



Debugging in Heterogeneous Environments with TotalView

ECMWF HPC Workshop
30th October 2014



Agenda

- Introduction
- Challenges
- TotalView overview
- Advanced features
- Current work and future plans

Introduction

Rogue Wave

What we do

Rogue Wave helps organizations simplify complex software development, improve code quality, and shorten cycle times

- History

- Founded: 1989
- Portfolio company of Audax Group
- Acquisitions:
 - Visual Numerics: 2009
 - TotalView Technologies: 2010
 - ILOG Visualization for C++ : 2012
 - OpenLogic : 2013
 - Klocwork : 2013
 - ILOG Visualization for Java: 2014

- Customers

- 3,000+ in 57 countries
- Financial services, telecoms, oil and gas, government and aerospace, research and academic

- Global Locations

- HQ: Boulder, CO
- NA: Houston, TX; Corvallis, OR; Natick MA
- EMEA: France, Germany, UK
- APAC: Japan

Challenges of developing for heterogeneous environments

Challenges

- Number of CPU cores increasing but clock speed is static or decreasing
- How to program accelerators
 - Lower level languages (OpenCL, CUDA)
 - Directives based (OpenACC, OpenMP)
- New algorithms or programming models may be needed
- Data sizes increasing exponentially
- Memory is increasingly important
- Power consumption and constraints

How does Rogue Wave help?

TotalView debugger

- Troubleshooting and analysis tool
 - Visibility into applications
 - Control over applications
- Scalability
- Usability
- Support for HPC platforms and languages

TotalView Overview

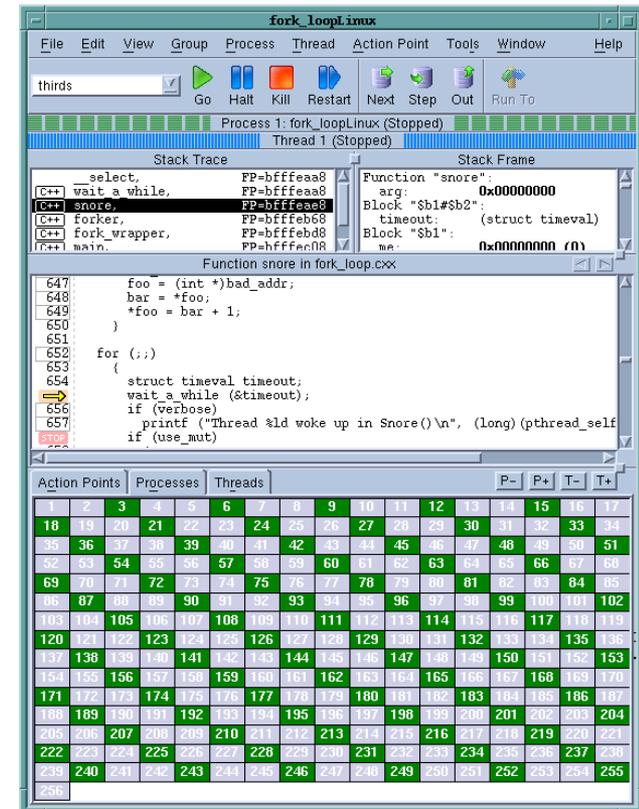
What is TotalView®?

Application Analysis and Debugging Tool: Code Confidently

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

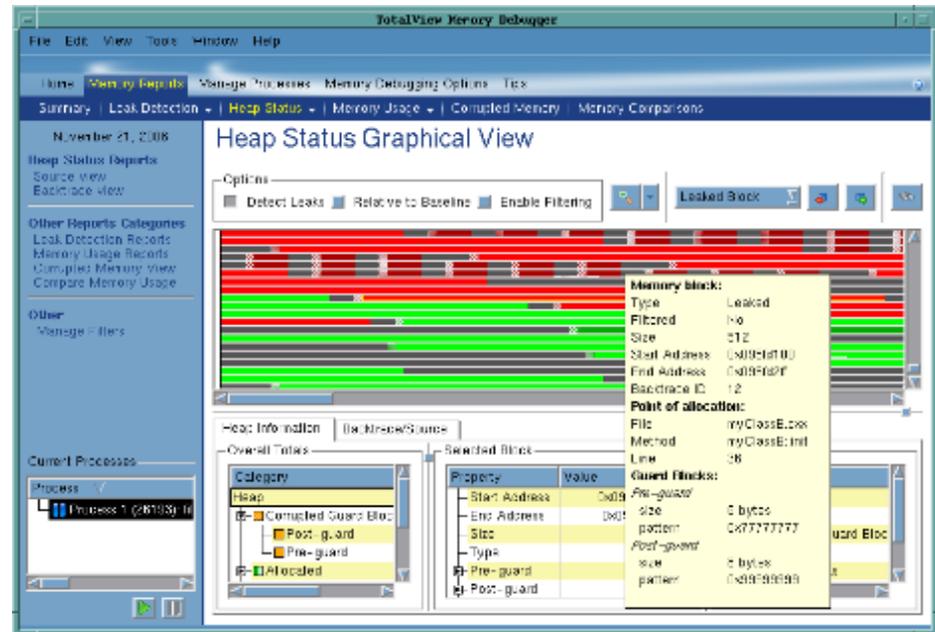
Major Features

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



MemoryScape®

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



Deterministic Replay Debugging



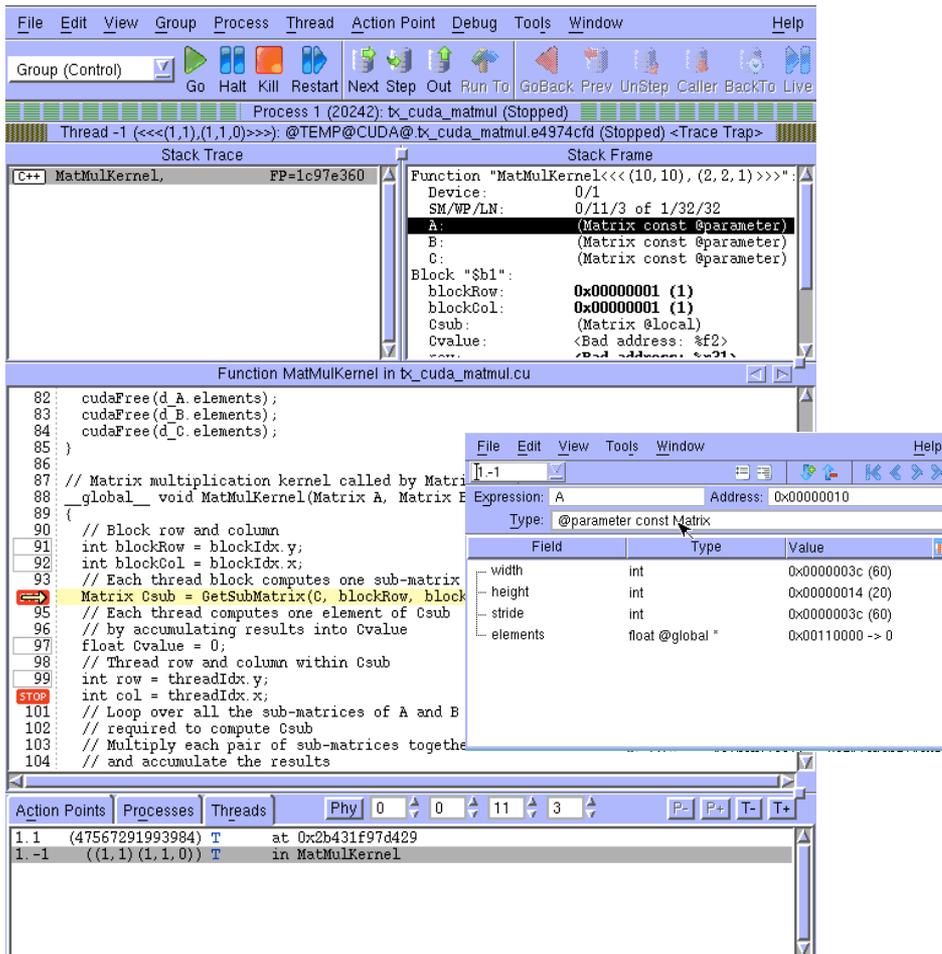
- Reverse Debugging: Radically simplify your debugging
 - Captures and Deterministically Replays Execution
 - Not just “checkpoint and restart”
 - Eliminate the Restart Cycle and Hard-to-Reproduce Bugs
 - Step Back and Forward by Function, Line, or Instruction
- Specifications
 - A feature included in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

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

Cambridge Study

- Survey conducted by the Judge Business School at Cambridge University concluded that Reverse Debuggers allow users, on average, to spend 13% less of their programming time debugging.
 - Programming was 50% of total work week on average
 - Debugging was 50% of programming time without reverse debugging
 - Debugging was 37% of programming time with reverse debugging
 - That frees up 130 hours (>3 work weeks, 6.5% total time) per developer per year for design and new feature development
- The survey looked at total value (salaries & overhead) of debugging as a task and they determined that this savings could, across the whole world economy, be worth \$41 billion in increased productivity.
 - The productivity improvement should be worth \$2,500 per developer per year (salary only) or \$5,000 per year with overhead.
- <http://www.roguewave.com/company/news-events/press-releases/2013/university-of-cambridge-reverse-debugging-study.aspx>

TotalView for the NVIDIA® GPU Accelerator



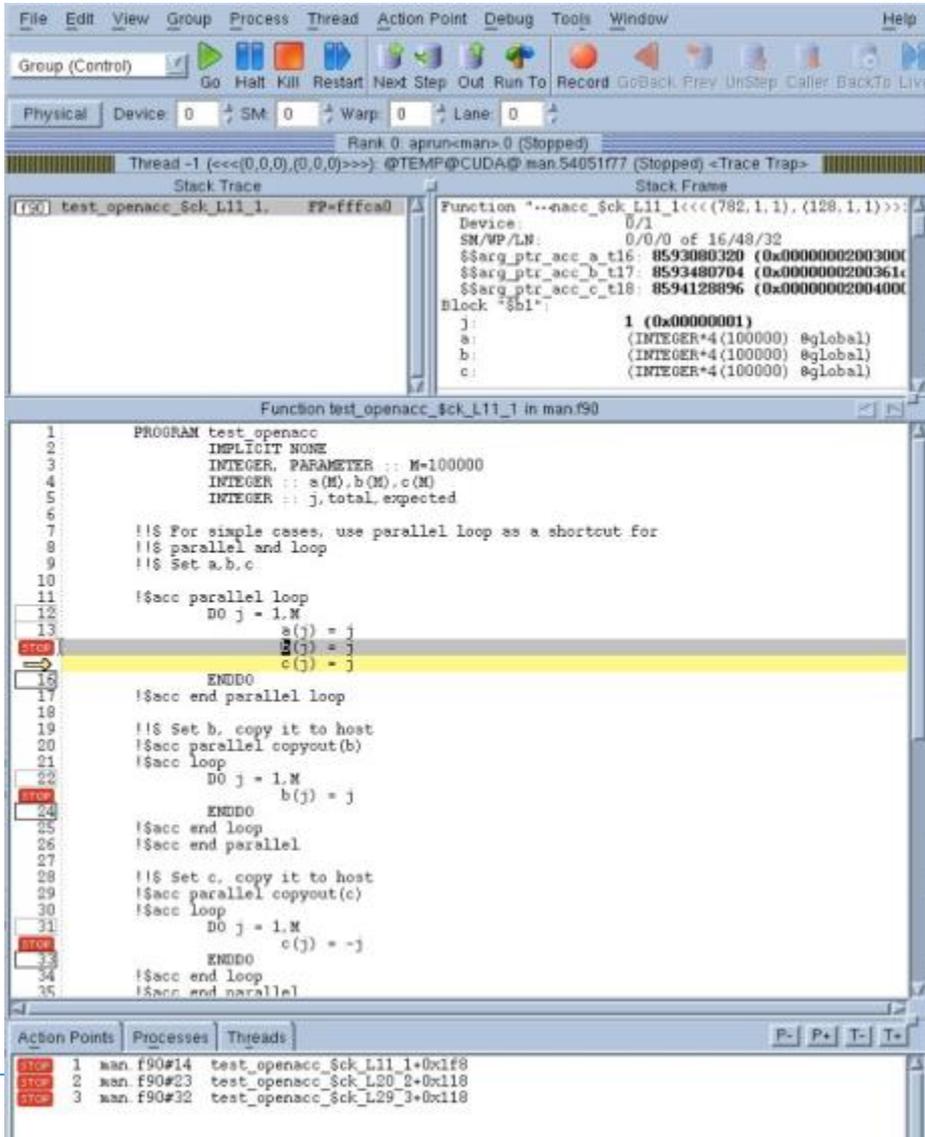
- NVIDIA CUDA 6.5
 - With support for Unified Memory
- Features and capabilities include
 - Support for dynamic parallelism
 - Support for MPI based clusters and multi-card configurations
 - Flexible Display and Navigation on the CUDA device
 - Physical (device, SM, Warp, Lane)
 - Logical (Grid, Block) tuples
 - CUDA device window reveals what is running where
 - Support for types and separate memory address spaces
 - Leverages CUDA memcheck

Displaying NVIDIA GPU Device Information

The screenshot shows a debugger window with a menu bar (File, Edit, View, Group, Process, Thread, Action Point, Debug, Tools, Window, Help) and a toolbar with various execution controls. Below the toolbar, there are dropdown menus for 'Physical', 'Device: 0', 'SM: 2', 'Warp: 0', and 'Lane: 0'. The main window displays a tree view of CUDA devices, with 'Device 0/3' expanded to show its internal structure.

Name	Description
Device 0/3	
Device Type	gf100
Lanes	32
SM 2/4	
Valid Warps	0000000000000001
Warp 00/48	Block (0,0,0)
Lane 00/32	Thread (0,0,0)
LPC	00000001d95f810
Lane 01/32	Thread (1,0,0)
LPC	00000001d95f810
Lane 02/32	Thread (0,1,0)
LPC	00000001d95f810
Lane 03/32	Thread (1,1,0)
LPC	00000001d95f810
Valid/Active/Divergent	0000000F, 0000000F, 00000000
SM 3/4	
Warp 00/48	Block (0,1,0)
Lane 00/32	Thread (0,0,0)
LPC	00000001d95f810
Lane 01/32	Thread (1,0,0)
LPC	00000001d95f810
Lane 02/32	Thread (0,1,0)
LPC	00000001d95f810
Lane 03/32	Thread (1,1,0)
LPC	00000001d95f810
Valid/Active/Divergent	0000000F, 0000000F, 00000000
SM 6/4	
Warp 00/48	Block (1,1,0)
Lane 00/32	Thread (0,0,0)
LPC	00000001d95f810
Lane 01/32	Thread (1,0,0)
LPC	00000001d95f810
Lane 02/32	Thread (0,1,0)
LPC	00000001d95f810
Lane 03/32	Thread (1,1,0)
LPC	00000001d95f810
Valid/Active/Divergent	0000000F, 0000000F, 00000000
SM 7/4	
Warp 00/48	Block (1,0,0)
Lane 00/32	Thread (0,0,0)
LPC	00000001d95f810
Lane 01/32	Thread (1,0,0)
LPC	00000001d95f810
Lane 02/32	Thread (0,1,0)
LPC	00000001d95f810
Lane 03/32	Thread (1,1,0)
LPC	00000001d95f810
Valid/Active/Divergent	0000000F, 0000000F, 00000000
SM Type	sm_20
SMs	14
Warps	48
Device 1/3	

TotalView for OpenACC



OpenACC®

DIRECTIVES FOR ACCELERATORS

- Step host & device
- View variables
- Set breakpoints

Compatibility with
Cray CCE 8
OpenACC

TotalView for the Intel® Xeon Phi™ coprocessor

Supports All Major Intel Xeon Phi Coprocessor Configurations

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

User Interface

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

Memory Debugging

- Both native and symmetric mode

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

The screenshot displays the TotalView debugger interface. At the top, there is a menu bar (File, Edit, View, Group, Process, Thread, Action Point, Debug, Tools, Window, Help) and a toolbar with various control icons. Below the toolbar, the 'Group (Control)' window shows a tree view of processes and threads, with 'Thread 2 (139985823807232) (At Breakpoint 6)' selected. The 'Stack Trace' window shows the call stack for the selected thread, with 'compute07' at the top. The 'Stack Frame' window shows the current function's context, including local variables and registers. The main window displays the source code for 'Function compute07 in sample07.c', with a yellow highlight on line 110: 'for (i=0; i<size; i++)'. The 'Action Points' window at the bottom shows a list of breakpoints for various threads.

Spectrum of Intel Xeon Phi Execution Models

CPU-Centric

Intel® Xeon Processor

Multi-core Hosted

Intel® Xeon Phi-Centric

Intel® Xeon Phi

Offload

Symmetric

Many-Core Hosted

General purpose
serial and parallel
computing

Codes with
highly-parallel
phases

Codes with
balanced needs

Highly-parallel
codes

Main()
Foo()
MPI_*()

Main()
Foo()
MPI_*()

Main()
Foo()
MPI_*()

Foo()

Main()
Foo()
MPI_*()

Main()
Foo()
MPI_*()

PCIe

Productive Programming Models Across the Spectrum

Debugging Intel Xeon Phi Applications with Offloaded Code

Xeon side

The Xeon side of the debugger interface shows the following components:

- Process List:** Process 1 (31634): intro_sample0.c.out (Stopped). Thread 1 (140091609065248) (Stopped).
- Stack Trace:** Shows the call stack for the stopped thread, including pthread_cond_wait, TaskScheduler::WaitForEvent_signal, and the main function.
- Function View:** Shows the function signature for 'sample08' with local variables: pi: 0, count: 0, i: 0.
- Code Editor:** Displays the source code for 'sample08.c'. The code includes OpenMP pragmas for offloading and parallelization. The current execution point is at line 54, which is highlighted in yellow.
- Thread List:** Shows the thread's execution history, including states like 'in pthread_cond_wait', 'in __poll', and 'in pthread_cond_wait'.

Xeon Phi side

The Xeon Phi side of the debugger interface shows the following components:

- Process List:** Process 2 (79768192_168_1_100): offload_main (Mixed). Thread 2 (139773936764672) (At Breakpoint 6).
- Stack Trace:** Shows the call stack for the thread, including L_sample08_51_par_loop1_2_19, pthread_cond_wait, and pthread_cond_timedwait.
- Function View:** Shows the function signature for 'L_sample08_51_par_loop1_2_19' with local variables: count: 0x00002710 (10000), pi: 0, t: 5e-05, i: 0x00000000 (0).
- Code Editor:** Displays the source code for 'L_sample08_51_par_loop1_2_19'. The code is the offloaded version of 'sample08.c'. The current execution point is at line 54, which is highlighted in yellow.
- Thread List:** Shows the thread's execution history, including states like 'in sew_wait', 'in L_sample08_51_par_loop1_2_19', 'in __poll', 'in pthread_cond_wait', 'in pthread_cond_timedwait', and 'in L_sample08_51_par_loop1_2_19'.

One debugging session for MIC-accelerated code

Debugging Intel Xeon Phi MPI Applications

The screenshot shows a debugger interface with the following components:

- Stack Trace:**
 - C main, FP=7fff0e864b60
 - C __libc_start_main, FP=7fff0e864c20
 - C _start, FP=7fff0e864c30
- Function main in tx_basic_mpi.c:**

```
84  
85 message = getenv("MESSAGE") ? getenv("MESSAGE") : "";  
86  
87 MPI_Init(&argc,&argv);  
88 MPI_Comm_size(MPI_COMM_WORLD,&numprocs);  
89 MPI_Comm_rank(MPI_COMM_WORLD,&myid);  
90  
91 if (myid == 0)  
92 {  
93     char* outfile = NULL;  
94     outfile = getenv("TX_OUTFILE");  
95  
96     if (outfile)  
97         out = fopen(outfile, "w");  
98  
99     if (!out)  
100         out = stdout;  
101     rank0(numprocs, out);  
102 }  
103 else  
104     rankn(myid);  
105  
106
```
- Select processes to attach to:**

Attach	Host	Comm	Rank	Program
<input checked="" type="checkbox"/>	192.168.1.100	0	0	/tmp/tx_basic_mpi
<input checked="" type="checkbox"/>	192.168.1.100	1	1	/tmp/tx_basic_mpi
<input checked="" type="checkbox"/>	192.168.1.100	2	2	/tmp/tx_basic_mpi
<input checked="" type="checkbox"/>	192.168.1.100	3	3	/tmp/tx_basic_mpi
<input checked="" type="checkbox"/>	192.168.1.100	4	4	/tmp/tx_basic_mpi
<input checked="" type="checkbox"/>	192.168.1.100	5	5	/tmp/tx_basic_mpi
<input checked="" type="checkbox"/>	192.168.1.100	6	6	/tmp/tx_basic_mpi
<input checked="" type="checkbox"/>	192.168.1.100	7	7	/tmp/tx_basic_mpi
<input checked="" type="checkbox"/>	192.168.1.100	8	8	/tmp/tx_basic_mpi
- Filters:**
 - Communicator: All
 - Array of Ranks:
 - Talking to Rank: All
 - List of Ranks:
 - Message Type: Send Receive Unexpected
 - Halt control group

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

Coarray Fortran

Diving on CAF array y

The screenshot shows a debugger window titled "y - hello_image - 2.1". The expression "y" is entered, with an address of "0x009caa40" and a type of "INTEGER*8(10)[*]". The slice is set to "(, :)". Below the input fields is a table with two columns: "Field" and "Value".

Field	Value
(1)[1]	1 (0x0000000000000001)
(2)[1]	1 (0x0000000000000001)
(3)[1]	1 (0x0000000000000001)
(4)[1]	1 (0x0000000000000001)
(5)[1]	1 (0x0000000000000001)
(6)[1]	1 (0x0000000000000001)
(7)[1]	1 (0x0000000000000001)
(8)[1]	1 (0x0000000000000001)
(9)[1]	1 (0x0000000000000001)

Diving on CAF array y
across processes

The screenshot shows a debugger window titled "y - hello_image - 2.1" with the same expression and type as the first window. The slice is "(, :)". The table below has three columns: "Field", "Process", and "Value".

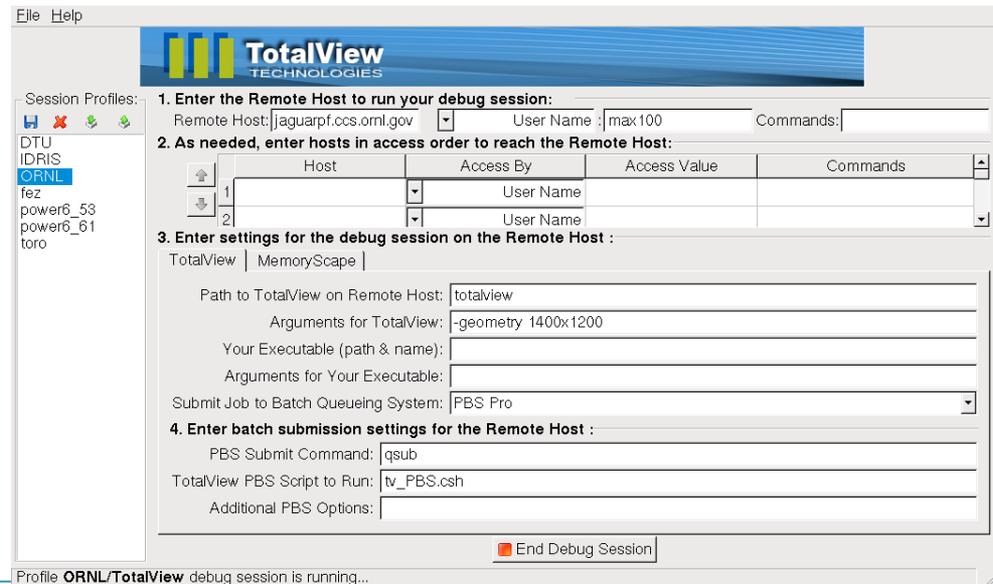
Field	Process	Value
(1)[1]	hello_image.1	1 (0x0000000000000001)
(1)[2]	hello_image.2	2 (0x0000000000000002)
(1)[3]	hello_image.3	3 (0x0000000000000003)
(1)[4]	hello_image.4	4 (0x0000000000000004)
(2)[1]	hello_image.1	1 (0x0000000000000001)
(2)[2]	hello_image.2	2 (0x0000000000000002)
(2)[3]	hello_image.3	3 (0x0000000000000003)
(2)[4]	hello_image.4	4 (0x0000000000000004)
(3)[1]	hello_image.1	1 (0x0000000000000001)
(3)[2]	hello_image.2	2 (0x0000000000000002)

Supported on Cray
platforms with CCE

Advanced Features

Remote Display Client

- Offers users the ability to easily set up and operate a TotalView debug session that is running on another system
- Consists of two components
 - Client – runs on local machine
 - Server – runs on any system supported by TotalView and “invisibly” manages the secure connection between host and client
- Remote Display Client is available for:
 - Linux x86, x86-64
 - Windows XP, Vista, 7
 - Mac OS X



Multi-dimensional array viewer

- See your arrays on a Grid display
- 2-D, 3-D, ...N-D
- Arbitrary slices
- Specify data representation
- Windowed data access – Fast

Array Viewer: int_4D_array[i][j][k][l]

File Help

Expression: Type: int[5][7][9][11]

Enter the array slice to display:

	Dimension	Start Index	End Index	Stride
Row	[i]	0	4	1
Column	[j]	0	6	1

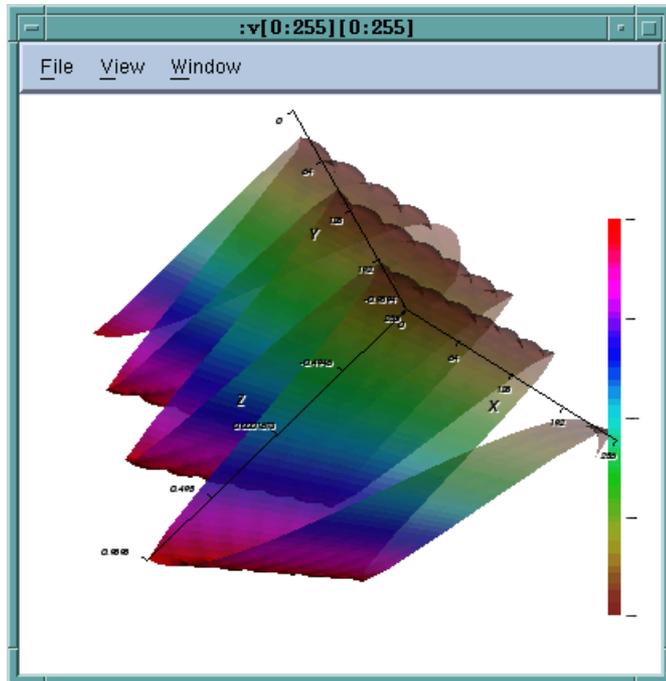
Select an index for the other dimensions:

[i] [j] [k] [l]

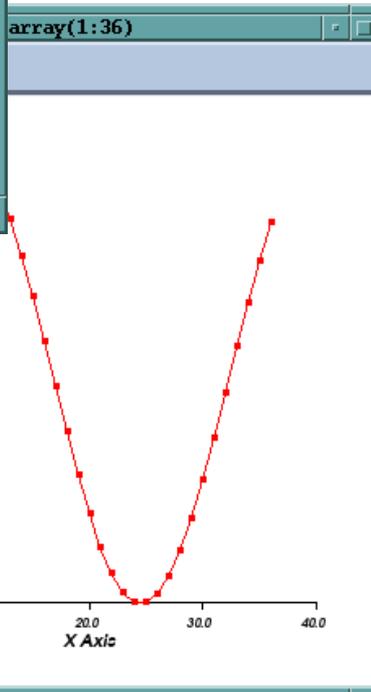
Automatic Slice: [0:4:1][0:6:1][0:0:1][0:0:1]

	[j] : 0	1	2	3
[i] : 0	0x00000000 (0)	0x00000064 (100)	0x000000c8 (200)	0x0000012c (300)
1	0x000003e8 (1000)	0x0000044c (1100)	0x000004b0 (1200)	0x00000514 (1300)
2	0x000007d0 (2000)	0x00000834 (2100)	0x00000898 (2200)	0x000008fc (2300)
3	0x00000bb8 (3000)	0x00000c1c (3100)	0x00000c80 (3200)	0x00000ce4 (3300)
4	0x00000fa0 (4000)	0x00001004 (4100)	0x00001068 (4200)	0x000010cc (4300)

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.

Debugging MPMD applications

totalview -args aprun -n 9 worker : -n 1 master

TotalView 8.14.1-8

ID	Rank	Host	Status	Description
1	<local>		R	aprun (2 active threads)
2	0 nid00048		B	aprun<worker>.0 (2 active threads)
3	1 nid00048		B1	aprun<worker>.1 (1 active threads)
4	2 nid00048		B1	aprun<worker>.2 (1 active threads)
5	3 nid00048		B1	aprun<worker>.3 (1 active threads)
6	4 nid00048		B1	aprun<worker>.4 (1 active threads)
7	5 nid00048		B1	aprun<worker>.5 (1 active threads)
8	6 nid00048		B1	aprun<worker>.6 (1 active threads)
9	7 nid00048		B1	aprun<worker>.7 (1 active threads)
10	8 nid00048		B1	aprun<worker>.8 (1 active threads)
11	9 nid00049		R	aprun<master>.9 (1 active threads)

aprun<worker>.2

Group (Control) [Go] [Halt] [Kill] [Restart] [Next Step] [Out] [Run To] [Record] [GoBack] [Prev] [UnStep] [Caller] [BackTo] [Live]

Rank 2: aprun<worker>.2 (At Breakpoint 1)
Thread 1 (18081): worker (At Breakpoint 1)

Stack Trace

FP	Function	FP
f90	worker,	FP=7fffffff7d20
C	__libc_start_main,	FP=7fffffff7de0
	_start,	FP=7fffffff7df0

Stack Frame

Function "worker":
No arguments.
Local variables:
dummy: 0 (0x00000000)
err: 0 (0x00000000)
job: 0 (0x00000000)
master: 0 (0x00000000)
mpi_argvs_null: (character (len=1) (1,1))
mpi_argv_null: (character (len=1) (1))
mpi_bottom: 0 (0x00000000)
mpi_errcodes_ignore: (INTEGER+4 (1))
mpi_in_place: 0 (0x00000000)

Function worker in worker.F90

```
11  INTEGER :: nprocs, myrank, tag, dummy, err
12  INTEGER, DIMENSION(MPI_STATUS_SIZE) :: stat
13  INTEGER :: master
14
15  ! Count variables
16  INTEGER job
17
18  CALL MPI_INIT(err)
19  CALL MPI_COMM_SIZE(MPI_COMM_WORLD, nprocs, err)
20  CALL MPI_COMM_RANK(MPI_COMM_WORLD, myrank, err)
21
22  ! The main loop
23  master= nprocs-1
24  tag = 1
25  dummy = 0
26  DO
27    CALL MPI_SEND(dummy, 1, MPI_INTEGER, master, tag, MPI_COMM_WORLD, err)
28    CALL MPI_RECV(job, 1, MPI_INTEGER, master, tag, MPI_COMM_WORLD, stat, err)
29    IF (job == -1) EXIT
30
31    write (*,*) "Worker ", myrank, " does work number ", job
32
33  ! The work that the workers do
```

Action Points | Processes | Threads

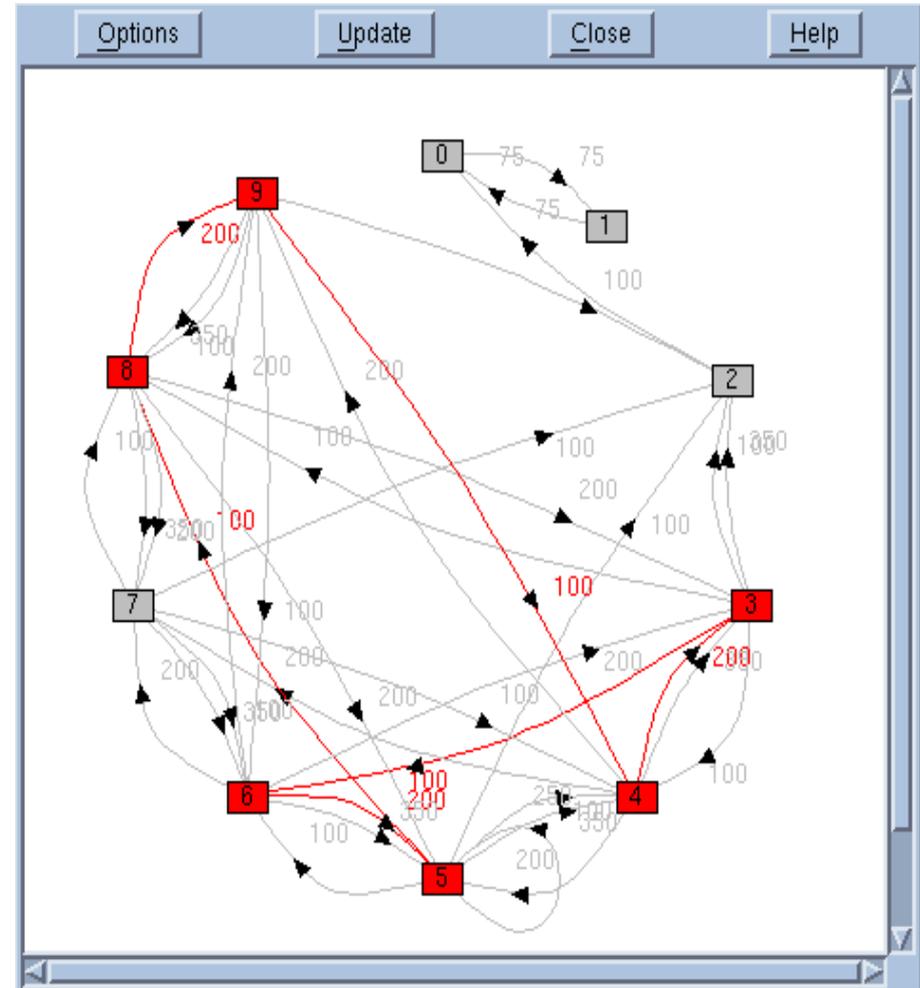
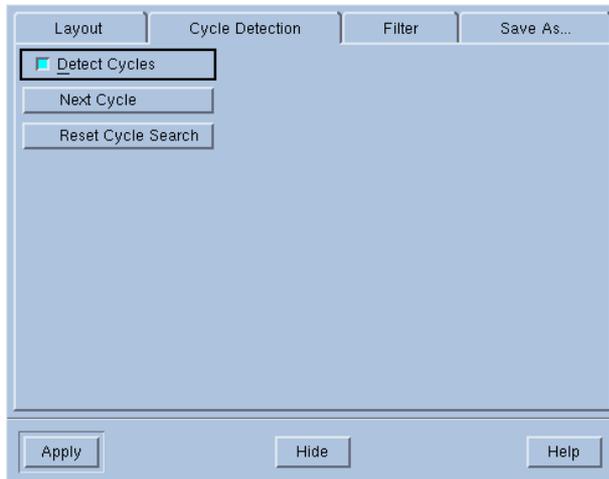
P- P+ Px T- T+

p1 0 1 2 3 4 5 6 7 8 9

Message Queue Graph

Message Queue Debugging

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

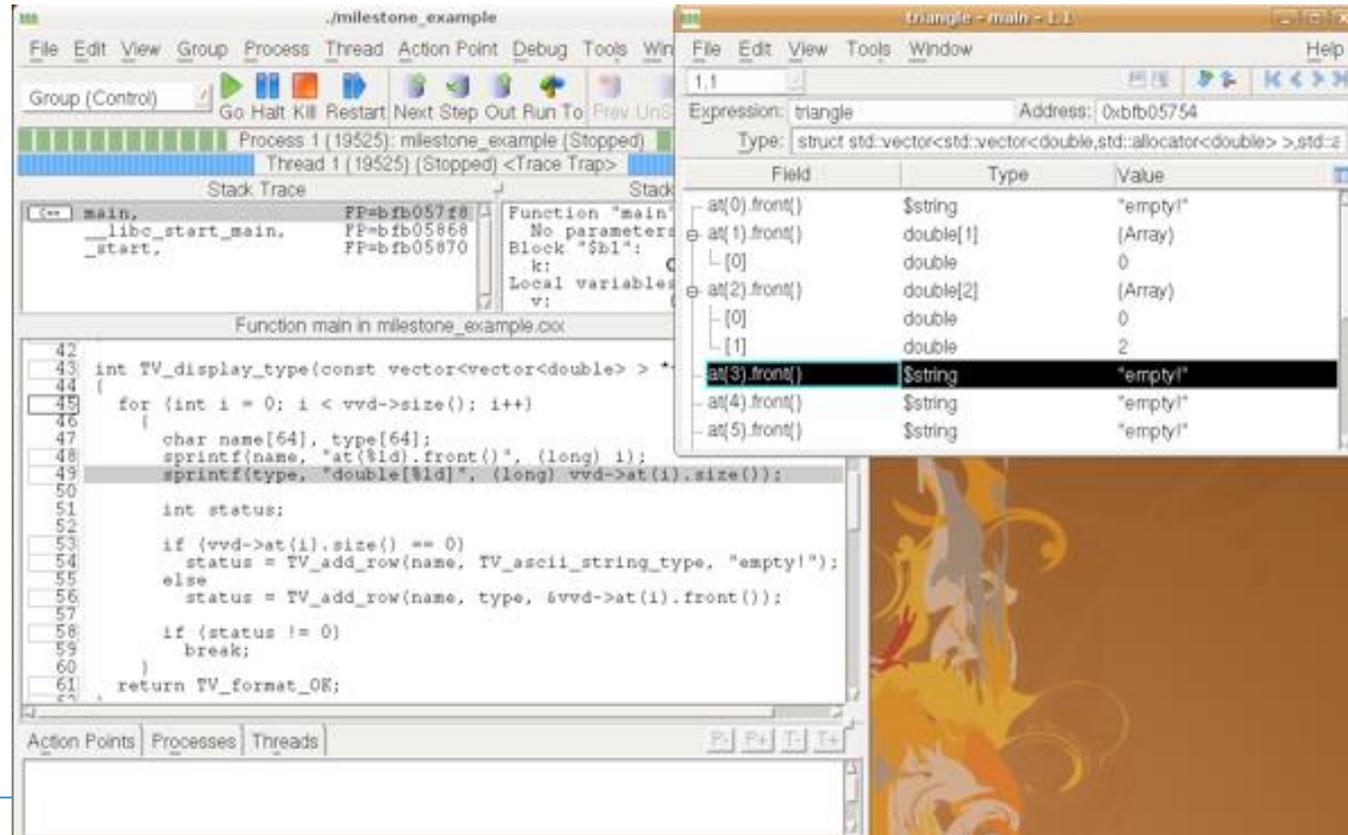


TVScript

- Gives you non-interactive access to TotalView's capabilities
- Useful for
 - Debugging in batch environments
 - Watching for intermittent faults
 - Parametric studies
 - Automated testing and validation
- TVScript is a script (not a scripting language)
 - It runs your program to completion and performs debugger actions on it as you request
 - Results are written to an output file
 - No GUI
 - No interactive command line prompt
- Used at sites such as DMI and STFC Daresbury for automated comparative debugging

C++View

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



Viewing Fortran User-Defined Types

TYPE WHOPPER

LOGICAL, DIMENSION(ISIZE) :: FLAGS

DOUBLE PRECISION, DIMENSION(ISIZE) :: DPSA

DOUBLE PRECISION, DIMENSION(:), POINTER :: DPPA

END TYPE WHOPPER

TYPE(WHOPPER), DIMENSION(:), ALLOCATABLE :: STUFFTYP1

The screenshot shows a window titled 'stuff - f90stepAlpha - 1.1'. The 'Expression' field contains 'stufftyp1' and the 'Address' is '0x140000090 [Sparse]'. The 'Actual Type' and 'Type' fields both show 'type(whopper),allocatable::(100)'. Below this is a table with three columns: 'Field', 'Type', and 'Value'.

Field	Type	Value
(1)	type(whopper)	(Struct)
-- flags	logical*4(1000)	(Array)
-- dpsa	double precision(1000)	(Array)
-- dppa	double precision,pointer:(double precision,pointer)	
(2)	type(whopper)	(Struct)
(3)	type(whopper)	(Struct)
(4)	type(whopper)	(Struct)
(5)	type(whopper)	(Struct)
(6)	type(whopper)	(Struct)
(7)	type(whopper)	(Struct)
(8)	type(whopper)	(Struct)

Current Work and Future Plans

What is new in TotalView 8.14.1

- CUDA 6.5 support
- Coarray Fortran support for the Cray CCE compiler
- Extended support for type transformations with the Intel compiler (unordered STL collection classes)
- Improved delayed symbol processing (better performance for larger executables)

Multi-phase R&D Projects Underway

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

Thanks!

- Visit the website
 - <http://www.roguewave.com/products/totalview.aspx>
 - Videos (3 new videos on Xeon Phi)
 - Documentation
 - Sign up for an evaluation
- Visit us at SC14 (booth 2338)



hpc matters.