



# ATPESC 2016

## TotalView: Debugging from Desktop to Supercomputer

Peter Thompson  
Principal Software Support Engineer  
August 10, 2016



# Our products and services



## Tools

**Klocwork** On-the-fly static code analysis for app security

**CodeDynamics** Commercial dynamic analysis

**OpenLogic Support** Enterprise-grade SLA support

**OpenLogic Audits** Detailed open source license and security risk guidance

**TotalView for HPC** Scalable debugging

**Zend Server** Enterprise PHP app server

**Zend Studio** PHP IDE

**Zend Guard** PHP encoding and obfuscation



## Libraries

**SourcePro** OS, database, network, and analysis abstraction for C++

**Visualization** Real-time data visualization at scale

**PV-WAVE** Visual data analysis

**IMSL Numerical Libraries** Scalable math and statistics algorithms

**HydraExpress** SOA/C++ modernization framework

**HostAccess** Terminal emulation for Windows

**Stingray** MFC GUI components

# How does Rogue Wave help in HPC?

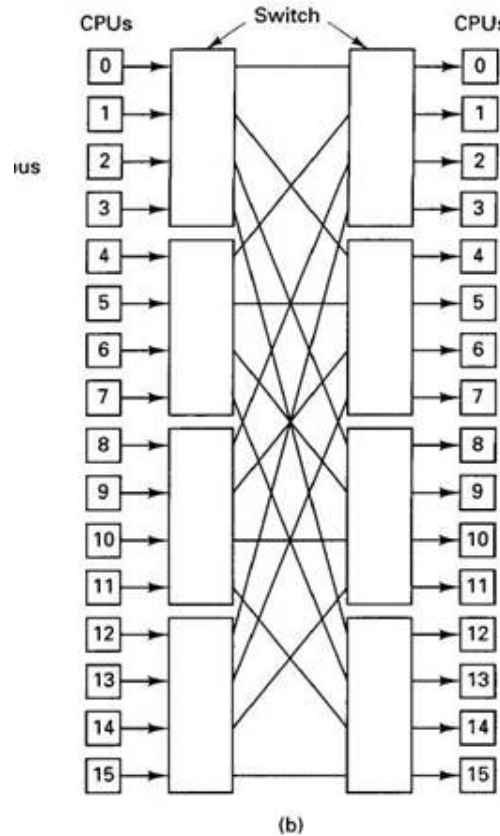
---

## TotalView for HPC

- Source code debugger for C/C++/Fortran
  - Visibility into applications
  - Control over applications
- Scalability
- Usability
- Support for HPC platforms and languages

# 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
  - Multiple and Mixed Language Support
- Core development group has been there from the beginning and have been/are involved in defining MPI interfaces, DWARF, and lately OMPD (Open MP debugging interface)

# Other capabilities added

---

- Support for most types of MPI
- Linux
- Lightweight Memory Debugging
- Type transformations – STL and user containers
- Memscript and tvscript
- Reverse Debugging - only on Linux x86-64
- Remote Display Client
- GPU debugging
- Intel Xeon Phi – Including KNL
- Currently looking at ARM64 port

# Key features of TotalView

---

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

Serial, Parallel and Accelerated applications



# Memory Analysis

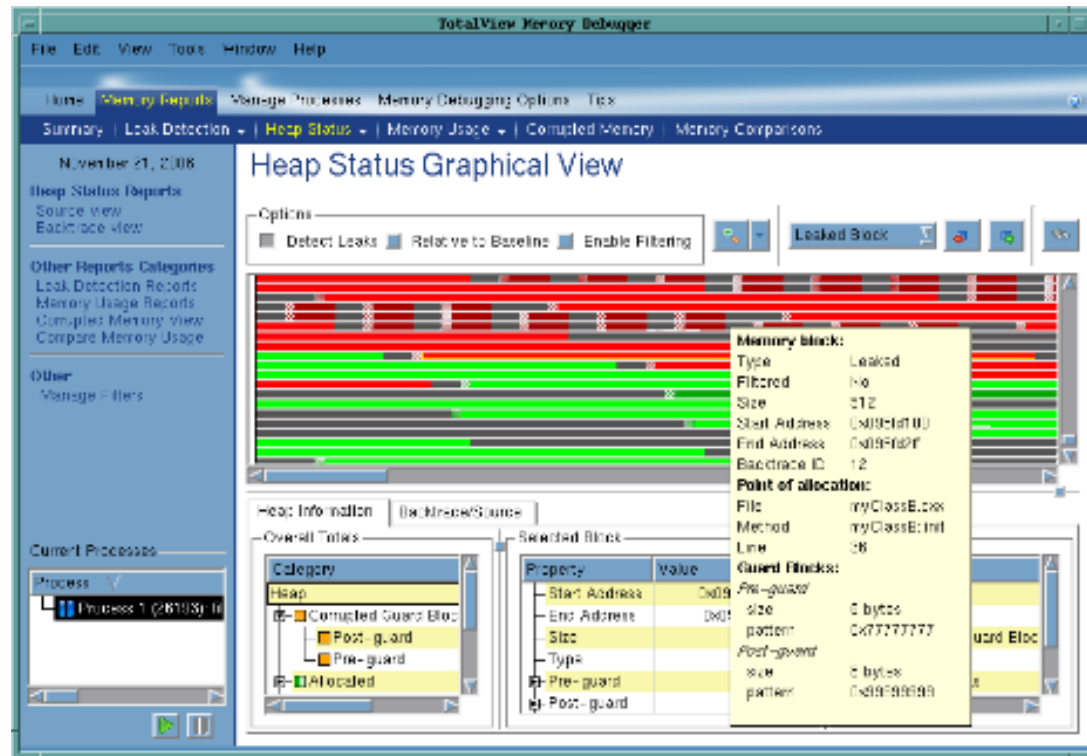
How do you identify buffer overflows?

## Runtime Memory Analysis : Eliminate Memory Errors

- Detects memory leaks *before* they are a problem
- Explore heap memory usage

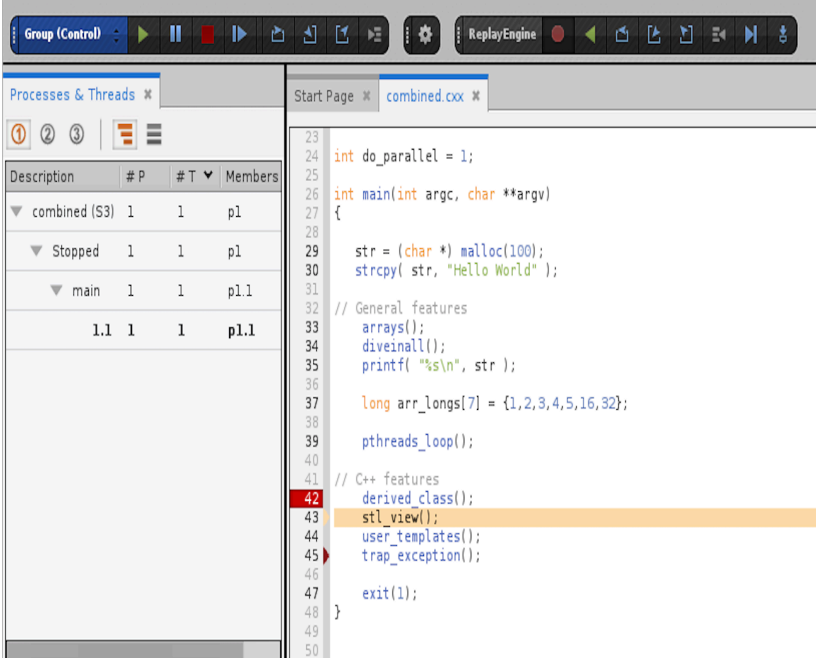
## Features

- Detects
  - Malloc API misuse
  - Memory leaks
  - Buffer overflows
- Low runtime overhead
- Easy to use
  - Works with vendor libraries
  - No recompilation
  - No instrumentation



# Reverse debugging

- How do you isolate an intermittent failure?
  - Without TotalView
    - Set a breakpoint in code
    - Realize you ran past the problem
    - Re-load
    - Set breakpoint earlier
    - Hope it fails
    - Keep repeating
  - With TotalView
    - Start recording
    - Set a breakpoint
    - See failure
    - Run backwards/forwards in context of failing execution
  - Reverse Debugging
    - Re-creates the context when going backwards
    - Focus down to a specific problem area easily
    - Saves days in recreating a failure



```
23
24 int do_parallel = 1;
25
26 int main(int argc, char **argv)
27 {
28
29     str = (char *) malloc(100);
30     strcpy( str, "Hello World" );
31
32     // General features
33     arrays();
34     diveinall();
35     printf( "%s\n", str );
36
37     long arr longs[7] = {1,2,3,4,5,16,32};
38
39     pthreads_loop();
40
41     // C++ features
42     derived class();
43     stl_view();
44     user_templates();
45     trap_exception();
46
47     exit(1);
48 }
49
50
```

# memscript and tvscript

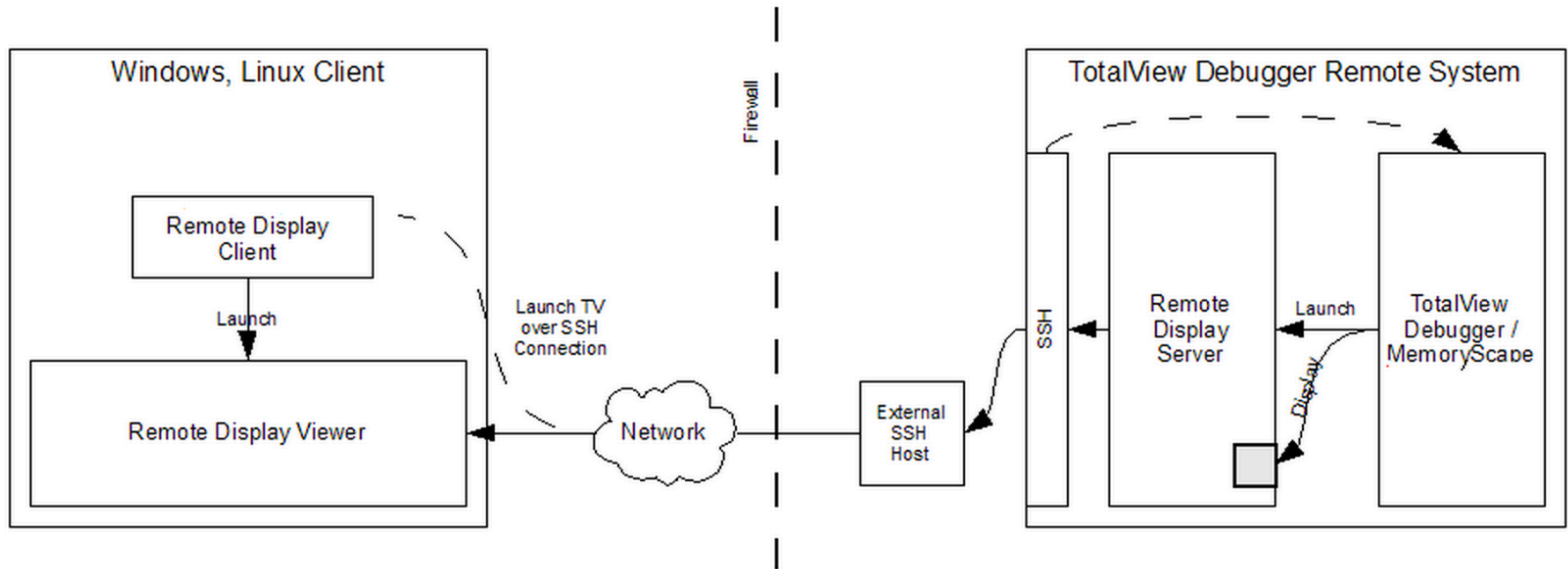
- Command line invocation to run TotalView and Memoryscape unattended
- tvscript can be used to set breakpoints, take actions at those breakpoints and have the results logged to a file. It can also do memory debugging
  - `tvscript -create_actionpoint "method1=>display_backtrace show_arguments" \ -create_actionpoint "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


# 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:  User Name
- 2. As needed, enter hosts in access order to reach the Remote Host:**

	Host	Access By	Access Value	Commands
<input type="button" value="↑"/>	1	<input type="text" value="User Name"/>		
<input type="button" value="↓"/>	2	<input type="text" value="User Name"/>		
- 3. Enter settings for the debug session on the Remote Host :**

Path to TotalView on Remote Host:

Arguments for TotalView:

Your Executable (path & name):

Arguments for Your Executable:

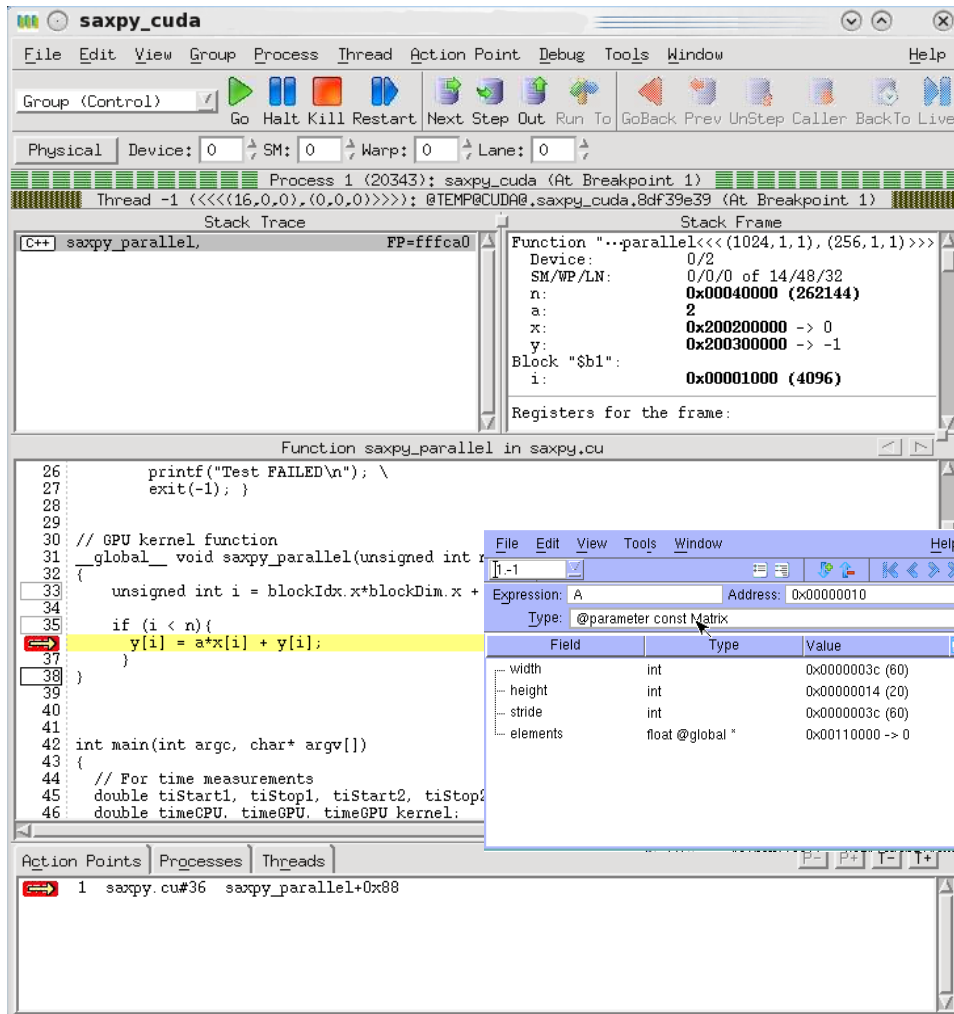
Submit Job to Batch Queueing System:
- 4. Enter batch submission settings for the Remote Host :**

Submit Command:

Script to execute via Submit Command:

Additional Submit Command Options:

# TotalView for the NVIDIA® GPU Accelerator



- NVIDIA CUDA 6.5, 7.0, 7.5, (8.0)
- 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

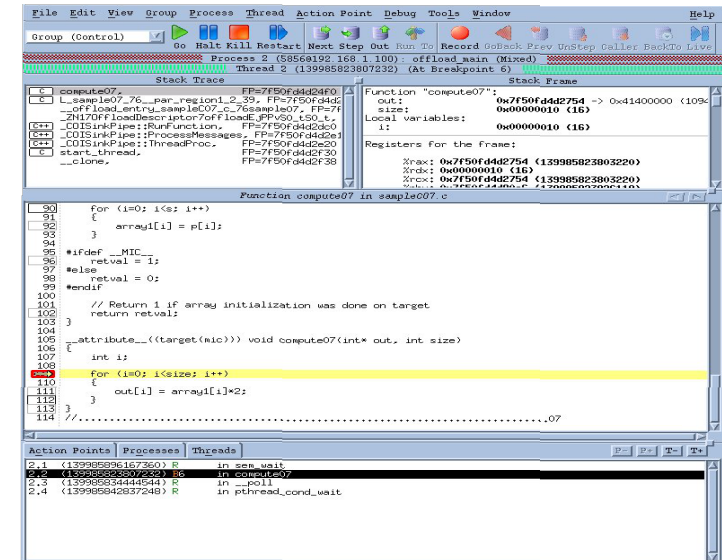
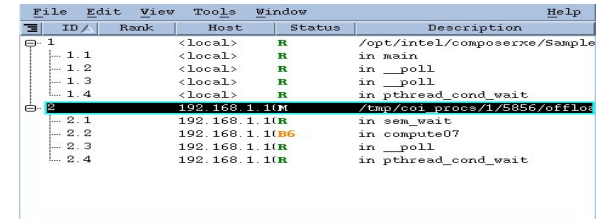
- Native Mode
  - With or without MPI
- Offload Directives
  - Incremental adoption, similar to GPU
- Symmetric Mode
  - Host and Coprocessor
- **Multi-device, Multi-node**
- Clusters
- KNL Support – Just works like a normal node
  - AVX2 support being added

## 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



# 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
- KNL machines starting to come online



# Linux OpenPower (LE) support

---

- Support for OpenPower (Linux power LE)
  - All major functionality
  - Support for CUDA Debugging on GPU Accelerators

# 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  $\neq$  P/CN)
- Symbols are read, stored, and shared 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

# New UI Framework – aka CodeDynamics

The screenshot displays the Rogue Wave IDE interface with the following components:

- Processes and Threads:** A tree view showing a breakpoint at line 564 in `wait_a_while` for process `p2.1`.
- Code Editor:** C++ source code for `tx_fork_loop.cxx` with a breakpoint at line 564. The code includes a `wait_a_while` function and a `for` loop.
- Call Stack:** A list of stack frames including `_select`, `wait_a_while`, `snore`, `forker`, `fork_wrapper`, `main`, `_libc_start_main`, and `_start`.
- Variables:** A table showing the state of variables in the current frame:

Name	Type	Value
Arguments		
arg	void *	0x00000000
Block		
timeout	struct timeval	{struct timeval}
Block		
me	int	0x00000000 (0)
old_ticket	int	0xffffffff (-1)
ticket	int	0x00000000 (0)
- Action Points:** A table showing a single action point:

ID	Type	File	Line
1	com.roguewave.totalview.breakpoint	tx_fork_loop.cxx	564
- Command Line:** A log of messages indicating that threads 1.2, 3.2, 4.1, 2.3, and 1.1 have hit the breakpoint at line 564 in the `wait_a_while` function.

Process: 2 (Stopped) Thread: 2.1 (Stopped) Frame: snore File: /home/tybid/beacon/linux-x86-64/ubuntu1304-x8664/totalview\_TVDEV\_WjB\_TVPA/buggers/src/tests/src/cxx\_fork\_loop.cxx Line: 681 Line: 686

# Using TotalView

---

For HPC we have two methods to start the debugger

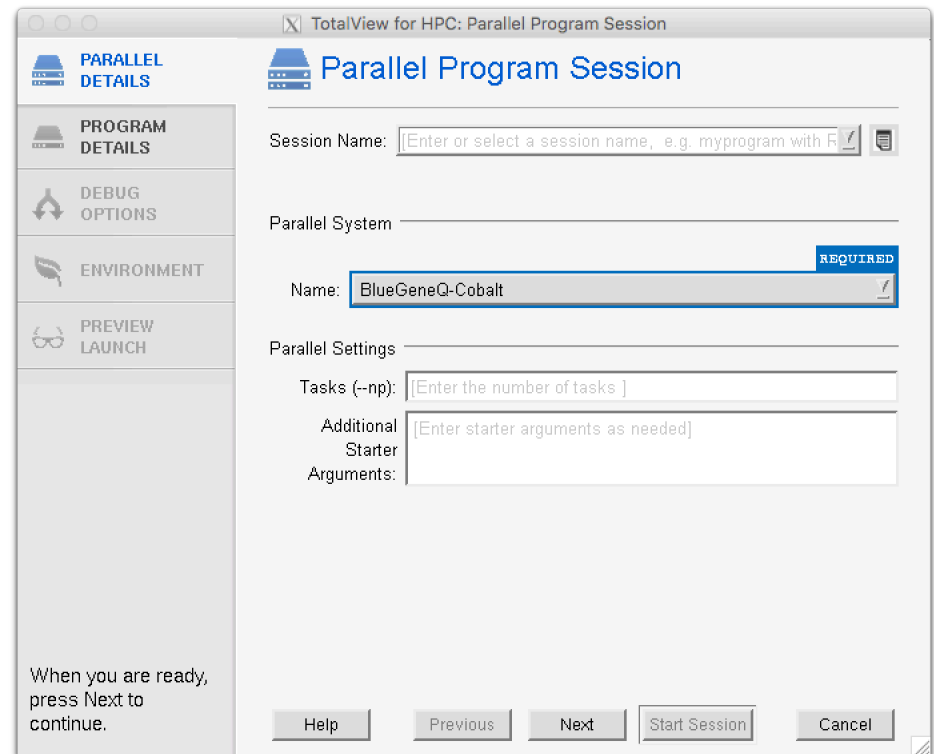
The ‘classic’ method

- `totalview –args mpiexec –np 512 ./myMPIprog myarg1 myarg2`
- This will start up TotalView on the parallel starter (`mpiexec`, `srun`, `runjob`, etc) and when you hit ‘Go’ the job will start up and the processes will be automatically attached. At that point you will see your source and can set breakpoints.
- Some points to consider...
  - You don’t see your source at first, since we’re ‘debugging’ the mpi starter
  - Some MPI’s don’t support the process acquisition method (most do, but might be stripped of symbols we need when packaging)
  - In general more scalable than the next method...

# Starting TotalView

The ‘indirect’ method

- Simply ‘totalview’ or ‘totalview myMPIprog’ and then you can choose a parallel system, number of tasks, nodes, and arguments to the program.
- With this method the program source is available immediately
- Less dependent on MPI starter symbols
- May not be as scalable as some ‘indirect’ methods launch a debug server per process



# The New UI for HPC

MPI debugging with the new UI requires starting in 'classic' mode with the –newUI argument

totalview –newUI –args mpiexec –np 4 ./cpi

The screenshot displays the TotalView for HPC 2016 interface. The main window shows the source code for a parallel pi calculation. The right sidebar contains several panels: 'Processes & Threads' showing a table of process information, 'Call Stack' showing the current call stack, and 'Data View' showing the contents of a struct field named 'myfield'. The bottom left panel shows the 'Command Line' window with a log of thread activity. The bottom right panel shows the 'Data View' window displaying the contents of a struct field named 'myfield'.

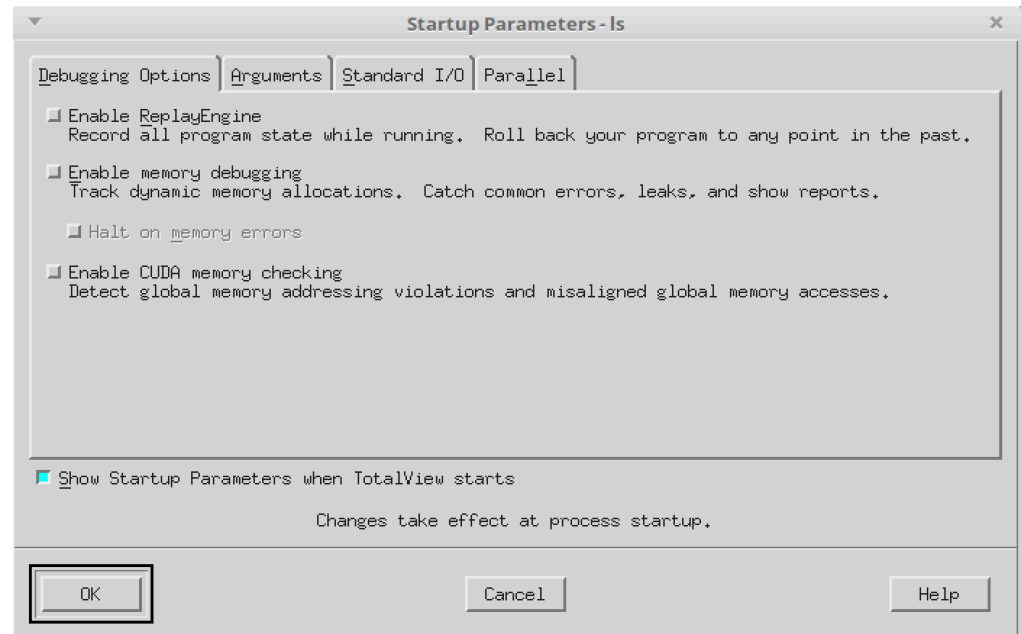
Description	# P	# T	Members
__poll_...	12	12	0-11.2
2.2	1	1	0.2
3.2	1	1	1.2
4.2	1	1	2.2
5.2	1	1	3.2
6.2	1	1	4.2

Name	Type	Value
myfield	str...	(struct Field)
x	float	1
y	float	2
value	do...	3.14159265358979
myfield.value	do...	3.14159265358979

```
Thread 12.2 has exited
Thread 13.2 has appeared
Thread 13.1 has appeared
Thread 13.1 has exited
Thread 13.2 has exited
Thread 13.1 hit breakpoint 1 at line 35 in "main"
Thread 12.1 hit breakpoint 1 at line 35 in "main"
Thread 11.1 hit breakpoint 1 at line 35 in "main"
Thread 10.1 hit breakpoint 1 at line 35 in "main"
Thread 8.1 hit breakpoint 1 at line 35 in "main"
Thread 9.1 hit breakpoint 1 at line 35 in "main"
Thread 7.1 hit breakpoint 1 at line 35 in "main"
Thread 6.1 hit breakpoint 1 at line 35 in "main"
Thread 5.1 hit breakpoint 1 at line 35 in "main"
Thread 4.1 hit breakpoint 1 at line 35 in "main"
Thread 3.1 hit breakpoint 1 at line 35 in "main"
Thread 2.1 hit breakpoint 1 at line 35 in "main"
Thread 1.1 stopped: Stop Signal
Thread 1.2 stopped: Stop Signal
Thread 1.1 stopped: Stop Signal
```

# Starting MemoryScape and ReplayEngine

- MemoryScape must be enabled at the start of the program or your program can be linked against the Heap Interposition Agent (HIA) to have memory debugging always enabled. It can't be turned on when the program has been started.
- ReplayEngine can be enabled at the start, or at any point during debugging. Once started, it can't be disabled.



# Running TotalView on Vesta

The screenshot displays the TotalView for HPC 2016.06.21 interface, showing the process state, stack trace, and MPI\_COMM\_WORLD ranks.

**Process State:**

Process State	Procs	Threads	Members
Running	1	1	p1
Unknown address	1	2	p1.1-2
Unknown address	1	2	p1.1-2
No Action	1	2	p1.1-2
Not Stopped	1	2	p1.1-2
Breakpoint	32	32	0-31
code	32	32	0-31.1
ALLc2.c#142	32	32	0-31.1
Break (2)	32	32	0-31.1
Trace Trap(5)	32	32	0-31.1

**Stack Trace:**

Function	Address
code	FF1dbffffb90
main	FF1dbffffba0
generic_start_main	FF1dbffffb90
__libc_start_main	FF1dbffffbe0

**Function code in ALLc2.c:**

```
130 pid, MPI_ANY_TAG, MPI_COMM_WORLD, &request[k]);
131
132
133 /* send everyone's message back */
134
135 while ((pid = (pid+1) % nnodes) != mypid)
136 {
137     k=k+1;
138     MPI_Isend (&receive_message[pid*size+0], size, MPI_INT,
139              pid, message_type[pid], MPI_COMM_WORLD, &request[k]);
140 }
141
142 if (k != -1)
143     MPI_Recv (&request[0], &status_array[0]); /*
144     k: 0x00000030 (61)
145     MPI_Wait(&request[0], &status_array[0]);
146
147 /* check my returned messages */
148
149 while ((pid = (pid+1) % nnodes) != mypid)
150 {
151     /* Verify to see if returned message is the same as the message sent */
152
```

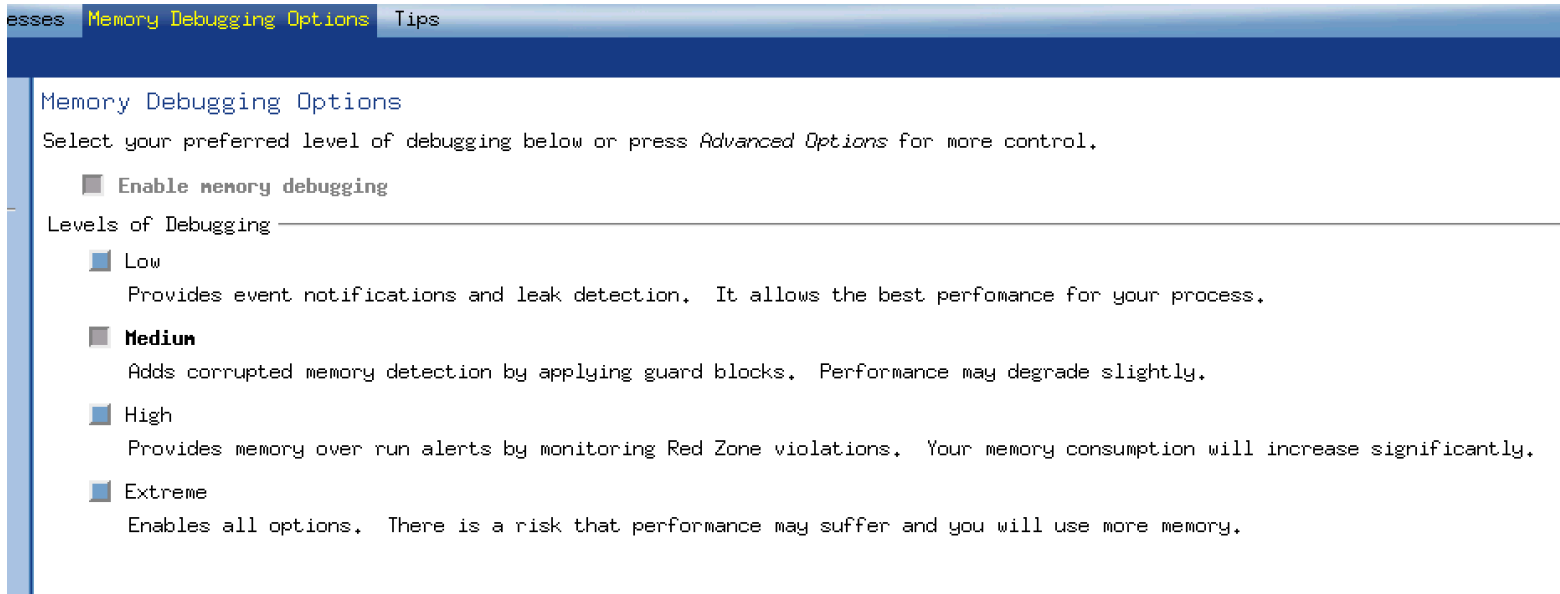
**MPI\_COMM\_WORLD Ranks:**

Rank	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
pid	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



# Memory Debugging on BG/Q

- In order to use memory debugging on the BG/Q, you must link against the agent. For instance, I first set up a environment variable
  - export TVLIB=/soft/debugger/totalview/linux-power/lib
  - mpixlc\_r -g -o ALLc2.mem ALLc2.c -L\$TVLIB \
  - -Wl,@\$TVLIB/tvheap\_bgqs.ld



# One of many reports available

MemoryScape 2016.06.21

File Tools Window Help

Home Memory Reports Manage Processes Memory Debugging Options Tips

Summary Leak Detection Heap Status Memory Usage Corrupted Memory Memory Comparisons

August 10, 2016

Save Data  
Save Report...  
Export Memory Data...

Heap Status Reports  
Graphical Report  
Source Report

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

Other Tasks  
Manage Filters

Process Selection

Process	Total Bytes	Count	Function	Line #	Source Information
215	376	24			
1179	840	21			
1126	400	10			
190	80,00KB	10			
188	80,00KB	10			
184	1280,00KB	10			
167	640,00KB	10			
1176	178,00KB	8			
			__wrap_posix_realign	60	tv_heap_breakpoint.c
			PAMI::Memory::HeapMemoryManager::...	119	HeapMemoryManager.h
			PAMI::MemoryAllocator(5680,16,4,...	167	MemoryAllocator.h
			PAMI::Protocol::Send::Eager(PAMI...	111	MemoryAllocator.h
			PAMI_Send	380	Dispatch.h
			MPIDI_SendMsg_eager	129	mpidi_sendmsg.c
			MPIDI_Isend_handoff	462	mpidi_sendmsg.c
			PMPI_Isend	146	mpidi_macros.h
			MPI_Isend	102	mpiwrappers_c.c
			code	115	ALLc2.c
			main	225	ALLc2.c
			generic_start_main	226	libc-start.c
			__libc_start_main	194	libc-start.c

Source

```
*/  
58  
59 TV_HEAP_event = *event;  
60 } /* TV_HEAP_notify_breakpoint_here () */  
61  
62
```

# More Information

---

- Product documentation on website:

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

- Contact [sales@roguewave.com](mailto:sales@roguewave.com) with any inquiries 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