ATPESC 2015

TotalView: Debugging from Desktop to Supercomputer

Peter Thompson
Principal Software Support Engineer
August 12, 2015

RogueWave SOFTWARE

Accelerating Great Code

What we do

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

Capabilities

APPLICATION SECURITY

Klocwork, OpenLogic, TotalView, IMSL, SourcePro

CODE REFACTORING

Klocwork

CODE REVIEW

Klocwork, OpenLogic

DEBUGGING COMPLEX CODE

Klocwork, TotalView

REUSABLE MATH ALGORITHMS

IMSL, SourcePro

OPEN SOURCE AUDITING

OpenLogic

OPEN SOURCE MANAGEMENT

OpenLogic

OPEN SOURCE SUPPORT

OpenLogic

CERTIFIED OPEN SOURCE

OpenLogic

STATIC CODE ANALYSIS

Klocwork

DEVELOPING USER INTERFACES

Visualization, Stingray, PV-WAVE

CODE MIGRATION

SourcePro, IMSL, HydraExpress

CODE BUILDING BLOCKS

SourcePro, IMSL, Stingray, Visualization

Global, diversified customer base

Used by 3,000 customers in over 57 countries across diverse industries to develop mission-critical applications and software

Financial Services

Bank of America
citibank
Credit Suisse
Liberty Mutual
Goldman Sachs
QUESTRADE
Bloomberg
PRUDENTIAL
MorganStanley
UBS
BARCLAYS
MIZUHO
E*TRADE
Deutsche Bank
JPMorganChase

Telecom

at&t
HUAWEI
BT
verizon
Alcatel·Lucent
GenCorp
Sprint
MOTOROLA
Polycom
QUALCOMM
ERICSSON
L3
BlackBerry

Gov't / Defense

BOEING
NASA
NORTHROP GRUMMAN
BAE SYSTEMS
Raytheon
Lawrence Livermore National Laboratory
LOCKHEED MARTIN
Tokyo Institute of Technology
Sandia National Laboratories

Technology

CISCO
ca
ORACLE
IBM
bmcsoftware
DELL
EMC² where information lives
RSA SECURITY
hp
SUNGARD
fiserv.
SAP

Other Verticals

PIXAR ANIMATION STUDIOS
ExxonMobil
McKESSON Empowering Healthcare
Schlumberger
Disney
DELPHI
chevron
INDUSTRIAL LIGHT & MAGIC
Shell
Cerner
OLYMPUS
LG
SIEMENS
MARATHON
أرامكو السعودية Saudi Aramco
DREAMWORKS
Schneider Electric
TOTAL
JOHN DEERE

Debugging occurs in many industries

They use software to deliver value or inform decisions

Financial services

Oil and Gas

Aerospace and Defense

Engineering

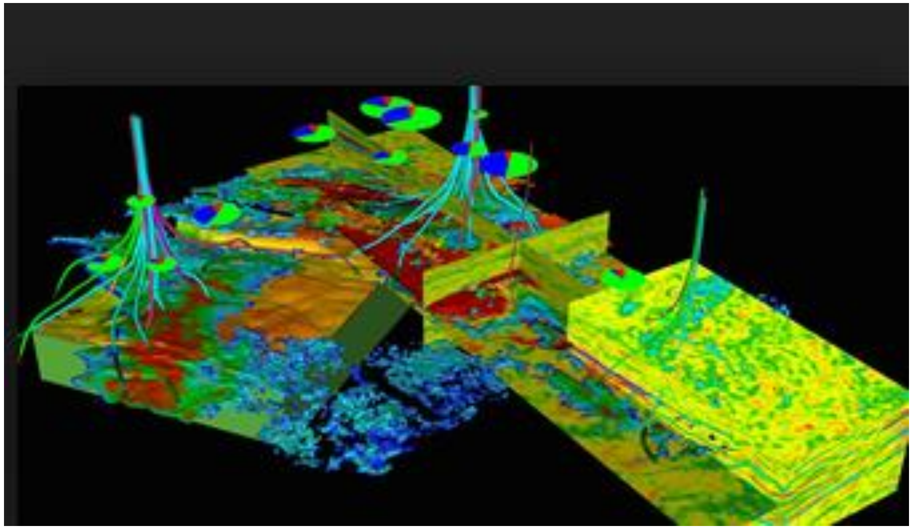
Digital Content Creation

ISVs

Biological sciences

Scientific and Technical Computing

High Performance Computing



HPC Trends

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

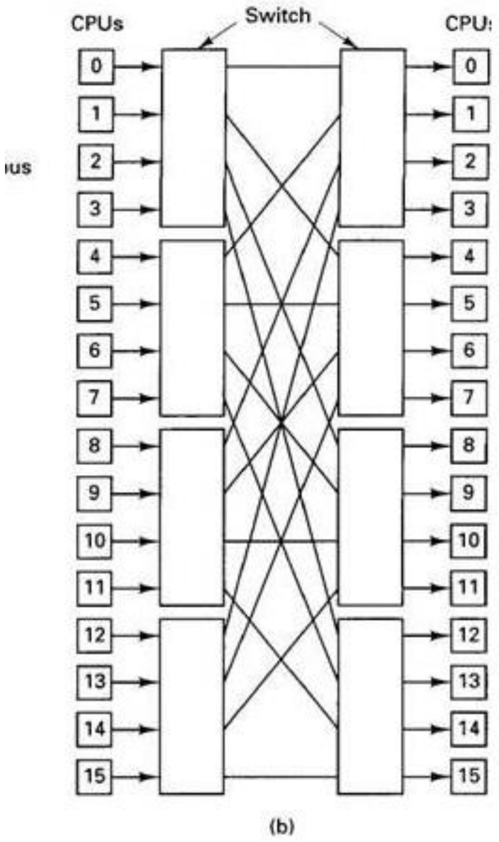
How does Rogue Wave help?

TotalView debugger

- Troubleshooting and analysis tool
  - Visibility into applications
  - Control over applications
- Scalability
- Usability
- Support for HPC platforms and languages
- Student Express license available for both undergraduate and grad students

TotalView Overview

TotalView Origins



Mid-1980’s Bolt, Berenak, and Newman (BBN) Butterfly Machine
An early ‘Massively Parallel’ computer

How do you debug a Butterfly?

- TotalView project was developed as a solution for this environment
- Able to debug multiple processes and threads
- Point and click interface
- C, C++, and Fortran (and assembler)

A solution in search of a problem…

- From the Butterfly, TotalView was ported to other machines
  - IBM RS6000, Cray, Solaris Sparc, DEC Alpha, Irix…
- As various MPI’s were being developed, Bill Gropp, Rusty Lusk and Jim Cownie worked on an interface for automatic process acquisition
- Some years later, Bill and Jim developed another interface for showing Message Queue information

Other capabilities added

- Support for most types of MPI
- Linux
- Lightweight Memory Debugging
- Type transformations
- Memscript and tvscript
- Reverse Debugging
- Remote Display Client
- GPU debugging
- Intel Xeon Phi

Key features of TotalView

- Interactive Debugging
- Interactive Memory Debugging
- Reverse Debugging
- Unattended Debugging

Serial, Parallel and Accelerated applications

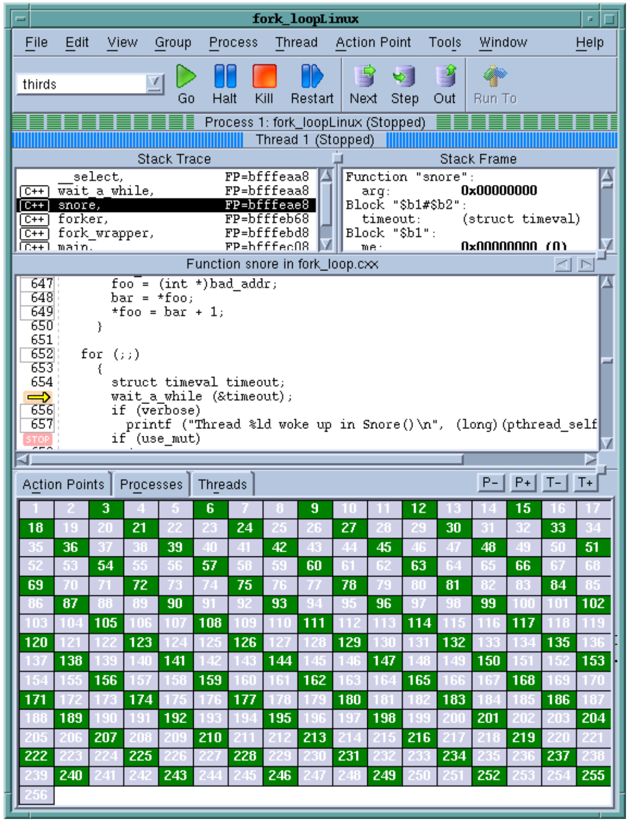
What is TotalView®?

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™, Fortran Coarrays
  - CUDA™, OpenACC®, and Intel® Xeon Phi™ coprocessor
- Low tool overhead resource usage
- Includes a Remote Display Client which frees you to work from anywhere
- Memory Debugging with MemoryScape™
- Deterministic Replay Capability Included on Linux/x86-64
- Non-interactive Batch Debugging with TVScript and the CLI
- TTF & C++View to transform user defined objects



Process 1: fork_loopLinux (Stopped)

Thread 1 (Stopped)

Stack Trace

```
      __select,          FP=bfffeaa8
C++   wait_a_while,      FP=bfffeaa8
C++   snore,             FP=bfffeae8
C++   forker,            FP=bfffeb68
C++   fork_wrapper,      FP=bfffebd8
C++   main,              FP=bfffec08
```

Stack Frame

```
Function "snore":
  arg:          0x00000000
Block "$b1#$b2":
  timeout:      (struct timeval)
Block "$b1":
  me:           0x00000000 (0)
```

Function snore in fork_loop.cxx

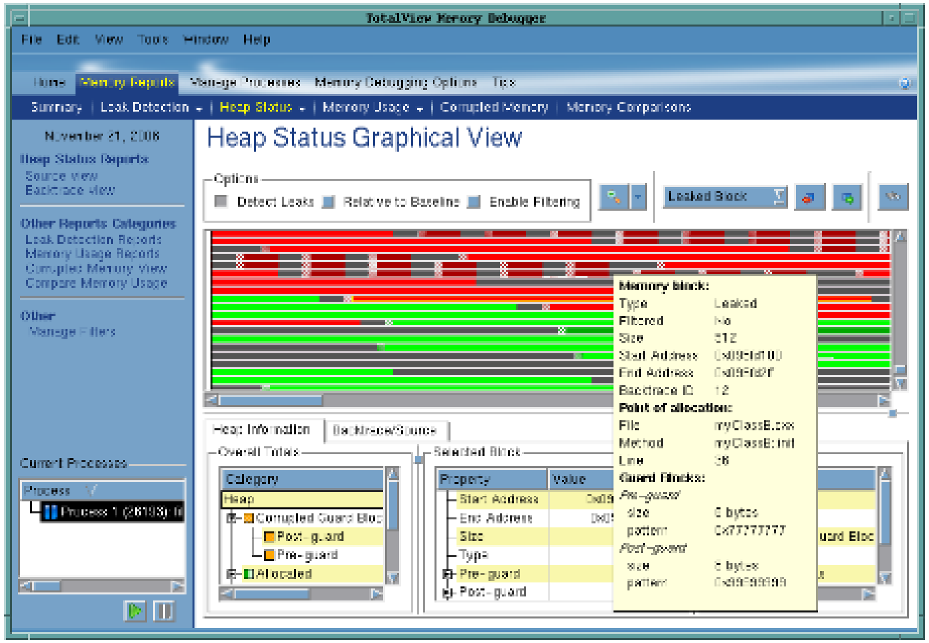
```
        foo = (int *)bad_addr;
        bar = *foo;
        *foo = bar + 1;
      }

    for (;;)
      {
        struct timeval timeout;
        wait_a_while (&timeout);
        if (verbose)
          printf ("Thread %ld woke up in Snore()\n", (long)(pthread_self
        if (use_mut)
```

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	128	129	130	131	132	133	134	135	136
137	138	139	140	141	142	143	144	145	146	147	148	149	150	151	152	153
154	155	156	157	158	159	160	161	162	163	164	165	166	167	168	169	170
171	172	173	174	175	176	177	178	179	180	181	182	183	184	185	186	187
188	189	190	191	192	193	194	195	196	197	198	199	200	201	202	203	204
205	206	207	208	209	210	211	212	213	214	215	216	217	218	219	220	221
222	223	224	225	226	227	228	229	230	231	232	233	234	235	236	237	238
239	240	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255
256																

What Is MemoryScape®?

- Runtime Memory Analysis : Eliminate Memory Errors
  - Detects memory leaks *before* they are a problem
  - Explore heap memory usage with powerful analytical tools
  - Use for validation as part of a quality software development process

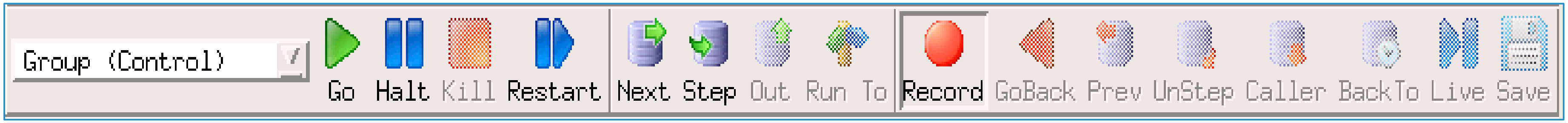
- Major Features
  - Included in TotalView, or Standalone
  - Detects
    - Malloc API misuse
    - Memory leaks
    - Buffer overflows
  - Supports
    - C, C++, Fortran
    - Linux, Unix, and Mac OS X
    - Intel® Xeon Phi™
    - MPI, pthreads, OMP, and remote apps
  - Low runtime overhead
  - Easy to use
    - Works with vendor libraries
    - No recompilation or instrumentation



Memory block:

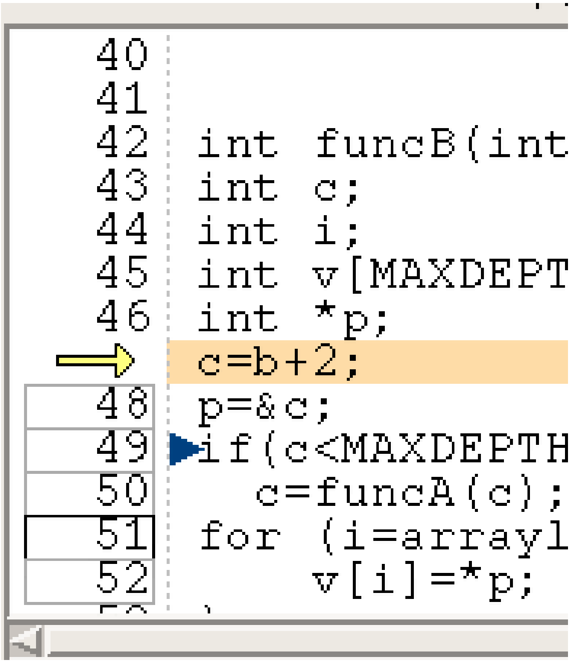
Type	Leaked
Filtered	No
Size	512
Start Address	0x0951d100
End Address	0x0951d2ff
Backtrace ID	12
Point of allocation:	
File	myClassB.cxx
Method	myClassB::init
Line	36
Guard Blocks:	
Pre-guard	
size	8 bytes
pattern	0x77777777
Post-guard	
size	8 bytes
pattern	0x99999999

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



```
int  funcB(int
int  c;
int  i;
int  v[MAXDEPT
int  *p;
c=b+2;
p=&c;
if(c<MAXDEPTH
   c=funcA(c);
for  (i=arrayl
     v[i]=*p;
```

Memscript and Tvscript

- Command line invocation to run TotalView and Memoryscape unattended
- Tvscript can be used to set breakpoints, take actions at those breakpoints and have the results logged to a file. It can also do memory debugging
  - tvscript –create_actionpoint “method1=>display_backtrace show_arguments” \ -create_actionpoint “method.c#342=>print x” myprog –a dataset 1
- Memscript can be used to run memory debugging on processes and display data when a memory event takes place. Exit is ALWAYS an event

```
Memscrip  -event_action \
"alloc_null=list_allocations,any_event=check_guard_blocks” \
-guard_blocks  -maxruntime "00:30:00”  -display_specifiers \
"noshow_pc,noshow_block_address,show_image”\
 myProgram -a myProgramArg1
```

- Memscript data can be saved in html, memory debug file, text heap status file

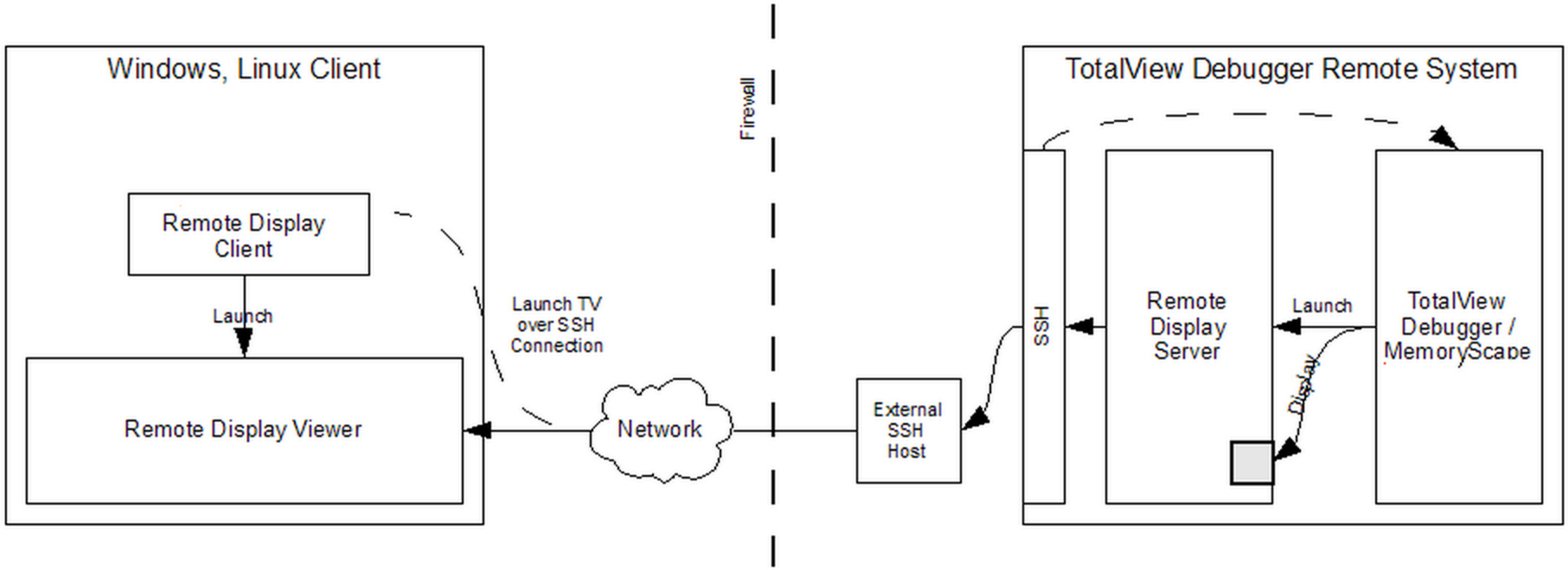
Dassault Systems

- ***"MemoryScape enabled us to identify memory issues, and by using its scripting interface, we were able to automate the evaluation process. Now, the system automatically uncovers any hidden latent errors in our code with every build, allowing our developers to proactively fix potential errors prior to release."***
  - **Nick Monyatovsky, Software Engineer at SIMULIA**
- Computer Aided Engineering ISV for Aero/Auto/Industry
- Struggling with intermittent errors
- Continuous Integration – Better Product Quality

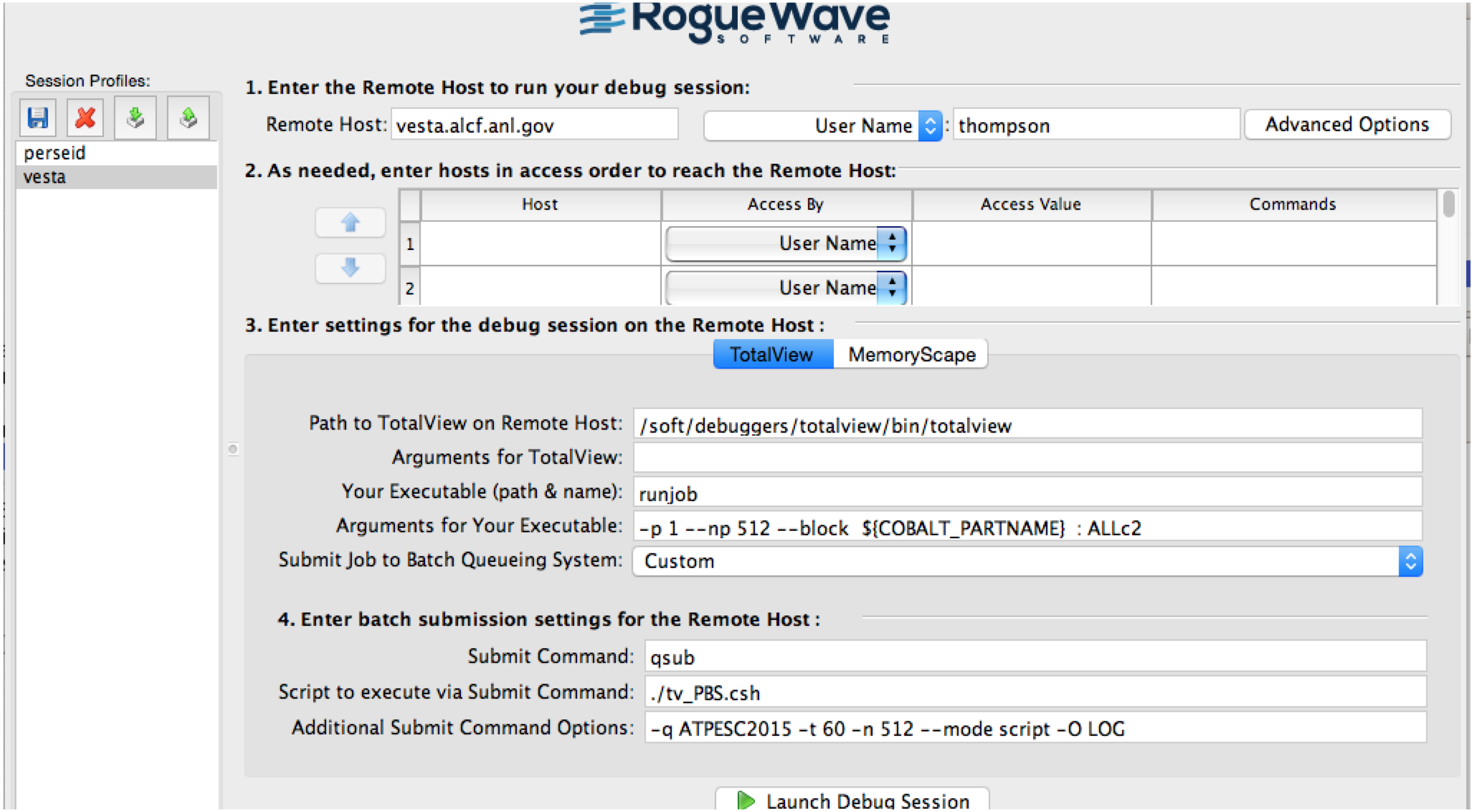
Remote Display Client (RDC)

- Push X11 bits and events across wide networks can be painful. The RDC can help

Figure 17 – Remote Display Components



The RDC setup



Session Profiles:

- perseid
- vesta

1. Enter the Remote Host to run your debug session:

Remote Host: vesta.alcf.anl.gov User Name: thompson

2. As needed, enter hosts in access order to reach the Remote Host:

	Host	Access By	Access Value	Commands
1		User Name		
2		User Name		

3. Enter settings for the debug session on the Remote Host :

Path to TotalView on Remote Host: /soft/debuggers/totalview/bin/totalview

Arguments for TotalView:

Your Executable (path & name): runjob

Arguments for Your Executable: -p 1 --np 512 --block ${COBALT_PARTNAME} : ALLc2

Submit Job to Batch Queueing System: Custom

4. Enter batch submission settings for the Remote Host :

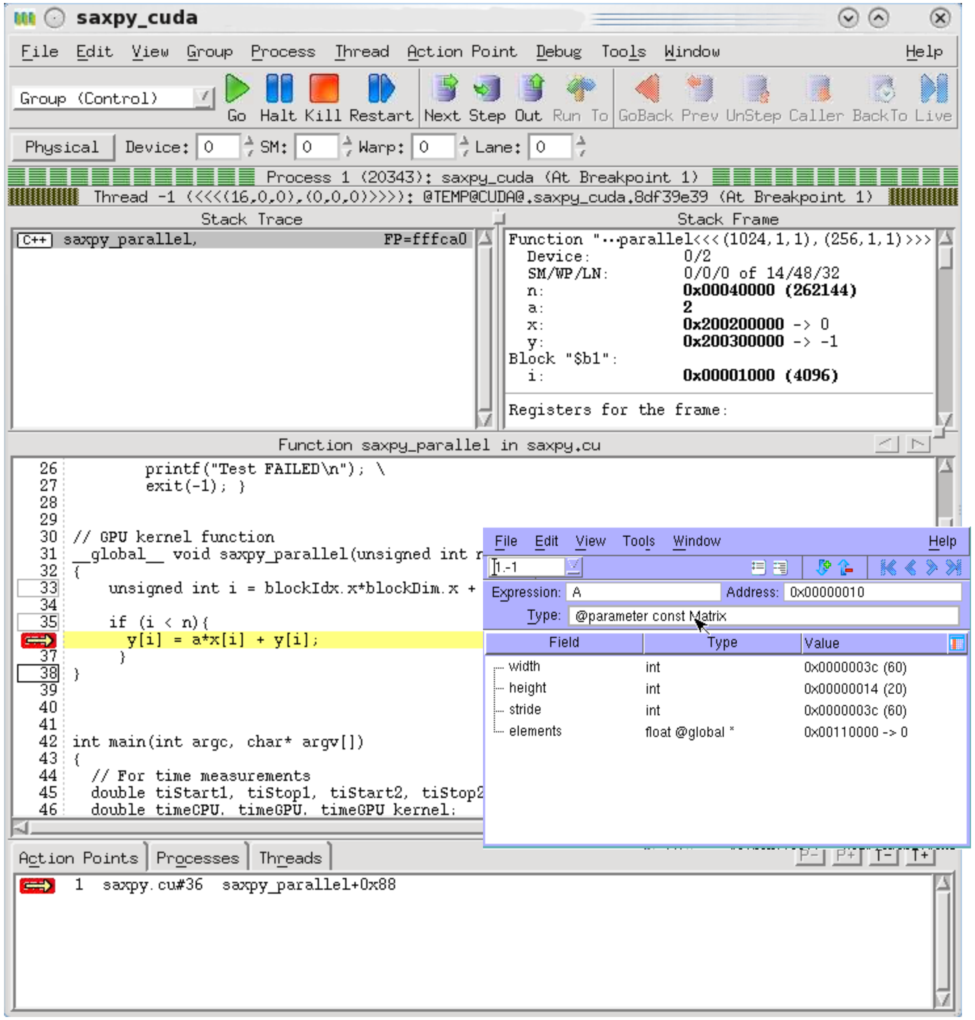
Submit Command: qsub

Script to execute via Submit Command: ./tv_PBS.csh

Additional Submit Command Options: -q ATPESC2015 -t 60 -n 512 --mode script -O LOG

Support for New Platforms

TotalView for the NVIDIA ® GPU Accelerator


saxpy_cuda
File Edit View Group Process Thread Action Point Debug Tools Window Help
Group (Control)
Go Halt Kill Restart Next Step Out Run To GoBack Prev UnStep Caller BackTo Live
Physical
Stack Trace
Stack Frame
File Edit View Tools Window Help
Action Points Processes Threads
P- P+ T- T+

Device: 0 SM: 0 Warp: 0 Lane: 0

```
Process 1 (20343): saxpy_cuda (At Breakpoint 1)
Thread -1 (<<<(16,0,0),(0,0,0)>>>): @TEMP@CUDA@.saxpy_cuda.8df39e39 (At Breakpoint 1)
```

```
C++ saxpy_parallel,                    FP=fffca0
```

```
Function "…parallel<<<(1024,1,1),(256,1,1)>>>
  Device:                0/2
  SM/WP/LN:              0/0/0 of 14/48/32
  n:                     0x00040000 (262144)
  a:                     2
  x:                     0x200200000 -> 0
  y:                     0x200300000 -> -1
Block "$b1":
  i:                     0x00001000 (4096)

Registers for the frame:
```

Function saxpy_parallel in saxpy.cu

```
          printf("Test FAILED\n"); \
          exit(-1); }


// GPU kernel function
__global__ void saxpy_parallel(unsigned int r
{
    unsigned int i = blockIdx.x*blockDim.x +

    if (i < n){
      y[i] = a*x[i] + y[i];
    }
}



int main(int argc, char* argv[])
{
  // For time measurements
  double tiStart1, tiStop1, tiStart2, tiStop2
  double timeCPU, timeGPU, timeGPU kernel;
```

1.-1

Expression: A Address: 0x00000010

Type: @parameter const Matrix

Field	Type	Value
width	int	0x0000003c (60)
height	int	0x00000014 (20)
stride	int	0x0000003c (60)
elements	float @global *	0x00110000 -> 0

```
1   saxpy.cu#36   saxpy_parallel+0x88
```

- NVIDIA CUDA 6.0, 6.5 and 7.0
- 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 CUDA Core debugging
  - Leverages CUDA memcheck
  - Support for 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

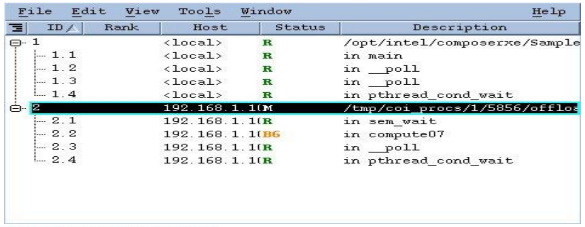
User Interface

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

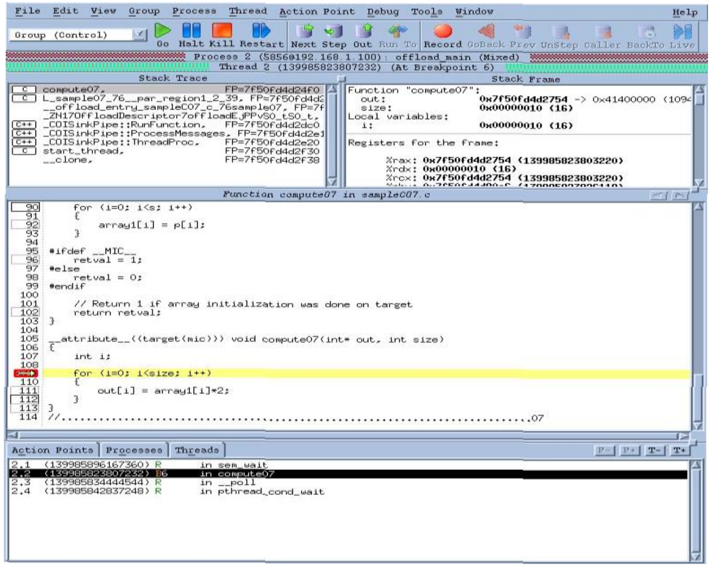
Memory Debugging

- Both native and symmetric mode

Anticipated support for KNL in late 2015 – early 2016



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		192.168.1.1(	M	/tmp/coi_procs/1/5856/offloa
2.1		192.168.1.1(	R	in sem_wait
2.2		192.168.1.1(	B6	in compute07
2.3		192.168.1.1(	R	in __poll
2.4		192.168.1.1(	R	in pthread_cond_wait



Process 2 (5856@192.168.1.100): offload_main (Mixed)
Thread 2 (139985823807232) (At Breakpoint 6)

```
C    compute07,                                  FP=7f50fd4d24f0
C    L_sample07_76__par_region1_2_39, FP=7f50fd4d2
     __offload_entry_sampleC07_c_76sample07, FP=7f
     _ZN17OffloadDescriptor7offloadEjPPvS0_tS0_t,
C++  _COISinkPipe::RunFunction,     FP=7f50fd4d2dc0
C++  _COISinkPipe::ProcessMessages, FP=7f50fd4d2e1
C++  _COISinkPipe::ThreadProc,      FP=7f50fd4d2e20
C    start_thread,                  FP=7f50fd4d2f30
     __clone,                       FP=7f50fd4d2f38
```

```
Function "compute07":
  out:                 0x7f50fd4d2754 -> 0x41400000 (1094
  size:                0x00000010 (16)
Local variables:
  i:                   0x00000010 (16)

Registers for the frame:

        %rax: 0x7f50fd4d2754 (139985823803220)
        %rdx: 0x00000010 (16)
        %rcx: 0x7f50fd4d2754 (139985823803220)
```

Function compute07 in sampleC07.c

```
    for (i=0; i<s; i++)
    {
        array1[i] = p[i];
    }

#ifdef __MIC__
    retval = 1;
#else
    retval = 0;
#endif

    // Return 1 if array initialization was done on target
    return retval;
}

__attribute__((target(mic))) void compute07(int* out, int size)
{
    int i;

    for (i=0; i<size; i++)
    {
        out[i] = array1[i]*2;
    }
}
//.................................................................07
```

```
2.1  (139985896167360) R    in sem_wait
2.2  (139985823807232) B6   in compute07
2.3  (139985834444544) R    in __poll
2.4  (139985842837248) R    in pthread_cond_wait
```

Knights Landing Memory

- KNL will have on-board High Bandwidth Memory (HBM) which can be accessed much faster than going out to main memory.
  - Cache
  - Explicitly managed for placement of frequently accessed data

- MemoryScape will be able to track allocations made both the standard heap and the on-chip HBM

- Optimization may include making sure that the right data structures are available to the processor in HBM
  - MemoryScape can show you data structure usage and placement

Linux OpenPower (LE) support

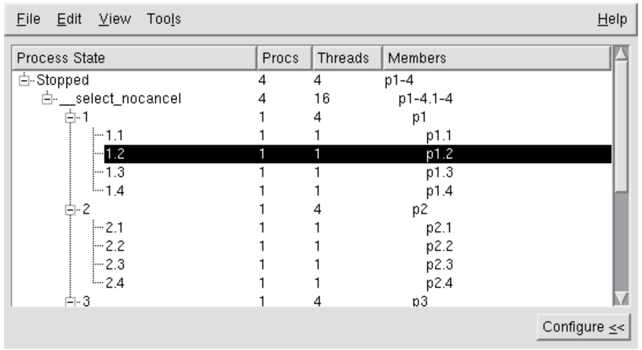
- Experimental support for OpenPower (Linux power LE)
  - All major functionality
  - Support for CUDA Debugging on GPU Accelerators
  - Contact Nikolay.Piskun@roguewave.com

Current Work
and
Future Plans

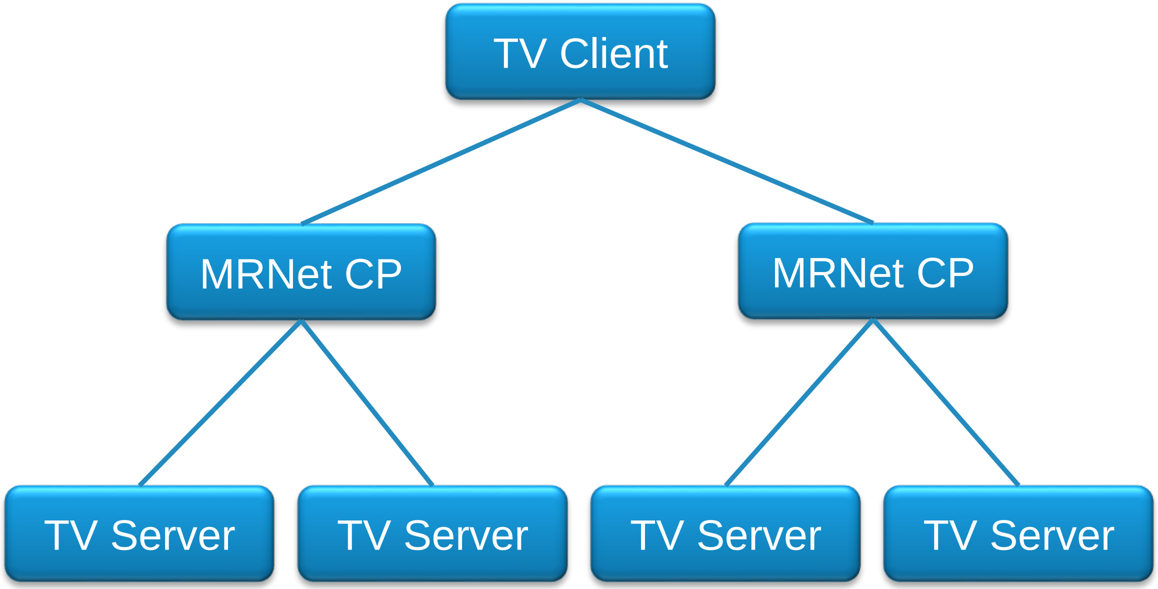
TotalView 8.15

New Features

- Scalable Infrastructure
  - Faster start up on Linux
  - Scales to O(100,000) processes & O(1,000,000) threads
- Updated CUDA support
  - CUDA 7.0
- Support updates including:
  - Clang 3.5
  - Intel 15.0
  - MPT 2.12
  - SLES 12, Fedora 21



Process State	Procs	Threads	Members
Stopped	4	4	p1-4
__select_nocancel	4	16	p1-4.1-4
1	1	4	p1
1.1	1	1	p1.1
1.2	1	1	p1.2
1.3	1	1	p1.3
1.4	1	1	p1.4
2	1	4	p2
2.1	1	1	p2.1
2.2	1	1	p2.2
2.3	1	1	p2.3
2.4	1	1	p2.4
3	1	4	p3

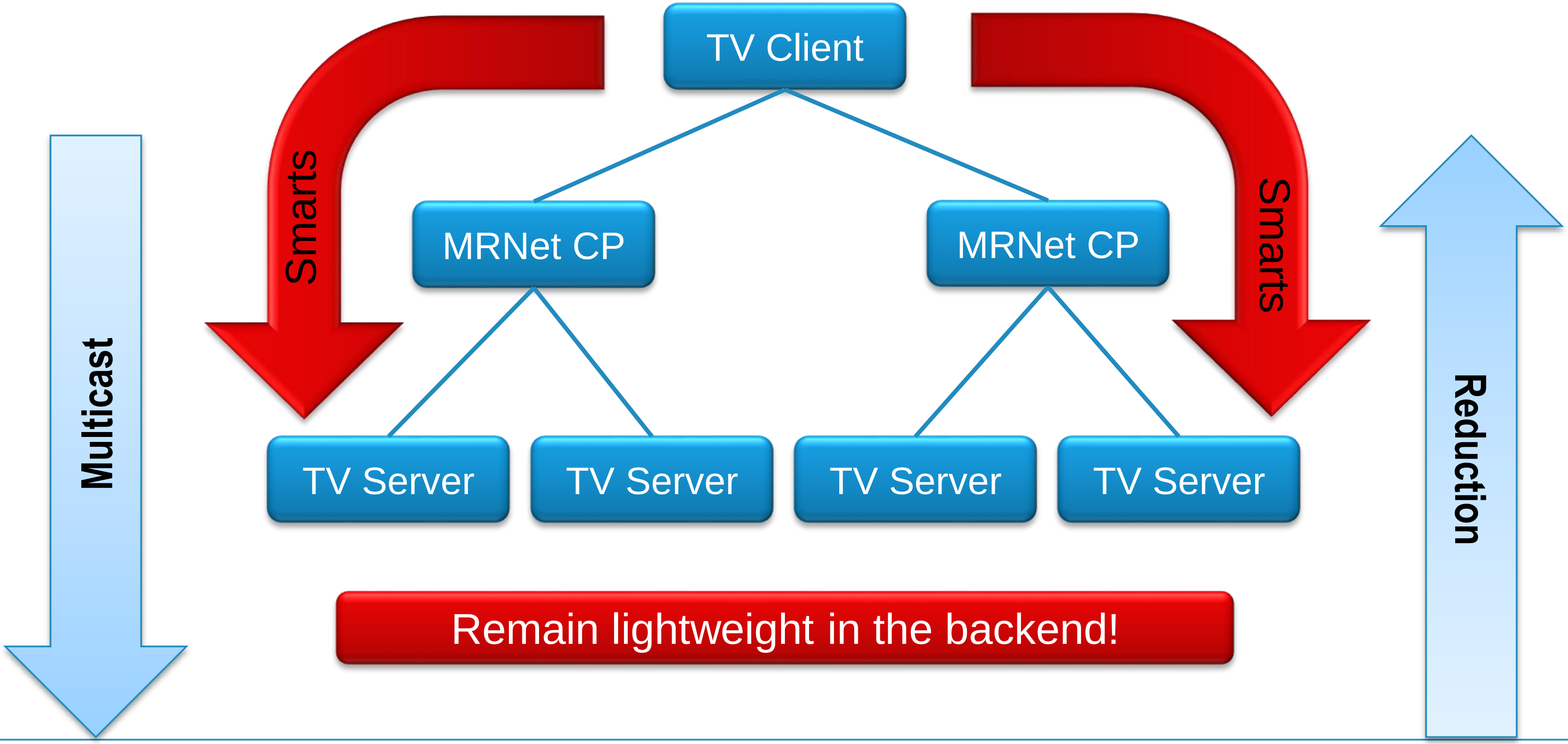


TotalView’s Scalability Strategy

Push debugger “smarts” to the backend, not the whole debugger!

TotalView uses an “MRNet tree” of servers

Use classic optimization techniques too: caching, hoisting invariants, etc.



Remain lightweight in the backend!

TotalView’s Memory Efficiency

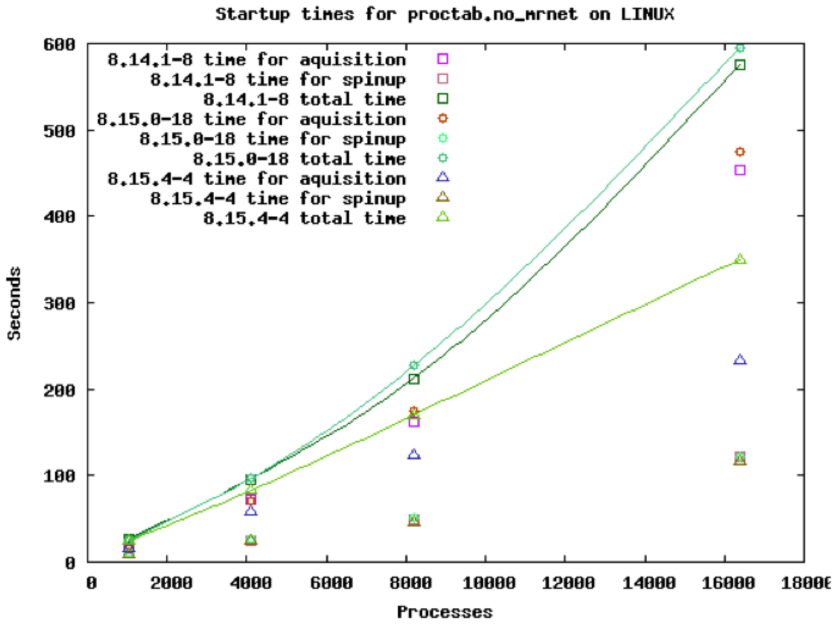
- TotalView is lightweight in the back-end (server)
- Servers don’t “steal” memory from the application
- Each server is a multi-process debugger agent
  - One server can debug thousands of processes
  - Not a conglomeration of single process debuggers
  - TotalView’s architecture provides flexibility (e.g., P/SVR)
  - No artificial limits to accommodate the debugger (e.g., BG/Q ~~1 P/CN~~)
- Symbols are read, stored, and <u>shared</u> in the front-end (client)
- Example: LLNL APP ADB, 920 shlibs, Linux, 64 P, 4 CN, 16 P/CN, 1 SVR/CN

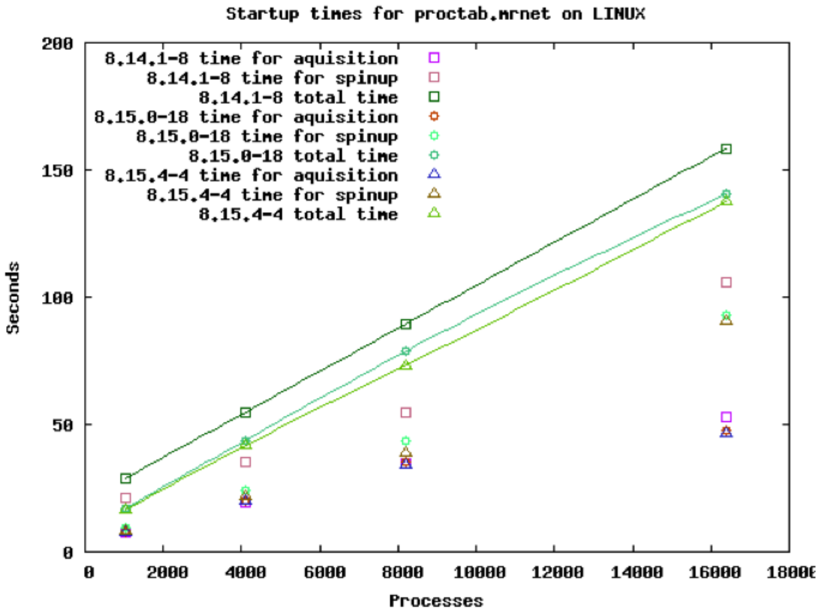
Process	VSZ (largest, MB)	RSS (largest, MB)
TV Client	4,469	3,998
MRNet CP	497	4
TV Server	304	53

Linux Start up Performance in TV 8.15.4

5x faster at 16k between 8.14.1 and 8.15.4.

Note that we switched to MRNET by default in 8.15.0

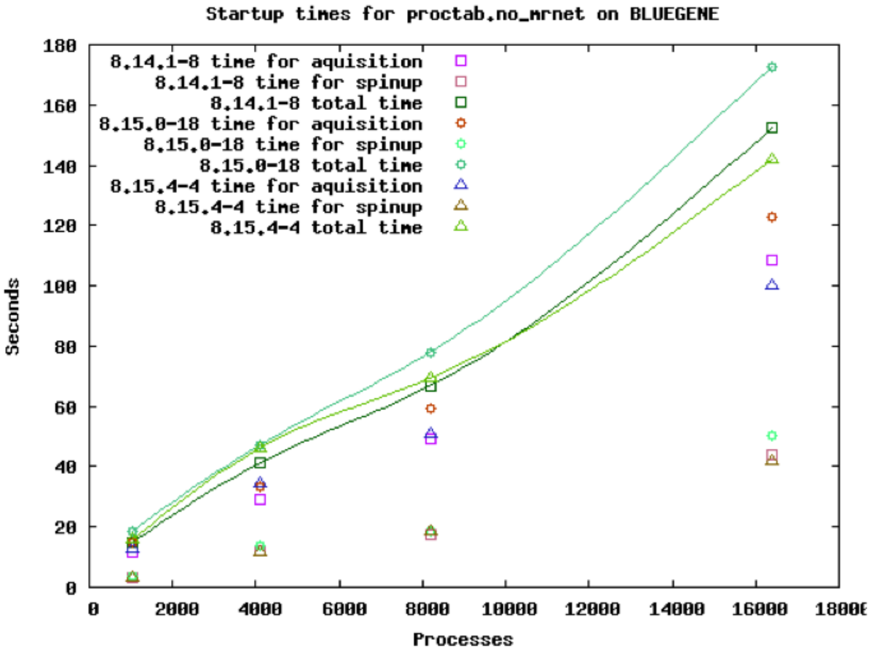


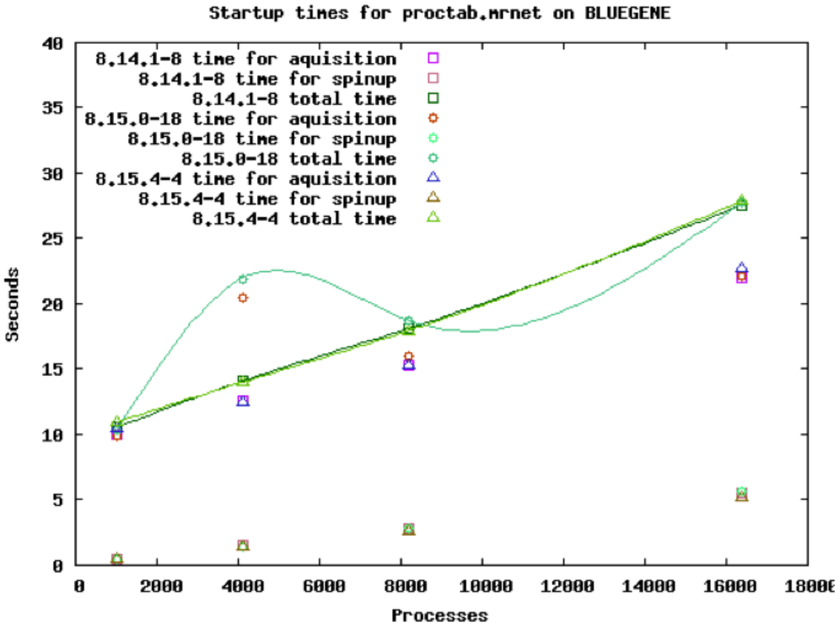


BG Start up Performance in TV 8.15.4

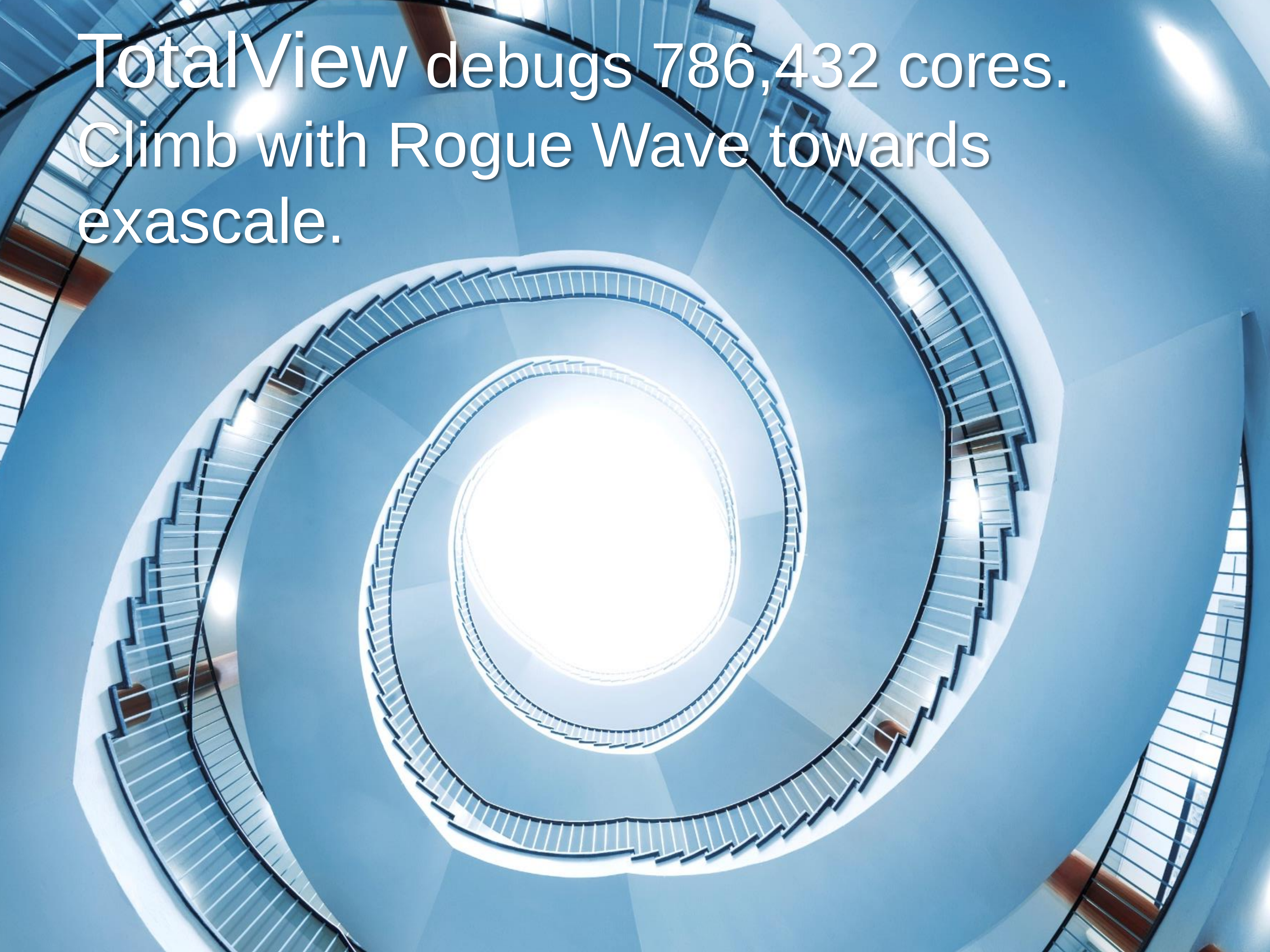
6.4x faster at 16k between 8.14.1 and 8.15.4.

Note that we switched to MRNET by default in 8.15.0

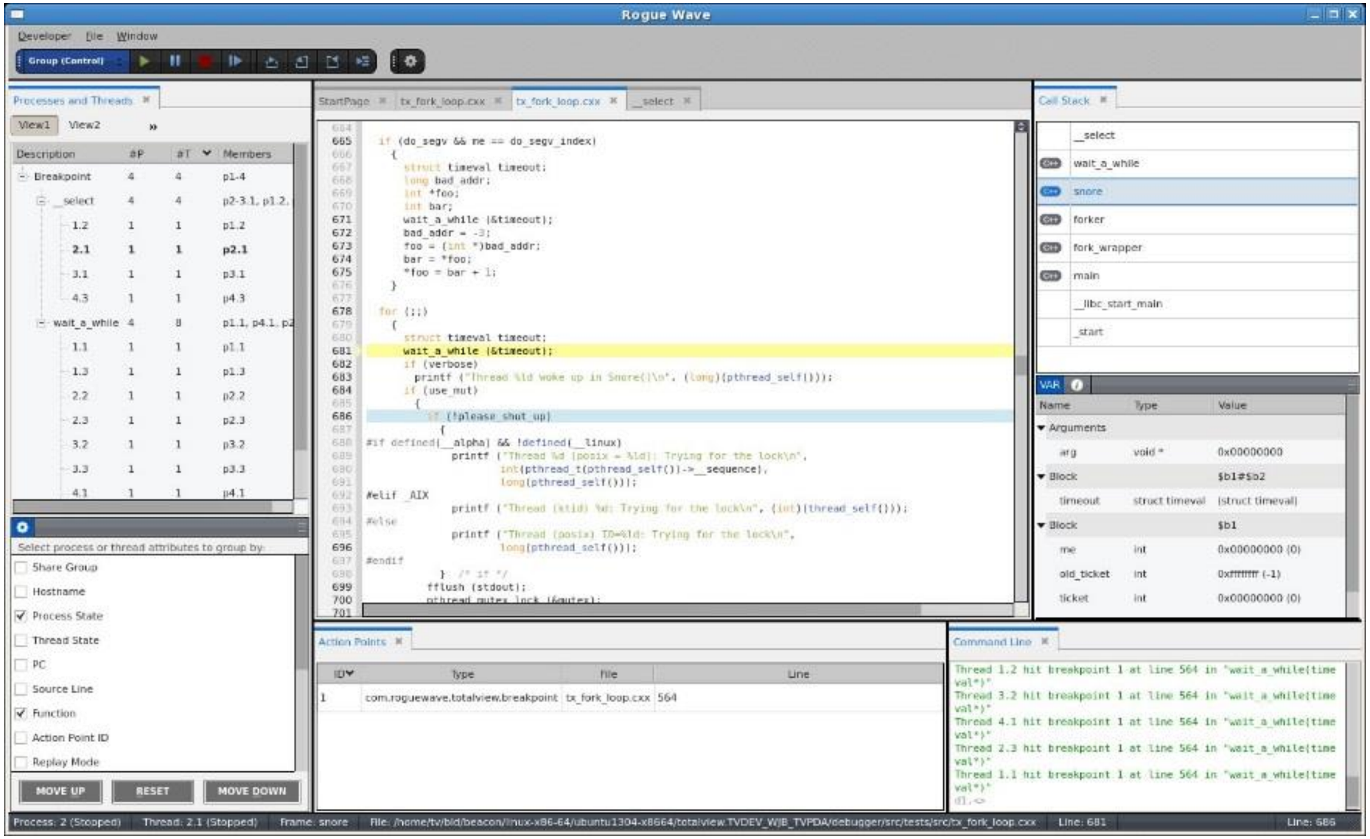




TotalView debugs 786,432 cores. Climb with Rogue Wave towards exascale.



New Revolutionary UI Framework



Description	#P	#T	Members
Breakpoint	4	4	p1-4
_select	4	4	p2-3.1, p1.2,
1.2	1	1	p1.2
2.1	**1**	**1**	**p2.1**
3.1	1	1	p3.1
4.3	1	1	p4.3
wait_a_while	4	8	p1.1, p4.1, p2
1.1	1	1	p1.1
1.3	1	1	p1.3
2.2	1	1	p2.2
2.3	1	1	p2.3
3.2	1	1	p3.2
3.3	1	1	p3.3
4.1	1	1	p4.1

Select process or thread attributes to group by:

□ Share Group
□ Hostname
☑ Process State
□ Thread State
□ PC
□ Source Line
☑ Function
□ Action Point ID
□ Replay Mode

```
  if (do_segv && me == do_segv_index)
    {
      struct timeval timeout;
      long bad_addr;
      int *foo;
      int bar;
      wait_a_while (&timeout);
      bad_addr = -3;
      foo = (int *)bad_addr;
      bar = *foo;
      *foo = bar + 1;
    }

  for (;;)
    {
      struct timeval timeout;
      wait_a_while (&timeout);
      if (verbose)
        printf ("Thread %ld woke up in Snore()\n", (long)(pthread_self()));
      if (use_mut)
        {
          if (!please_shut_up)
            {
#if defined(__alpha) && !defined(__linux)
              printf ("Thread %d (posix = %ld): Trying for the lock\n",
                      int(pthread_t(pthread_self())->__sequence),
                      long(pthread_self()));
#elif _AIX
              printf ("Thread (ktid) %d: Trying for the lock\n", (int)(thread_self()));
#else
              printf ("Thread (posix) ID=%ld: Trying for the lock\n",
                      long(pthread_self()));
#endif
            } /* if */
          fflush (stdout);
          pthread_mutex_lock (&mutex);
```

Call Stack:

- _select
- C++ wait_a_while
- C++ snore
- C++ forker
- C++ fork_wrapper
- C++ main
- __libc_start_main
- _start

Name	Type	Value
▼ Arguments		
arg	void *	0x00000000
▼ Block		$b1#$b2
timeout	struct timeval	(struct timeval)
▼ Block		$b1
me	int	0x00000000 (0)
old_ticket	int	0xffffffff (-1)
ticket	int	0x00000000 (0)

ID	Type	File	Line
1	com.roguewave.totalview.breakpoint	tx_fork_loop.cxx	564

```
Thread 1.2 hit breakpoint 1 at line 564 in "wait_a_while(time
val*)"
Thread 3.2 hit breakpoint 1 at line 564 in "wait_a_while(time
val*)"
Thread 4.1 hit breakpoint 1 at line 564 in "wait_a_while(time
val*)"
Thread 2.3 hit breakpoint 1 at line 564 in "wait_a_while(time
val*)"
Thread 1.1 hit breakpoint 1 at line 564 in "wait_a_while(time
val*)"
d1.<>
```

Process: 2 (Stopped)   Thread: 2.1 (Stopped)   Frame: snore   File: /home/tv/bld/beacon/linux-x86-64/ubuntu1304-x8664/totalview.TVDEV_WJB_TVPDA/debugger/src/tests/src/tx_fork_loop.cxx   Line: 681   Line: 686

- Design your own Debugger/Tool
- Personalize to your likeness.
- In Evergreen Beta soon

More Information

- Product documentation on website:

  http://www.roguewave.com/help-support/documentation/totalview

- Contact sales@roguewave.com with any inquires about our future plans with regard to TotalView product.

Questions

Thanks!

- Visit the website
  - http://www.roguewave.com/products/totalview.aspx
  - Documentation
  - Sign up for an evaluation
  - Contact customer support & post on the user forum