. a1	REAL*8(4)	(REAL*8(4))
.. v1	INTEGER*4	0 (0x00000000)
.. v2	INTEGER*4	0 (0x00000000)

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



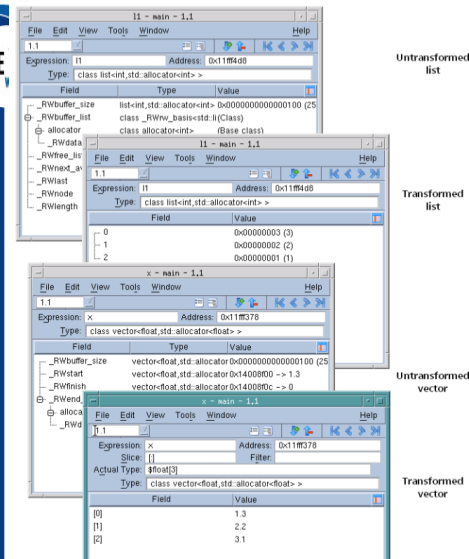
©Copyright 2013 Rogue

©

STLView



STLView transforms templates into readable and understandable information



©Copyright 2013 Rogue

70



LAB 2: VIEWING, EXAMING, WATCHING AND EDITING DATA

©Copyright 2013 Rogue Wave Software, Inc.

71



DEBUGGING FOR PARALLEL APPLICATIONS

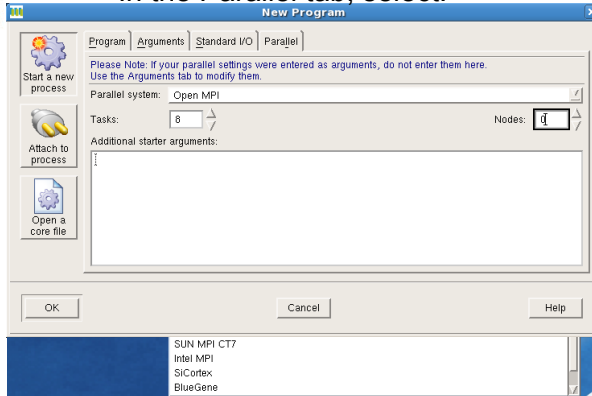
©Copyright 2013 Rogue Wave Software, Inc.

72

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



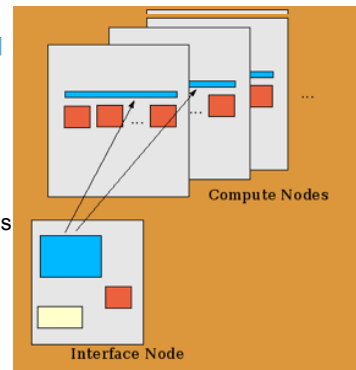
IBM	<code>totalview poe -a myprog -procs 4 -rmpool 0</code>
QUADRICS Intel Linux under SLURM	<code>totalview srun -a -n 16 -p pdebug myprog</code>
MVAPICH Opteron Linux under SLURM	<code>totalview srun -a -n 16 -p pdebug myprog</code>
SGI	<code>totalview mpirun -a myprog -np 16</code>
Sun	<code>totalview mprun -a myprog -np 16</code>
MPICH	<code>mpirun -np 16 -tv myprog</code>
MPICH2 Intel MPI	<code>Totalview python -a 'which mpiexec' -tvsvu -np 16 myprog</code>

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



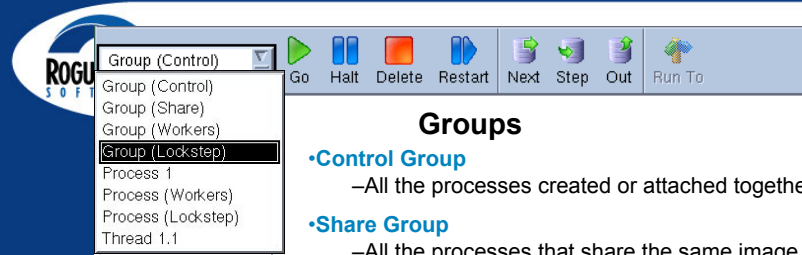
Provides Robust, Scalable and efficient operation with Minimal Program Impact

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

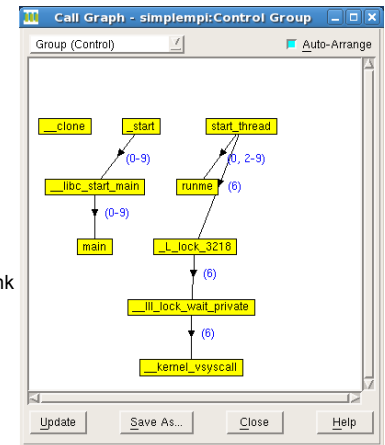
Call Graph



Quick view of program state

- Each call stack is a path
- Functions are nodes
- Calls are edges
 - Labeled with the MPI rank
- Construct process groups

Look for outliers



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

Parallel Back Trace

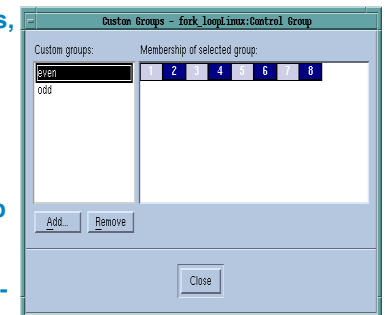
Processes	Location	PC	Host	Rank	ID	Status
1	_clone	0x0082c096	127.0.0.1	6	4.11 - 2919971728	Stopped
1	_clone	0x0082c096	127.0.0.1	6	4.9 - 2940951440	Stopped
10	_start	0x08051111
↳-10	_libc_start_main	0x00770e9c
↳-10	main	0x080513a0
	main#78	0x080513a0	127.0.0.1	0	1.1 - 3085433552	Breakpoint
	main#78	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
67	start_thread	0x00902852
↳-1	_L_lock_3218	0x0090381a
↳-1	_ll_lock_wait_private	0x00909743
↳-1	kernel_vsycall	0xb711e402	127.0.0.1	6	4.8 - 2951441296	Stopped
↳-66	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

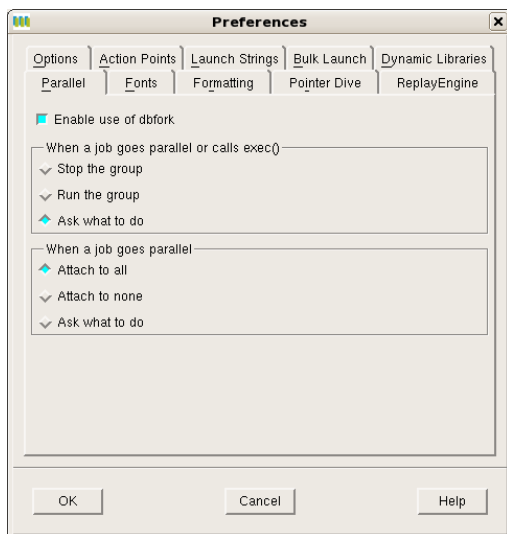


Group > Custom Groups, to create a process group of some other specification

- Group Membership shown in Processes Tab
- User defined groups appear in the “Go” drop-down menu



Preferences

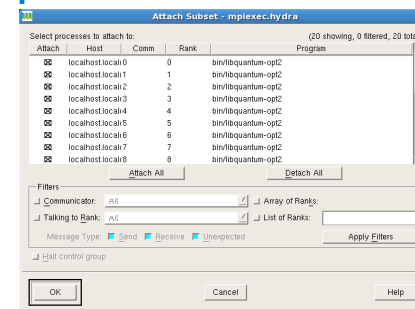


81

Subset Attach



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



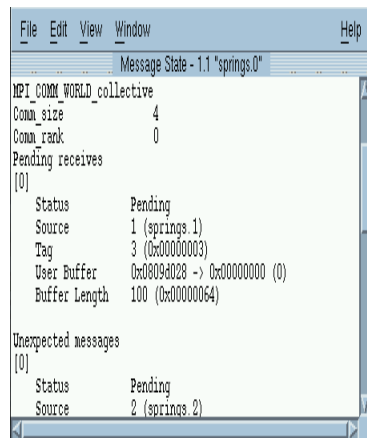
82

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

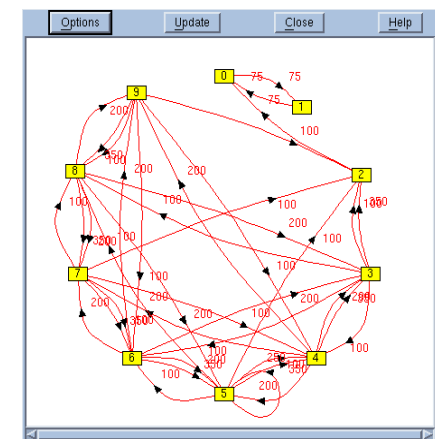


83

Message Queue Graph



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



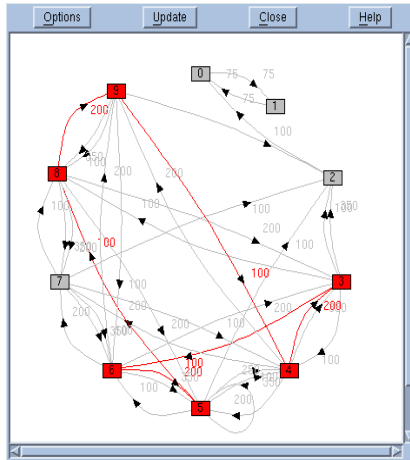
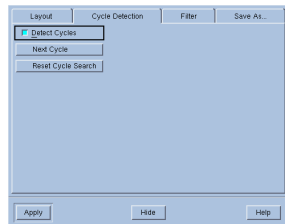
84

Message Queue Graph



Message Queue Debugging

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



85

©Copyright 2013 Rogue Wave Software, Inc.

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

86

©Copyright 2013 Rogue Wave Software, Inc.



LAB 3: EXAMINING AND CONTROLLING A PARALLEL APPLICATION

87

©Copyright 2013 Rogue Wave Software, Inc.



ADVANCED ASYNCHRONOUS CONTROL

88

©Copyright 2013 Rogue Wave Software, Inc.

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



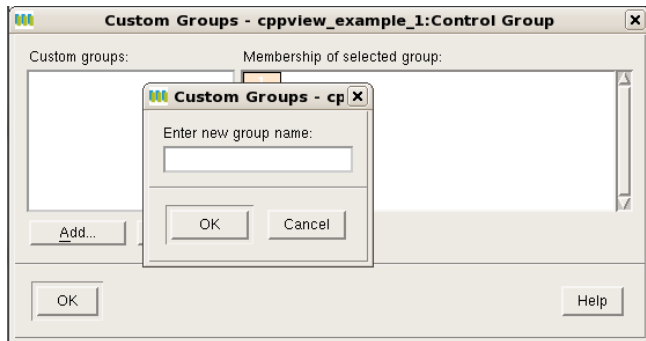
- Only the Workers group can be modified by the user
 - CLI, use dworker 0 to remove from the workers group or dworker 1 to add
- Create a Custom Group from the Group menu

Group	Process	Thread	A
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

Creating a Custom Group



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

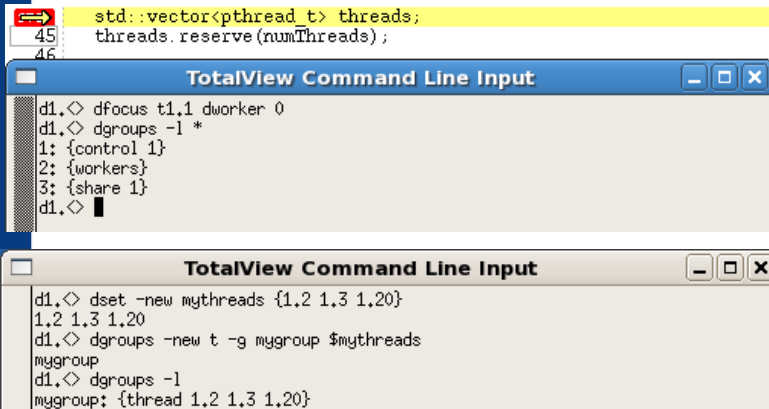


Custom Groups in the CLI

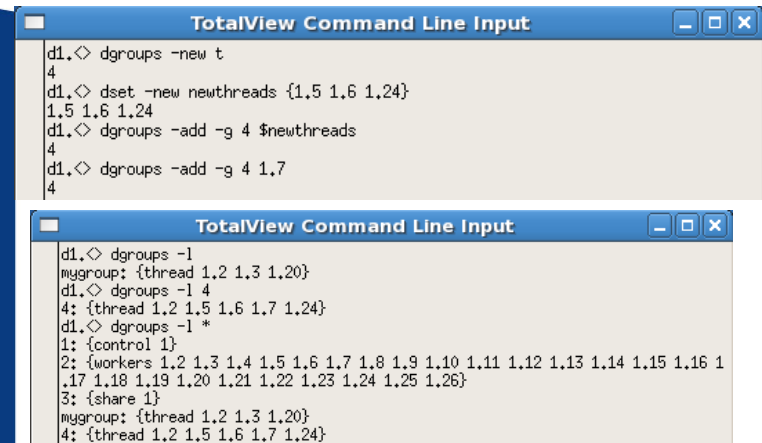


- In the CLI, use the **dgroups** command to create & modify groups
 - `dgroups -new t|p [-g groupname] [id_list]`
 - `dgroups -add [-g groupname] [id_list]`
 - `dgroups -remove [-g groupname] [id_list]`
 - `dgroups -intersect [-g groupname id_list]`
 - `dgroups -delete [-g groupname]`
 - t or p – can 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



```
TotalView Command Line Input
d1.> dgroups -i -g 4 $newthreads
d1.> dgroups -l 4
4: {thread 1,5 1,6 1,24}
d1.> dgroups -remove -g 4 1,6
d1.> dgroups -l 4
4: {thread 1,5 1,24}
d1.> dgroups -delete -g 4
d1.> dgroups -l *
1: {control 1}
2: {workers 1,2 1,3 1,4 1,5 1,6 1,7 1,8 1,9 1,10 1,11 1,12 1,13 1,14 1,15 1,16 1,17 1,18 1,19 1,20 1,21 1,22 1,23 1,24 1,25 1,26}
3: {share 1}
mygroup: {thread 1,2 1,3 1,20}
```

Breakpoints



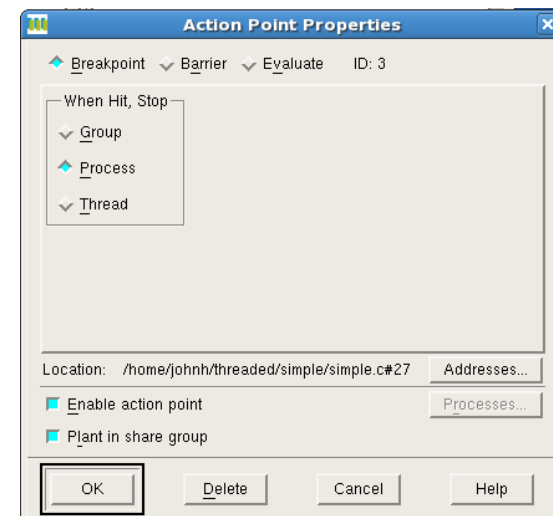
- **Control where they are planted, defaults to the Share Group**
 - Uses the SHARE_ACTION_POINT variable, true plants in the Share Group, false plants in the focus process only
- **Control what is stopped by hitting the breakpoint, the group, the process, or just the thread**
 - Uses the STOP_ALL variable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

Breakpoints

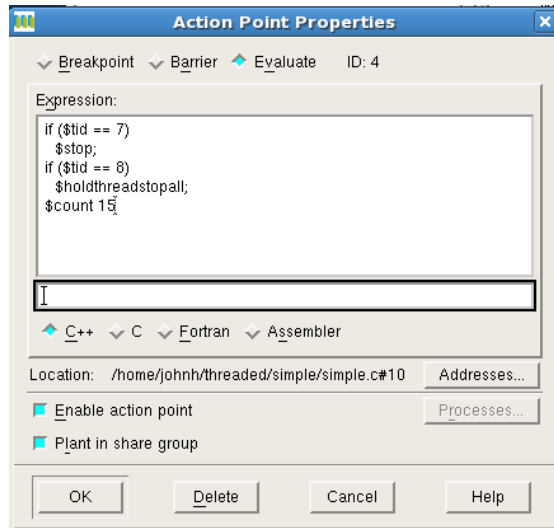


- **Control what is stopped and finer control over when it is stopped by using eval option and writing test code**
 - Code can be C, C++, FORTRAN 77, Fortran 9x, or assembler
 - Can use TotalView-specific values and commands like \$tid, \$pid, \$stop
 - Use -lang and -e flags to dbreak in CLI

Breakpoints in UI



Eval Breakpoints in UI



101

©Copyright 2013 Rogue Wave Software, Inc.

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

102

©Copyright 2013 Rogue Wave Software, Inc.

Barriers

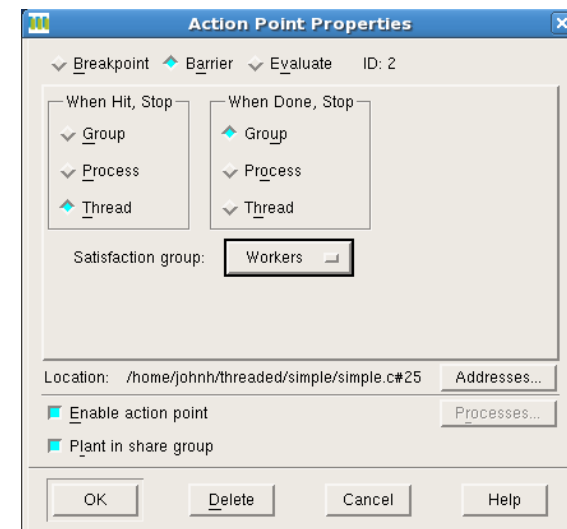


- **Control what is stopped when the barrier is satisfied, the group or the process**
 - Uses the BARRIER_STOP_WHEN_DONE variable set to: group, process, or none (same as process for a process barrier)
 - Use `-stop_when_done` flag in CLI to override default

103

©Copyright 2013 Rogue Wave Software, Inc.

Barriers Satisfaction Group in UI



104

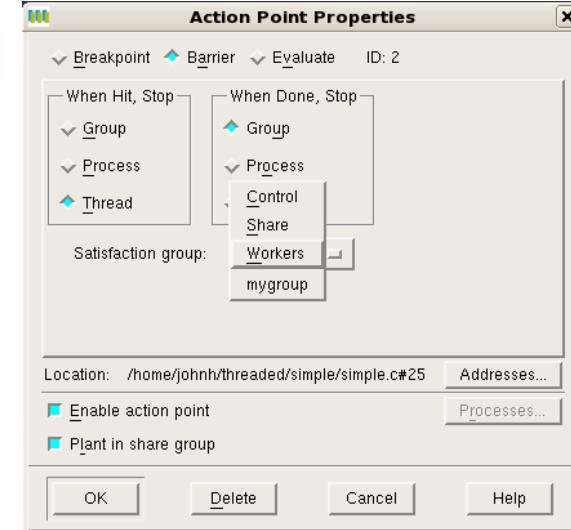
©Copyright 2013 Rogue Wave Software, Inc.

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.

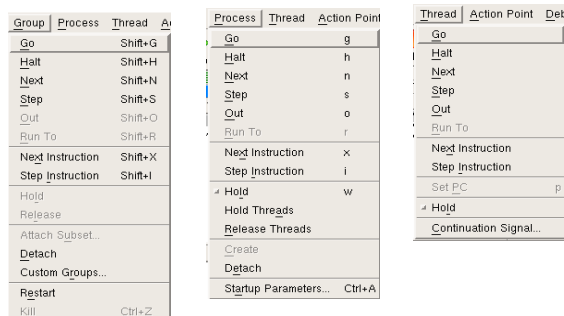
Barriers – Select Satisfaction Group UI



Asynchronous Controls



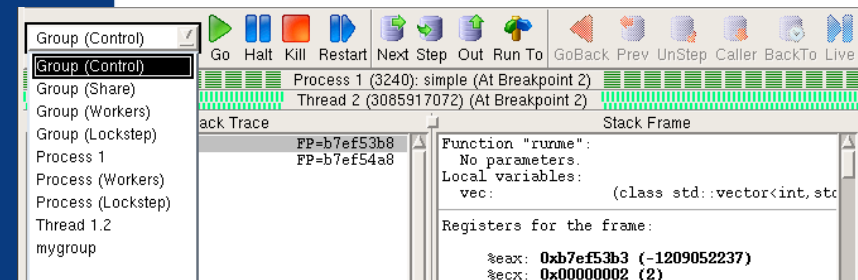
- **Once things are stopped, now what?**
- **CLI commands operate on the current focus, so you can step, next, go, etc. based on your focus of a group, process, or thread**
- **UI has separate menus for Group, Process, and Thread control**



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



Rank 2: simplempi.2 (At Breakpoint 3) [Held]
 Thread 2 (3083479952): simplempi (At Breakpoint 3)

Process	Thread	Action Point
<u>G</u> o		g
<u>H</u> alt		h
<u>N</u> ext		n
<u>S</u> tep		s
<u>O</u> ut		o
<u>R</u> un To		r
Next <u>I</u> nstruction		x
Step <u>I</u> nstruction		i
<input checked="" type="checkbox"/> <u>H</u> old		w
Hold <u>T</u> hreads		
Release <u>T</u> hreads		
<u>C</u> reate		
<u>D</u> etach		
Startup Parameters...	Ctrl+A	

Holds - CLI



```
TotalView Command Line Input
g1.<> dstatus
1 (34580127.0.0.1) Mixed [simplempi.0]
1.1 (3458/30848683040127.0.0.1) Running PC=0xb7f7d402
1.2 (3458/30831481760127.0.0.1) h Breakpoint PC=0x08051489. [/home/johnh/t
hreaded/simple/simple.c#25]
1.3 (3458/30039602080127.0.0.1) h Breakpoint PC=0x08051489. [/home/johnh/t
hreaded/simple/simple.c#28]
1.4 (3458/29934703520127.0.0.1) h Breakpoint PC=0x08051489. [/home/johnh/t
hreaded/simple/simple.c#25]
1.5 (3458/29829804960127.0.0.1) h Breakpoint PC=0x08051489. [/home/johnh/t
hreaded/simple/simple.c#28]
3 (34590127.0.0.1) Mixed [simplempi.1]
3.1 (3459/30852615200127.0.0.1) Running PC=0xb7fdd402
3.2 (3459/30835413920127.0.0.1) h Breakpoint PC=0x08051489. [/home/johnh/t
hreaded/simple/simple.c#28]
3.3 (3459/30043534240127.0.0.1) h Breakpoint PC=0x08051489. [/home/johnh/t
hreaded/simple/simple.c#25]
3.4 (3459/29938635680127.0.0.1) h Breakpoint PC=0x08051489. [/home/johnh/t
hreaded/simple/simple.c#28]
3.5 (3459/29833737120127.0.0.1) h Breakpoint PC=0x08051489. [/home/johnh/t
hreaded/simple/simple.c#25]
4 (34600127.0.0.1) Mixed [simplempi.3]
4.1 (3460/30844054560127.0.0.1) Running PC=0xb7f0c402
4.2 (3460/30826893280127.0.0.1) h Breakpoint PC=0x08051489. [/home/johnh/t
hreaded/simple/simple.c#25]
4.3 (3460/30034973500127.0.0.1) h Breakpoint PC=0x08051489. [/home/johnh/t
hreaded/simple/simple.c#28]
4.4 (3460/29930075040127.0.0.1) h Breakpoint PC=0x08051489. [/home/johnh/t
hreaded/simple/simple.c#25]
4.5 (3460/29825176480127.0.0.1) h Breakpoint PC=0x08051489. [/home/johnh/t
hreaded/simple/simple.c#28]
5 (34610127.0.0.1) H Breakpoint [simplempi.2]
5.1 (3461/30832000000127.0.0.1) Stopped PC=0xb7fce402
5.2 (3461/30834799520127.0.0.1) Breakpoint PC=0x08051461. [/home/johnh/t
hreaded/simple/simple.c#25]
5.3 (3461/30042919840127.0.0.1) Breakpoint PC=0x08051461. [/home/johnh/t
hreaded/simple/simple.c#25]
5.4 (3461/29938021280127.0.0.1) Breakpoint PC=0x08051461. [/home/johnh/t
hreaded/simple/simple.c#25]
5.5 (3461/2983312720127.0.0.1) Breakpoint PC=0x08051461. [/home/johnh/t
hreaded/simple/simple.c#25]
```

Holds



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

Holds - releasing



Rank 2: simplempi.2 (Mixed)
Thread 2 (3083479952): simplempi (At Breakpoint 5) [h]

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

©Copyright 2013 Rogue Wave Software, Inc.

113



LAB 9: ASYNCHRONOUS CONTROL

©Copyright 2013 Rogue Wave Software, Inc.

114

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

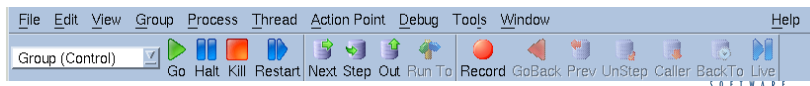
Specifications

- 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

```

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;

```



115 | Copyright © 2013 Rogue Wave Software | All Rights Reserved

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

116 | Copyright © 2013 Rogue Wave Software | All Rights Reserved



ReplayEngine controls

Replay Engine – The right way to debug

	Step forward over functions		Step <i>backward</i> over functions
	Step forward into functions		Step <i>backward</i> into functions
	Advance forward out of current Function, after the call		Advance backward out of current Function, to before the call
	Advance forward to selected line		Advance backward to selected line
	Run forward		Run <i>backward</i>
	Advance forward to "live" session		

117 | Copyright © 2013 Rogue Wave Software | All Rights Reserved



Demo



118 | Copyright © 2013 Rogue Wave Software | All Rights Reserved

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

119 | Copyright © 2013 Rogue Wave Software | All Rights Reserved



120 | Copyright © 2013 Rogue Wave Software | All Rights Reserved



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

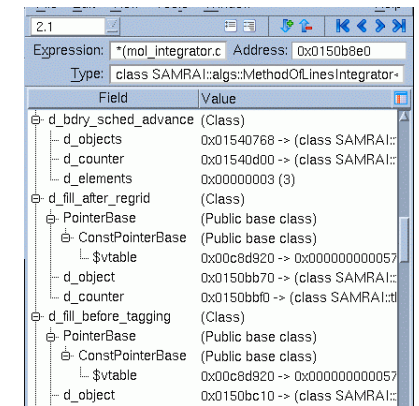
Alejandro Hernandez
UC Santa Barbara, LLNL Intern

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.)



Advanced Example – NaN/Inf/etc.

Field	Type	Value
delv	double[6350]	(Array)
e	double[6350]	(Array)
e_stat	\$string	"subNormal"
elmu	double[6350]	(Array)
elmu_stat	\$string	"subNormal"
eps	double[6350]	(Array)
eps_stat	\$string	"subNormal"

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.

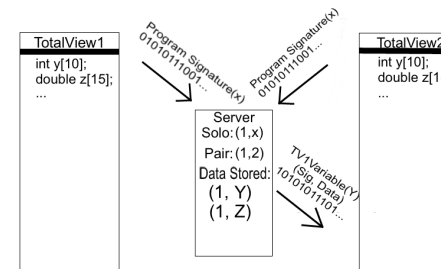


C++View Interface Provides Custom Debugging Support

- **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) ;`



Debugging Large-Scale OO Programs Can Be Simplified

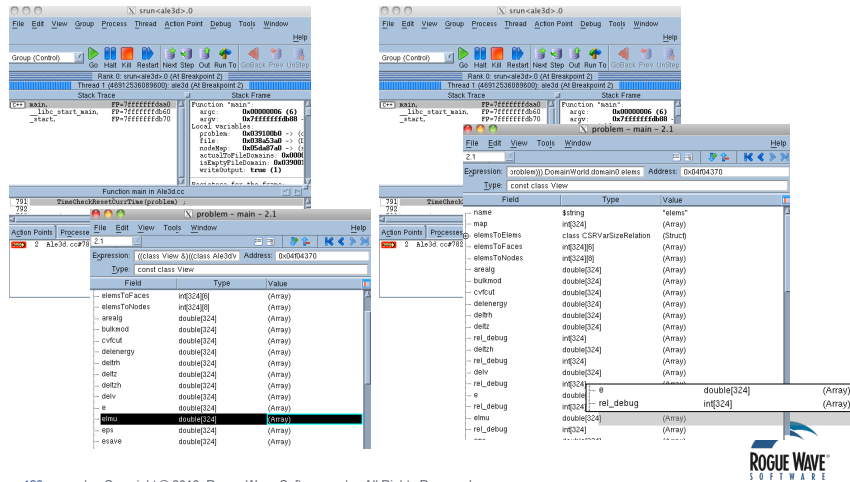


Semi-Automated Data Comparison Debugging

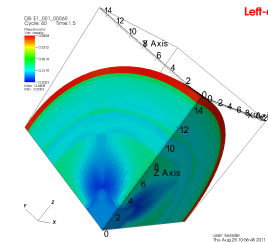
- **Semi-Automated Data Comparison Debugging:**
 - Compare two different versions of the same code to search for data differences
 - Data compared within the program
 - Comparison information displayed within the TotalView debugger



Differences Displayed as Integrated Part of Debugger



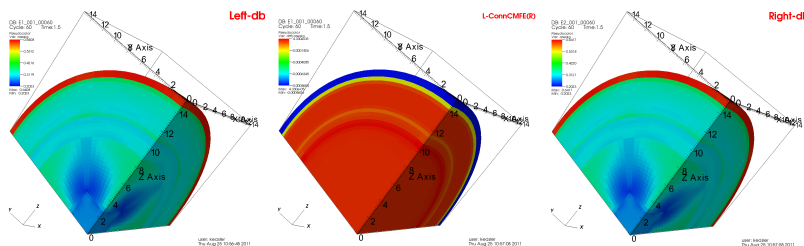
The Next Step is To Visualize Field Data...



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




...and Visualize Computed Differences between code Versions



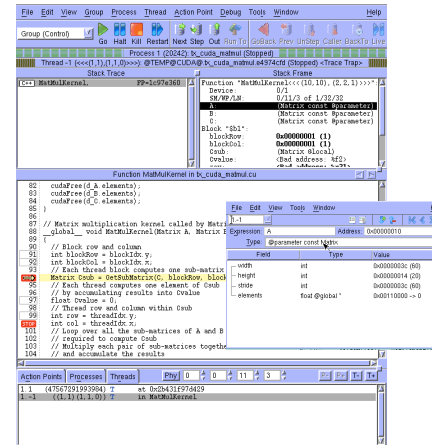
Semi-Automated Data-Comparison Debugging Provides an Additional Tool for Finding Bugs

- 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



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

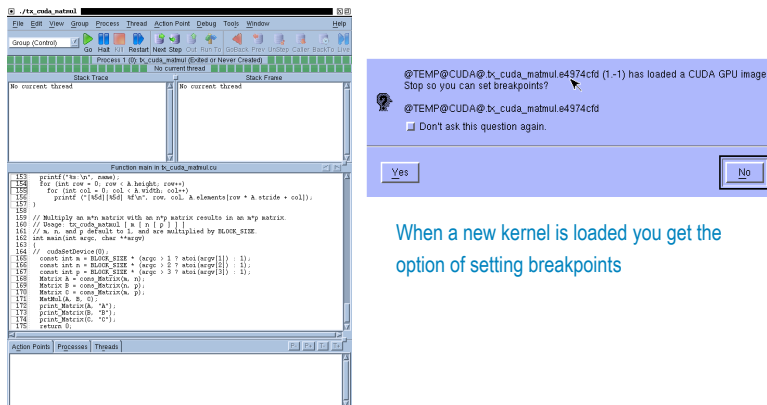
TotalView for CUDA



- 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

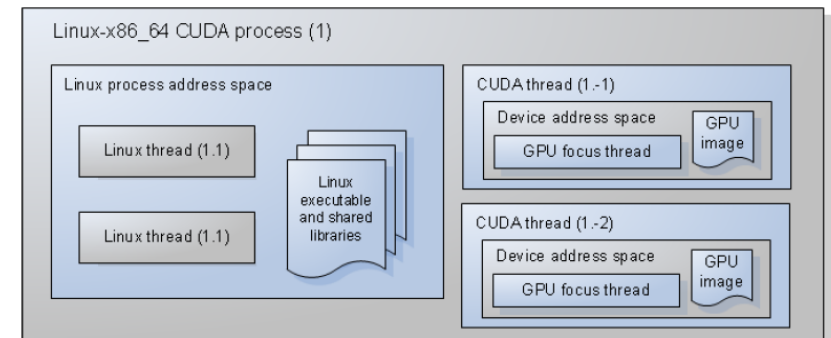


When a new kernel is loaded you get the option of setting breakpoints

- 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.

Action Points	Processes	Threads
1.1 (140657268762400) T		at 0x7fed51c04ae0
1.-1 ((8,0,0) (12,0,0)) F1		in saxpy_parallel

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?

Name	Description
Device 0/3	
Device Type	g1100
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)
Lane 10/32	Thread (10,0,0)
Lane 11/32	Thread (11,0,0)
Lane 12/32	Thread (12,0,0)
Lane 13/32	Thread (13,0,0)
Lane 14/32	Thread (14,0,0)
Lane 15/32	Thread (15,0,0)
Lane 16/32	Thread (16,0,0)
Lane 17/32	Thread (17,0,0)
Lane 18/32	Thread (18,0,0)
Lane 19/32	Thread (19,0,0)
Lane 20/32	Thread (20,0,0)
Lane 21/32	Thread (21,0,0)
Lane 22/32	Thread (22,0,0)
Lane 23/32	Thread (23,0,0)
Lane 24/32	Thread (24,0,0)
Lane 25/32	Thread (25,0,0)
Lane 26/32	Thread (26,0,0)
Lane 27/32	Thread (27,0,0)
Lane 28/32	Thread (28,0,0)
Lane 29/32	Thread (29,0,0)
Lane 30/32	Thread (30,0,0)
Lane 31/32	Thread (31,0,0)
SM Type	sm_20
SMs	14
Warps	48
Device 1/3	
Device Type	gt200
Lanes	32
SM Type	sm_13

Different PC within warp

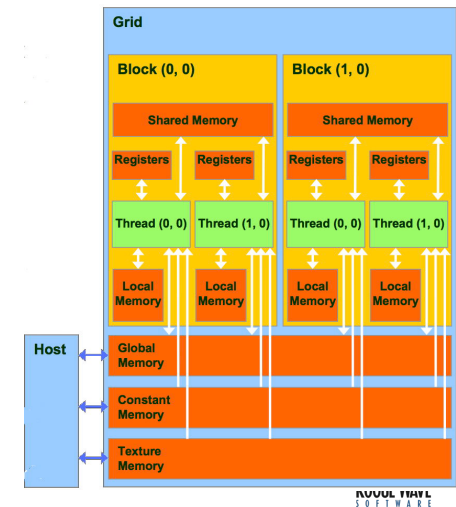
→ Diverging threads



GPU Memory Hierarchy

Hierarchical memory

- Local (thread)
 - Local
 - Register
- Shared (block)
 - Global (GPU)
 - Global
 - Constant
 - Texture
- System (host)
 - Texture Memory
 - Constant Memory
 - Global Memory



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.

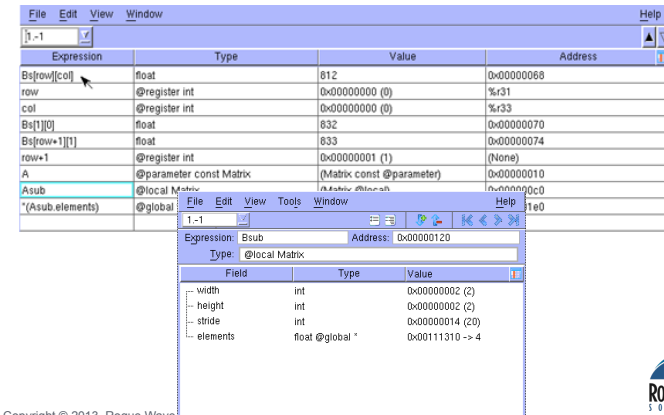
@global Address is an offset within global storage.

@register Address is a PTX register name.



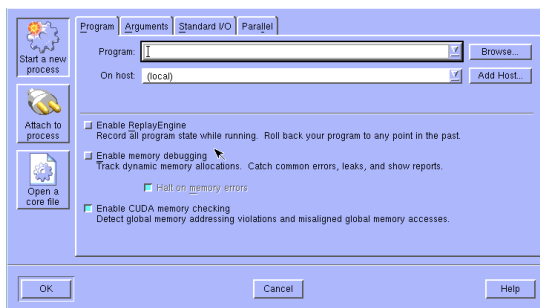
CUDA Variables

- Storage qualifiers appear in the data type



CUDA Segmentation Faults

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

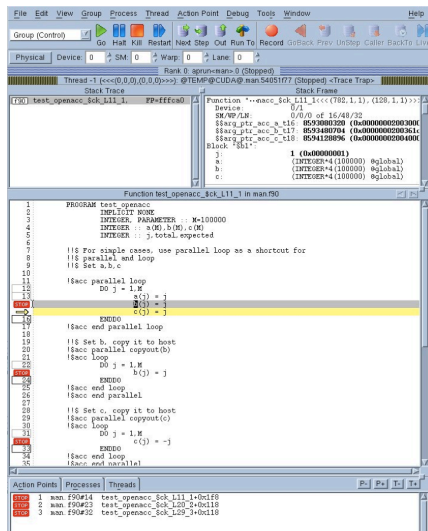


CUDA Built-in Runtime Variables

- 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



- 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**

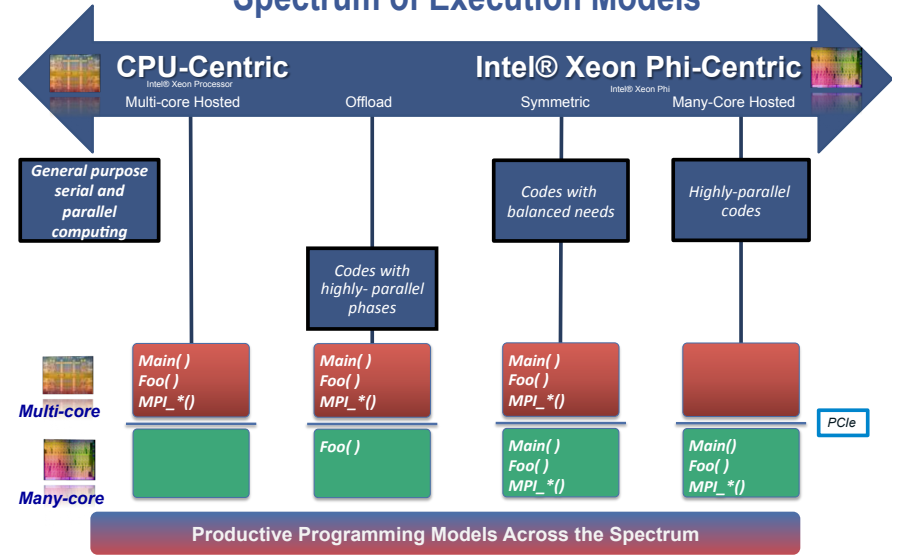


CUDA Debugging - Tips

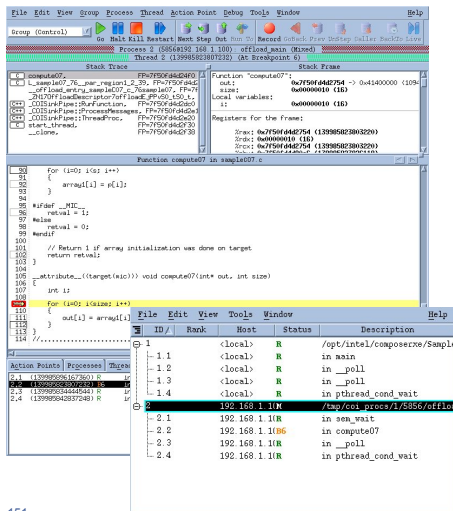
■ Check intermediate results

- 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)





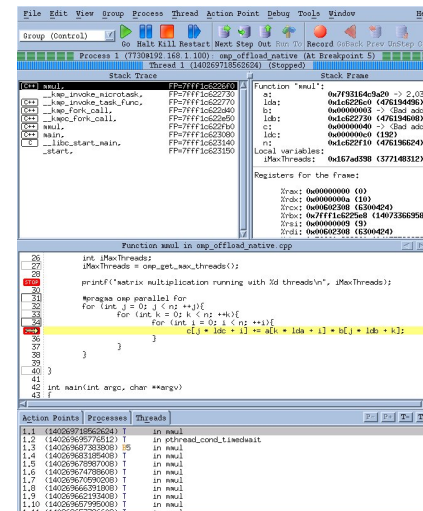
Xeon Phi Port of TotalView



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

The screenshot shows the TotalView 8.12 interface. At the top, a window titled 'Select processes to attach to' lists various processes on Host0 and Host1. The main window displays the source code of a program, with a 'Function main in tv_basic_mpi.c' highlighted. The 'Stack Trace' and 'Stack Frame' windows are visible, showing the current execution state. The 'Action Points' window at the bottom shows a list of breakpoints.

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



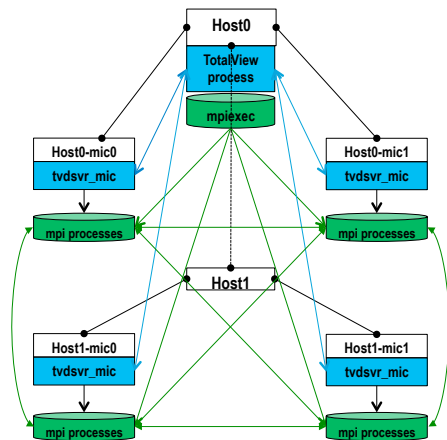
Debugging Applications with Offloaded Code

The screenshot shows two side-by-side TotalView 8.12 windows. The left window is titled 'Xeon side' and shows the source code of a program with a breakpoint set. The right window is titled 'Xeon Phi side' and shows the same source code with a different breakpoint set. The 'Stack Trace' and 'Stack Frame' windows are visible in both, showing the execution state on the respective hardware.

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 tvdsrvr 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_mpi`
 - Set: `dset TV::mic_native_server_launch_string { ssh -n %R "/bin/rm -f /tmp/tvdsrvrmain%K"; //1 scp %B/tvdsrvrmain%K %R:/tmp/tvdsrvrmain_mic; //2 ssh -n %R -n "/tmp/tvdsrvrmain%K -callback %L -set_pw %P -verbosity %V %F" //3 }`
1. Removes your previous tvdsrvrmain_mic
 2. Copies it from the installation directory to the /tmp/ directory on the coprocessor
 3. Starts the server on the Xeon Phi coprocessor.





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

Contact Support

If you are a current customer and require Technical Support or License Administration for the products you have already purchased, please contact us based on the product of interest.

- For **Technical Support**
- For **License Administration**
- For **Product Update or Download Requests**

For Technical Support

TotalView Family of Products

Workdays, except Japan

Engineers are available from 8:00 AM to 5:00 PM Eastern Time, Monday-Friday
US: 800-856-3766
Telephone: 508-852-7700
FAX: 508-852-3703
Email: tsupport@roguewave.com
Submit a Web Support Incident at: [File a Support Request](#)

IMSL Numerical Libraries

Workdays, except Japan

Engineers are available from 8:00 AM to 5:00 PM Central Time, Monday-Friday

TotalView Documentation

TotalView

Dynamic source code and memory debugging for C, C++ and Fortran applications

TotalView is a GUI-based source code defect analysis tool that gives you unprecedented control over processes and thread execution and visibility into program state and variables.

It allows you to debug one or many processes and/or threads with complete control over program execution, from basic debugging operations like stepping through code to sophisticated techniques that are becoming more commonplace in the high performance computing world. You can reproduce and troubleshoot difficult problems that can occur in concurrent programs that take advantage of threads, OpenMP, MPI, or computational accelerators.

TotalView provides analytical displays of the state of your running program for efficient debugging of memory errors and leaks and diagnosis of subtle problems like deadlocks and race conditions. TotalView works with C, C++ and Fortran applications written for Linux (including the Blue Gene platforms), UNIX and Mac OS X platforms. To learn more about TotalView, see the [Features](#) page, or take a look at the introductory video, [Getting Started with TotalView](#).

Request a free evaluation copy of TotalView to try it for yourself!

GET STARTED

- Evaluate
- Request a Demo
- Request For Quote (RFQ)
- Contact Us

OVERVIEW

- Features
- What's New

RESOURCES

- Brochures and Datasheets
- White Papers and Publications
- Videos
- Case Studies
- License Options

TECHNICAL INFORMATION

- Supported Platforms

SUPPORT

- Documentation
- Technical Support

GET STARTED

- Evaluate
- Request a Demo
- Request For Quote (RFQ)

TotalView Documentation

TotalView Family

- TotalView®
- MemoryScope
- PrologEngine

ThreadSpotter

- Version 2010.4 Manual

Legacy .h++ Products

Analysis.h++, DB Tools.h++, LAPACK.h++, Math.h++, MPI.h++, MPI-3.h++, Software PAM Migration, Standard C++ Library, ThinkAbi.h++, Tools.h++, Tools.h++ Professional

IMSL Numerical Libraries

- C Library
- Fortran Library
- Java™ Library
- JNET Library
- PyIMSL Studio

PV-WAVE Family

- PV-WAVE
- TS-WAVE
- J-WAVE™

Source Pro C++

- Version 11.1
- Version 11.0
- Version 10.1

Stingray

- Version 10
- Version 2006

HydraExpress

- Version 4.0.0
- Version 4.3.0
- Version 3.5.0

HostAccess

- Version 7.40
- Version 7.30

Thanks!

QUESTIONS?