TotalView Training

Developing parallel, data-intensive applications is hard. **We make it easier.**
Agenda

• Introduction
• Startup
• Remote Display Debugging
• UI Navigation and Process Control
• Action Points
• Data Monitoring and Visualization
Agenda

• Debugging for Parallel Applications
• Memory Debugging with MemoryScape
• Batch Debugging
• Reverse Debugging with ReplayEngine
• What’s New in TotalView
• Questions
INTRODUCTION
What is TotalView?

A comprehensive debugging solution for demanding parallel and multi-core applications

- Wide compiler & platform support
  - C, C++, Fortran 77 & 90
  - Unix, Linux, OS X
- Handles Concurrency
  - Multi-threaded Debugging
  - Parallel Debugging
  - CUDA
  - Intel Xeon Phi
- Integrated Memory Debugging
  - MemoryScape
- Reverse Debugging
  - ReplayEngine
- Supports a Variety of Usage Models
  - Powerful and Easy GUI
  - CLI for Scripting
  - Remote Debugging
  - Unattended Batch Debugging

©Copyright 2012 Rogue Wave Software, Inc.
Supported Compilers and Architectures

• **Platform Support**
  • Linux x86, x86-64, ia64, Power
  • Cray XT, XE, XK
  • Solaris Sparc and AMD64
  • AIX
  • IBM BGL, BGP
  • Mac Intel
  • Cell

• **Languages / Compilers**
  • C/C++, Fortran, UPC, Assembly
  • Many Commercial & Open Source Compilers

• **Parallel Environments**
  • MPI
    • MPICH1& 2, Open MPI, Intel MPI, SGI MPT & Propack, SLURM, poe, MPT, Quadrics, MVAPICH1 & 2, Bullx MPI, & many others
  • UPC
STARTUP
Starting TotalView

Start New Process

The New Program window allows you to start a new process with various options.

- **Program:** Enter the program name or browse for it.
- **On host:** Specify the host name (local) or browse for a different host.
- **Enable ReplayEngine:** Record all program state while running. Roll back your program to any point in the past.
- **Enable memory debugging:** Track dynamic memory allocations. Catch common errors, leaks, and show reports.
  - **Halt on memory errors:** Automatically halt the program on memory errors.
- **Enable CUDA memory checking:** Detect global memory addressing violations and misaligned global memory accesses.

Use the buttons at the bottom to **OK**, **Cancel**, or **Help**.
Start New Process – Select a recent process
Starting TotalView

Start New Process – Arguments tab
Starting TotalView

Start New Process – Command-line Args

![New Program dialog box with command-line arguments set to "Hello World"]
Starting TotalView

Start New Process – set environment variables

![Image of TotalView interface showing command-line arguments and environment variables]

- Command-line arguments: `Hello World`
- Environment variables: `EXE_HOME_DIR=/home/johnh/myapp`
Starting TotalView

Start New Process – Standard I/O redirection
Attach to Process

![New Program dialog box](image)

Program: /usr/bin/dbus-launch
On host: (local)
PID: 3295

Select processes to attach to:

<table>
<thead>
<tr>
<th>Program</th>
<th>Host</th>
<th>Local Path</th>
<th>State</th>
<th>PID</th>
<th>PPID</th>
</tr>
</thead>
<tbody>
<tr>
<td>VBoxClient</td>
<td>10.0.2.15</td>
<td>/GuestAdditions-4.1.8/bin/</td>
<td>S</td>
<td>3240</td>
<td></td>
</tr>
<tr>
<td>dbus-launch</td>
<td>10.0.2.15</td>
<td>/usr/bin/</td>
<td>S</td>
<td>3295</td>
<td></td>
</tr>
<tr>
<td>dbus-daemon</td>
<td>10.0.2.15</td>
<td>/bin/</td>
<td>S</td>
<td>3296</td>
<td>1</td>
</tr>
<tr>
<td>gconfd-2</td>
<td>10.0.2.15</td>
<td>/usr/libexec/</td>
<td>S</td>
<td>3302</td>
<td>1</td>
</tr>
</tbody>
</table>
Starting TotalView

Open a Core File

[Image of a dialog box for starting TotalView with a program name set to `/home/johnh/threaded/simple_threaded/simple`, host set to `local`, and a core file set to `core`.]

©Copyright 2012 Rogue Wave Software, Inc.
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 ]
REMOTE DISPLAY DEBUGGING
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
Remote Display Client

- Free to install on as many clients as needed
- No license required to run the client
  - Only the server running TotalView requires licenses. Must be version 8.6 or later of TotalView or version 2.4 or later of MemoryScape.
- Presents a local window that displays TotalView or MemoryScape running on the remote machine
- Requires SSH and X Windows on Server
Remote Display Client

- User must provide information necessary to connect to remote host
- Passwords are NOT stored
- Information required includes:
  - User name, public key file, other ssh information
  - Directory where TotalView/MemoryScape is located
  - Path and name of executable to be debugged
  - If using indirect connection with host jump, each host
    - Host name
    - Access type (User name, public key, other ssh information)
    - Access value
- Client also allows for batch submission via PBS Pro or LoadLeveler
Remote Display Client

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

3. Enter settings for the debug session on the Remote Host:
   - TotalView
     - Path to TotalView on Remote Host: totalview
     - Arguments for TotalView:
     - Your Executable (path & name):
     - Arguments for Your Executable:
     - Submit Job to Batch Queueing System: Not Applicable

No session running

Launch Debug Session
Session Profile Management

• Connection information can be saved as a profile, including all host jumping information
• Multiple profiles can be generated
• Profiles can be exported and shared
• Generated profiles can be imported for use by other users
UI NAVIGATION AND PROCESS CONTROL
Interface Concepts

Root Window

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

†Status Info

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

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

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

A single point of control for the process and other related processes.
### Stack Trace and Stack Frame Panes

| Language | Name          | Frame Pointer     | Local Variables                                                                
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>C++</td>
<td>wait_a_while,</td>
<td>FP=b7efe348</td>
<td>Function &quot;wait_a_while&quot;:</td>
</tr>
<tr>
<td>C++</td>
<td>need_to_wait,</td>
<td>FP=b7efe358</td>
<td>microseconds:              0x0000014 (20)</td>
</tr>
<tr>
<td>C++</td>
<td>random_vector,</td>
<td>FP=b7efe388</td>
<td>Registers for the frame:</td>
</tr>
<tr>
<td>C++</td>
<td>runme,</td>
<td>FP=b7efe3b8</td>
<td>%eax: 0xb7efe3a4 (-1209015388)</td>
</tr>
<tr>
<td></td>
<td>start_thread,</td>
<td>FP=b7efe4a8</td>
<td>%ecx: 0x00000000 (0)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>%edx: 0x09a9523c (162091580)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>%ebx: 0x00913ff4 (9519092)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>%esp: 0xb7efe340 (-1209015488)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>%ebp: 0xb7efe348 (-1209015480)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>%esi: 0x00000000 (0)</td>
</tr>
</tbody>
</table>

- Click to refocus source pane
- Click to modify
- Dive for variable window

©Copyright 2012 Rogue Wave Software, Inc.
Source Code Pane

View as Source - or Assembly - or Both!

```c
#include <mpi.h>

#include //ADD_MPI

void wait_a_while(size_ 

void need_to_wait() 

{ wait_a_while(); 

void wait_a_while(size_ 

    usleep(microseconds); 

}

void random_vector(std 

    size_t count = (size 

    if(count < 100) count 

    for(size_t i=0; i<count; 

    vec.push_back(rand 
```
Tabbed Pane

Action Points Tab
all currently defined action points

Processes Tab
all current processes

Threads Tab:
all current threads, ID’s, Status
Process Status

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

Search Path: Searching Rules

Search for source files in the following order:

- \$(COMPILATION_DIRECTORY)
- \$(EXECUTABLE_PATH)
- \$(EXECUTABLE_DIRECTORY)
- $links($EXECUTABLE_DIRECTORY)
- \$TOTALVIEW_SRC
Preferences - Formatting

Data Type:
- 8-bit Integer
- 16-bit Integer
- 32-bit Integer
- 64-bit Integer
- 128-bit Integer
- Single Float
- Double Float
- Extended Float String

Presentation:
- Automatic

Data Format:
- Decimal
- Scientific
- Automatic

Min Width:
- 1
- 1
- 1

Precision:
- 15
- 15
- 15

Align:
- Left
- Left
- Left

Pad:
- Space
- Space
- Space

Prefix:
- 0x
- 0

Preview:
"166666666.66667"
Finding Functions, Variables, and Source Files

- Functions:
  - `wait(void)`
  - `need_to_wait(void)`
  - `wait_a_while(unsigned int)`
  - `MPI::Op::Init(void '('const void*, void*, int, const MPI::Datatype&, bool)`)`

- Variables:
  - `threads`
Based on PC location

Stepping Commands
Stepping Commands

- Go
- Halt
- Next
- Step
- Out
- Run To
- Next Instruction
- Step Instruction
- Hold
- Release
- Attach Subset...
- Detach
- Custom Groups...
- Restart
- Ctrl+Z

- Process
- Thread
- Action Point
- Click
- Halt
- Next
- Step
- Out
- Run To
- Next Instruction
- Step Instruction
- Hold
- Release
- Create
- Detach
- Ctrl+A

- Process
- Thread
- Action Point
- Click
- Halt
- Next
- Step
- Out
- Run To
- Next Instruction
- Step Instruction
- Hold
- Release
- Create
- Detach
- Ctrl+A

- Process
- Thread
- Action Point
- Click
- Halt
- Next
- Step
- Out
- Run To
- Next Instruction
- Step Instruction
- Hold
- Release
- Create
- Detach
- Ctrl+A

- Process
- Thread
- Action Point
- Click
- Halt
- Next
- Step
- Out
- Run To
- Next Instruction
- Step Instruction
- Hold
- Release
- Create
- Detach
- Ctrl+A
Using Set PC to resume execution at an arbitrary point

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

Barrier Breakpoints

Conditional Breakpoints

Evaluation Breakpoints

Watchpoints
Setting Breakpoints

- Setting action points
  - Single-click line number
- Deleting action points
  - Single-click action point line
- Disabling action points
  - Single-click in Action Points Tab Pane
- Action Points Tab
  - Lists all action points
  - Dive on an action point to focus it in source pane
- 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
Barrier Breakpoints

Used to synchronize a group of threads or processes defined in the action point.

Threads or processes are held at barrier point until all threads or processes in the group arrive.

When all threads or processes arrive the barrier is satisfied and the threads or processes are released.
Conditional Breakpoint

```
if (my_pid == $tid) { $stop; }
```

Location: /home/ehinkel/Source/combined.cxx#505
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
TotalView understands C++ templates and gives you a choice ...
Watchpoints

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

Action Point -> Create Watchpoint…

```
Create Watchpoint
```

```
Enter expression for watchpoint:
p1[16]
```

```
| int malloc,     | PP=bfd5a2b8 |
| malloc,         | PP=bfd5a2d8 |
| corrupt_data,   | PP=bfd5a308 |
| main,           | PP=bfd5a398 |
| _libc_start_main, | PP=bfd5a408 |
| _start,         | PP=bfd5a410 |
```

```
Registers for the frame:
%eax: 0x00000049 (73)
%ecx: 0x00020f28 (134952)
%edx: 0x00000000 (0)
%ebx: 0x008afff4 (9109492)
%esp: 0xbfd5a1f0 (-1076518416)
%ebp: 0xbfd5a2b8 (-1076518216)
%esi: 0x0807c090 (134725776)
%edi: 0x0807c0d8 (134725848)
%0x: 0x00000000 (0)
```
Watchpoints

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

- Can create from right-click on variable in Source pane
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.
The Variable Window

- 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

Editing Variables

- Window contents are updated automatically
- Changed values are highlighted
- “Last Value” column is available
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

<table>
<thead>
<tr>
<th>Expression</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>rank</td>
<td>0x00000000 (0)</td>
</tr>
<tr>
<td>nnnodes</td>
<td>0x0000000a (10)</td>
</tr>
<tr>
<td>numThreads</td>
<td>0x0000000a (10)</td>
</tr>
<tr>
<td>tm</td>
<td>0xf9ac318 (1335542552)</td>
</tr>
<tr>
<td>tm/numThreads</td>
<td>0x07f5e04f (133554255)</td>
</tr>
<tr>
<td>tm/3600/24/365.25</td>
<td>42.3189596167009</td>
</tr>
</tbody>
</table>
Viewing Arrays

Data Arrays

Structure Arrays
## Array Viewer

- **Variable Window select Tools -> Array Viewer**

### View 2 dimensions of data

- Can be an arbitrary slice through a higher dimensional data cube
- Can be strided

<table>
<thead>
<tr>
<th>Expression:</th>
<th>Type: float[8][16]</th>
</tr>
</thead>
</table>

### Modify array slice:

<table>
<thead>
<tr>
<th>Dimension</th>
<th>Start Index</th>
<th>End Index</th>
<th>Stride</th>
</tr>
</thead>
<tbody>
<tr>
<td>Row [i]</td>
<td>0</td>
<td>7</td>
<td>1</td>
</tr>
<tr>
<td>Column [j]</td>
<td>0</td>
<td>15</td>
<td>1</td>
</tr>
</tbody>
</table>

### Format: Automatic

<table>
<thead>
<tr>
<th>Slice: [0:7:1][0:15:1]</th>
</tr>
</thead>
<tbody>
<tr>
<td>[i]: 0</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>6</td>
</tr>
<tr>
<td>7</td>
</tr>
</tbody>
</table>
Slicing Arrays

Slice notation is \([start:end:stride]\)
Filtering Arrays
• Visualize array data using Tools > Visualize from the Variable Window
• Large arrays can be sliced down to a reasonable size first
• Visualize is a standalone program
• Data can be piped out to other visualization tools

• Visualize allows to spin, zoom, etc.
• Data is not updated with Variable Window; You must revisualize
• $visualize()$ is a directive in the expression system, and can be used in evaluation point expressions.
Dive in All will display an element in an array of structures as if it were a simple array.
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
Typecasting a Dynamic Array

```
Expression: array
Slice: [:]
Type: struct compound_t[20]

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>[0]</td>
<td>struct compound_t</td>
<td>(Struct)</td>
</tr>
<tr>
<td></td>
<td>struct basic_t</td>
<td>(Struct)</td>
</tr>
<tr>
<td></td>
<td>float</td>
<td>5</td>
</tr>
<tr>
<td>[1]</td>
<td>struct compound_t</td>
<td>(Struct)</td>
</tr>
<tr>
<td></td>
<td>struct basic_t</td>
<td>(Struct)</td>
</tr>
<tr>
<td></td>
<td>float</td>
<td>5</td>
</tr>
<tr>
<td>[2]</td>
<td>struct compound_t</td>
<td>(Struct)</td>
</tr>
<tr>
<td></td>
<td>struct basic_t</td>
<td>(Struct)</td>
</tr>
<tr>
<td></td>
<td>float</td>
<td>3.99868e-34</td>
</tr>
</tbody>
</table>
```
C++ Class Hierarchies

Variable Window shows class hierarchy using indentation

Example:

- derived2 inherits from base1 and derived1
- derived1 inherits from base1
STLView transforms templates into readable and understandable information

- STLView supports std::vector, std::list, std::map, std::string
- See doc for which STL implementations are supported
STLView

STLView transforms templates into readable and understandable information.
LAB 2: VIEWING, EXAMINING, WATCHING AND EDITING DATA
DEBUGGING FOR PARALLEL APPLICATIONS
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: command line

<table>
<thead>
<tr>
<th>Platform</th>
<th>Command Line Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRAY</td>
<td><code>totalview aprun -a -n 16 myprog</code></td>
</tr>
<tr>
<td>IBM</td>
<td><code>totalview poe -a myprog -procs 4 -rmPOOL 0</code></td>
</tr>
<tr>
<td>QUADRRICS</td>
<td><code>totalview srun -a -n 16 -p pdebug myprog</code></td>
</tr>
<tr>
<td>Intel Linux under SLURM</td>
<td><code>totalview srun -a -n 16 -p pdebug myprog</code></td>
</tr>
<tr>
<td>MVAPICH</td>
<td><code>totalview srun -a -n 16 -p pdebug myprog</code></td>
</tr>
<tr>
<td>Opteron Linux under SLURM</td>
<td><code>totalview mpiRun -a myprog -np 16</code></td>
</tr>
<tr>
<td>Sun</td>
<td><code>totalview mprun -a myprog -np 16</code></td>
</tr>
<tr>
<td>MPICH</td>
<td><code>mpirun -np 16 -tv myprog</code></td>
</tr>
<tr>
<td>MPICH2 Intel MPI</td>
<td><code>totalview python -a </code>which mpiexec<code> -tvsu -np 16 myprog</code></td>
</tr>
</tbody>
</table>

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
• **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.
• Connecting to a subset of a job reduces tokens and overhead
• Can change this during a run
• Groups->Subset Attach
View MPI Message Queues

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

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

- **Filtering**
  - Tags
  - MPI Communicators
- **Cycle detection**
  - Find deadlocks
Parallel Backtrace View

- Group threads by common stack backtrace themes
- Starts with a compact representation of large jobs
- Tree based grid
LAB 3: EXAMINING AND CONTROLLING A PARALLEL APPLICATION
MEMORY DEBUGGING WITH MEMORYSCAPE
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
- Detects
  - Malloc API misuse
  - Memory leaks
  - Buffer overflows
- Supports
  - C, C++, Fortran
  - Linux, Unix, and Mac
  - MPI, pthreads, OMP
- Easy to use
  - Works with vendor libraries
  - No recompilation or instrumentation
  - Enables Collaboration
What is a Memory Bug?

- A Memory Bug is a mistake in the management of heap memory

  - Leaking: Failure to free memory
  - Dangling references: Failure to clear pointers
  - Failure to check for error conditions
  - Memory Corruption
    - Writing to memory not allocated
    - Overrunning array bounds
Why Are Memory Bugs Hard to Find?

What is a Memory Bug?

• Memory problems can lurk
  • For a given scale or platform or problem, they may not be fatal
  • Libraries could be source of problem
  • The fallout can occur at any subsequent memory access through a pointer
  • The mistake is rarely fatal in and of itself
  • The mistake and fallout can be widely separated

• Potentially 'racy'
  • Memory allocation pattern non-local
  • Even the fallout is not always fatal. It can result in data corruption which may or may not result in a subsequent crash

• May be caused by or cause of a 'classic' bug
The Agent and Interposition

Process

User Code and Libraries

Malloc API

TotalView
The Agent and Interposition

TotalView

Process

User Code and Libraries

Heap Interposition

Agent (HIA)

Malloc API

Allocation Table

Deallocation Table
Advantages of TotalView HIA Technology

• Use it with your existing builds
  • No Source Code or Binary Instrumentation

• Programs run nearly full speed
  • Low performance overhead

• Low memory overhead
  • Efficient memory usage
• Automatically detect allocation problems
• Memory Corruption Detection - Guard Blocks & Red Zones
• Leak detection
• Dangling pointer detection
• View the heap
• Memory Hoarding
• Memory Comparisons between processes
Leaks and Dangling Pointers

- Normal allocation: `ptr` points to memory.
- Leaked memory: `ptr` points to memory, but the memory is no longer valid.
- Dangling pointer: `ptr` points to invalid memory, indicating a potential crash or undefined behavior.
Enabling Memory Debugging
Memory Event Notification

Select events to trigger:

<table>
<thead>
<tr>
<th>Event</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>API usage error</td>
<td>Incorrect API or API instance used in operation</td>
</tr>
<tr>
<td>Allocation failed</td>
<td>Error: An allocation call failed or the address returned is NULL which generally m...</td>
</tr>
<tr>
<td>Double allocation</td>
<td>Error: Allocator returned a block already in use: heap may be corrupted</td>
</tr>
<tr>
<td>Double free</td>
<td>Error: Program attempted to free an already freed block</td>
</tr>
<tr>
<td>Free interior pointer</td>
<td>Error: Program attempted to free a block incorrectly, via an address in the middle...</td>
</tr>
<tr>
<td>Free notification</td>
<td>A block for which notification was requested is being freed</td>
</tr>
<tr>
<td>Free unknown block</td>
<td>Error: Program attempted to free an address not in the heap</td>
</tr>
<tr>
<td>Guard corruption error</td>
<td>Bounds error: The guard area around a block has been overwritten</td>
</tr>
<tr>
<td>Invalid aligned allocation request</td>
<td>Error: Program supplied an invalid alignment argument to the heap manager</td>
</tr>
<tr>
<td>Misaligned allocation</td>
<td>Error: Allocator returned a misaligned block: heap may be corrupted</td>
</tr>
<tr>
<td>Realloc notification</td>
<td>A block for which notification was requested is being reallocated</td>
</tr>
<tr>
<td>Realloc unknown block</td>
<td>Error: Program attempted to reallocate an address not in the heap</td>
</tr>
<tr>
<td>Red Zone overrun error</td>
<td>Bounds error: Attempting to access memory beyond the end of an allocated block</td>
</tr>
<tr>
<td>Red Zone overrun on deallocated block</td>
<td>Bounds error: Attempting to access memory beyond the end of a deallocated block</td>
</tr>
<tr>
<td>Red Zone underrun error</td>
<td>Bounds error: Attempting to access memory before the start of an allocated block</td>
</tr>
<tr>
<td>Red Zone underrun on deallocated block</td>
<td>Bounds error: Attempting to access memory before the start of a deallocated block</td>
</tr>
<tr>
<td>Red Zone use-after-free error</td>
<td>Access error: Attempting to access a block after it has been deallocated</td>
</tr>
<tr>
<td>Termination notification</td>
<td>The target is terminating, memory analysis can be performed</td>
</tr>
</tbody>
</table>

[Options: All, None]
Memory Debugging Options

Select your preferred level of debugging below or press Advanced Options for more control.

- **Enable memory debugging**

**Levels of Debugging**

- **Low**
  Provides event notifications and leak detection. It allows the best performance for your process.

- **Medium**
  Adds corrupted memory detection by applying guard blocks. Performance may degrade slightly.

- **High**
  Provides memory over run alerts by monitoring Red Zone violations. Your memory consumption will increase significantly.

- **Extreme**
  Enables all options. There is a risk that performance may suffer and you will use more memory.
Memory Debugging Advanced Options

Customize your options below or press Basic Options for predefined settings.

- Enable memory debugging

+ Halt execution on memory event or error
+ Guard allocated memory
+ Use Red Zones to find memory access violations
+ Paint memory
+ Hoard deallocated memory

Yellow buttons mean:
- multiple processes are selected
- the settings can vary among selected processes
- you can modify the settings for all these processes by pressing the yellow buttons
Guard Blocks

Guard allocated memory
When selected, the Memory Debugger writes guard blocks before and after a memory block that your program allocates

Pre-Guard and Post-Guard Size:
Sets the size in bytes of the block that the Memory Debugger places immediately before and after the memory block that your program allocates

Pattern:
Indicates the pattern that the Memory Debugger writes into guard blocks. The default values are 0x77777777 and 0x99999999
### Guard Blocks

#### Preceding Block
- Address: 0x00040a028
- Size: 171 bytes
- Pattern: 0x77777777

#### Corrupted Block
- Address: 0x00040a0e8
- Size: 32 bytes
- Pattern: 0x99999999

#### Following Block
- Address: 0x0004a107

#### Backtrace
1. **Function**: malloc
   - **Line #**: 149
   - **Source Information**: `malloc_wrap` in `hia_events.c`

2. **Function**: main
   - **Line #**: 106
   - **Source Information**: `main` in `hia_events.c`

3. **Function**: _libc_start_main
   - **Line #**: 105
   - **Source Information**: `libc.so.6`

4. **Function**: _start
   - **Line #**: 107
   - **Source Information**: `hia_events`

5. **Function**: break;

6. **Function**: case notify_guard_corrupt

7. **Function**: s3 = (char *)malloc ( 0x100

---

©Copyright 2012 Rogue Wave Software, Inc.
Red Zones: instant array bounds detection

• **Red Zones provides:**
  • Both read and write memory access violations.
  • Immediate detection of memory overruns.
  • Detection of access violations both before and after the bounds of allocated memory.
  • Detection of deallocated memory accesses.

• **Red Zones events**
  • MemoryScape will stop your program's execution and raise an event alerting you to the illegal access. You will be able to see exactly where your code overstepped the bounds.
Red Zones

Red Zones instant array bounds detection for Linux

- Red Zones allocation size range controls
  - The optional use of Red zones will increase the memory consumption of your program.
  - Controls are provided to allow the full management of Red Zone usage. These controls allow:
    - Restriction of red zones to allocations in several user defined size ranges
    - Easily turning red zones on and off at any time during your programs execution.
Memory Hoarding

- **Memory Hoarding**
  - Stops the memory manager from reusing memory blocks
  - Can detect certain memory errors

- **Hoard Low Memory Controls**
  - Automatically release hoarded memory when available memory gets low, allowing your program to run longer

- **Hoard Low Memory events**
  - MemoryScape can stop execution as notification that the hoard dropped below a particular threshold. This provides an indication that the program is getting close to running out of memory.
Memory Event Details Window

Memory Event Details - Process 1: filterapp-mpi.1 - 1

Event Location Allocation Location Deallocation Location Block Details

<table>
<thead>
<tr>
<th>ID</th>
<th>Function</th>
<th>Line #</th>
<th>Source Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>free</td>
<td></td>
<td>184malloc_wrappers_dlopen.c</td>
</tr>
<tr>
<td></td>
<td>double_free</td>
<td>74</td>
<td>main-727main.cxx</td>
</tr>
<tr>
<td></td>
<td>main</td>
<td>257</td>
<td>main-727main.cxx</td>
</tr>
<tr>
<td></td>
<td>__libc_start_main</td>
<td>1</td>
<td>libc.so.6</td>
</tr>
<tr>
<td></td>
<td>__start</td>
<td></td>
<td>filterapp-mpi</td>
</tr>
</tbody>
</table>

Source: /home/demouser/mpi-src/main.cxx

73 // Show that the deallocation stack is available now
74 junk = 0;
75 // Now release the memory the second time - illegal
77 #ifdef USEMPI
78 if ( rank == 1 )
79    malloc ( junk );
80 free ( junk );

Point of Allocation

<table>
<thead>
<tr>
<th>ID</th>
<th>Function</th>
<th>Line #</th>
<th>Source Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>99</td>
<td>malloc</td>
<td>166</td>
<td>main-727main.cxx</td>
</tr>
<tr>
<td></td>
<td>double_free</td>
<td>60</td>
<td>main-727main.cxx</td>
</tr>
<tr>
<td></td>
<td>__libc_start_main</td>
<td>1</td>
<td>libc.so.6</td>
</tr>
<tr>
<td></td>
<td>__start</td>
<td></td>
<td>filterapp-mpi</td>
</tr>
</tbody>
</table>

Source: /home/demouser/mpi-src/main.cxx

60 p = (int*) malloc( length );
61 printf ( "allocated %4d (%#x) bytes at %p\n", length, p );
62 // Breakpoint here
63 // Show allocated annotation

Close View in Block Properties window Help
Leak Detection

- Based on Conservative Garbage Collection
- Can be performed at any point in runtime
  - Helps localize leaks in time
- Multiple Reports
  - Backtrace Report
  - Source Code Structure
  - Graphically Memory Location
Dangling Pointer Detection

<table>
<thead>
<tr>
<th>Expression:</th>
<th>Address:</th>
</tr>
</thead>
<tbody>
<tr>
<td>addr</td>
<td>0xbfffd1f4</td>
</tr>
<tr>
<td>Type:</td>
<td>int *</td>
</tr>
</tbody>
</table>

Value

0x080496d0 (Dangling) → 0x00000000 (0)

<table>
<thead>
<tr>
<th>Expression:</th>
<th>Address:</th>
</tr>
</thead>
<tbody>
<tr>
<td>misaddr</td>
<td>0xbfffd1f0</td>
</tr>
<tr>
<td>Type:</td>
<td>int *</td>
</tr>
</tbody>
</table>

Value

0x080496e4 (Dangling Interior) → 0x00000000 (0)
Memory Comparisons

- “Diff” live processes
  - Compare processes across cluster
- Compare with baseline
  - See changes between point A and point B
- Compare with saved session
  - Provides memory usage change from last run
Memory Usage Statistics
Memory Reports

- **Multiple Reports**
  - Memory Statistics
  - Interactive Graphical Display
  - Source Code Display
  - Backtrace Display

- **Allow the user to**
  - Monitor Program Memory Usage
  - Discover Allocation Layout
  - Look for Inefficient Allocation
  - Look for Memory Leaks
Multiple processes and threads

- Memory debug many processes at the same time
  - MPI
  - Client-Server
  - Fork-Exec
  - Compare two runs
- Remote applications
- Multi-threaded applications
Script Mode - MemScript

**Automation Support**

- MemoryScape lets users run tests and check programs for memory leaks without having to be in front of the program
- Simple command line program called MemScript
  - Doesn’t start up the GUI
  - Can be run from within a script or test harness
- The user defines
  - What configuration options are active
  - What thing to look for
  - Actions MemoryScape should take for each type of event that may occur
LAB 4, 5, 6: MEMORY LABS
BATCH DEBUGGING

Using scripts for unattended debugging
tvscript and memscript

• A straightforward language for unattended and/or batch debugging with TotalView and/or MemoryScape
• Usable whenever jobs need to be submitted or batched
• Can be used for automation
• A more powerful version of printf, no recompilation necessary between runs
• Schedule automated debug runs with cron jobs
• Expand its capabilities using TCL
Output

- All of the following information is provided by default for each print:
  - Process id
  - Thread id
  - Rank
  - Timestamp
  - Event/Action description

- A single output file is written containing all of the information regardless of the number of processes/threads being debugged.
• **Simple interface to create an action point**
  - `create_actionpoint "#85=>print foreign_addr"

• **Sample output with all information**

```plaintext
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! Print
!
! Process:
! ./TVscript_demo (Debugger Process ID: 5, System ID: 2457@127.0.1.1)
! Thread:
!   Debugger ID: 5.1, System ID: 3077191888
! Rank:
!   0
! Time Stamp:
!   05-14-2012 17:11:24
! Triggered from event:
!   actionpoint
! Results:
!   err_detail = {
!     intervals = 0x0000000a (10)
!     almost_pi = 3.1424259850011
!     delta = 0.000833243988525023
!   }
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```
Command syntax

• **General syntax**
  • `tvscript [options] [filename] –a [program_args]`

• **MPI Options**
  • `-mpi starter` starter comes from Parallel tab dropdown
  • `-starter_args “args for starter program”`
  • `-nodes`
  • `-np or –procs or –tasks`
Command syntax

• **Action options**
  - `-create_actionpoint “src_expr[=>action1[,action2] …]”`
    - Repeat on command line for each actionpoint
  - `-event_action “event_action_list”`
    - event1=action1,event2=action2 or event1=>action1,action2
    - Can repeat on command line for multiple actions

• **General options**
  - `-display_specifiers “display_specifiers_list”`
  - `-maxruntime “hh:mm:ss”`
  - `-script_file scriptFile`
  - `-script_log_filename logFilename`
  - `-script_summary_log_filename summaryLogFilename`
Script Files

- Instead of putting everything on the command line, you can also write and use script files

  `tvscript --script_file script_file`

- Script files can also include TCL commands

- Logging functions
  - `tvscript_log msg` – logs msg to the log file
  - `tvscript_slog msg` – logs msg to the summary log file

- Property functions
  - `tvscript_get_process_property process_id property`
  - `tvscript_get_thread_property thread_id property`
LAB 7: BATCH MODE DEBUGGING WITH TVSCRIPT
REVERSE DEBUGGING WITH REPLAYENGINE
What is ReplayEngine?

- Provides record for deterministic replay
- Records program changes as they happen
- Captures input
  - Function calls
  - Network and file I/O
- Captures non-determinism
  - Forces single threaded execution
  - Records context switches
- Allows stepping back in execution, like a DVR for your programs
- Use breakpoints and watchpoints
- Support for MPI on Ethernet, Infiniband, Cray XE Gemini
- Support for Pthreads, and OpenMP
ReplayEngine Support

• Replay on Demand: enable it when you want it
• Supported on Linux for x86 and x86_64
• Cluster interconnects
  • IP (any interconnect): MPICH, MPICH2, OpenMPI, Intel MPI, SGI MPT, Cray XT-MPT, MVAPICH, MVAPICH2
  • Mellanox Infiniband
    • IB verb: MVAPICH, MVAPICH2, OpenMPI, Intel MPI
  • Qlogic Infiniband
    • PSM: MVAPICH, MVAPICH2, OpenMPI, Intel MPI
• **Editing during record mode**
  
  • Allows modification of variables during record mode (eval breakpoints, click/edit of variable values)
  
  • Modifications are recorded along with the rest of the execution
  
  • Not allowed to change values when in playback mode
  
  • Don’t attempt to step into recorded edits, but correct values show up on either side
ReplayEngine

An Intuitive User Interface

- Step forward over functions
- Step forward into functions
- Advance forward out of current Function, after the call
- Advance forward to selected line
- Advance forward to “live” session

- Step backward over functions
- Step backward into functions
- Advance backward out of current Function, to before the call
- Advance backward to selected line
ReplayEngine

Replay not running

Enable Replay “On Demand”

No Active Buttons!

Replay running

Active Buttons
Example

ReplayEngine

Consider the following very difficult program scenario:

• A crash occurs that destroys the stack backtrace, giving no information leading up to the problem

• ReplayEngine can be used to work backwards from the crash, and even to observe the stack recreate itself, providing the critical information on where and how the problem began.

• The ReplayEngine provides the ability to review any part of the program execution... to see all variables and function calls, from the beginning of the run to the current time
Function `funcB` in `ReplayEngine_demo.cpp`

```cpp
44 int *p;
46 c=c+2;
47 p=&c;
48
50 if( c<MAXDEPTH )
51 c=funcA(c);
52
53 for( i=arraylength-1; i>=0; i-- )
54 v[i]=*p;
55
56
```

Registers for the frame:

Function "funcB":
- b: 0x00000000 (6)
- Block "$b1":
  - c: 0x00000008 (8)
  - i: 0xbfeae6d8 (-107)
  - v: (int[20])
  - p: 0xbfeae6bc -> 0x

Stack Frame:

Process 1 (9179): ReplayEngine_demo (Stopped)
Thread 1 (9179) (Stopped) <Trace Trap>
Function funcB in ReplayEngine_demo.cpp

```cpp
int funcB(int b) {
    int c;
    int i;
    int v[MAXDEPTH];
    int *p;

    c = b + 2;
    p = &c;

    if (c < MAXDEPTH )
        c = funcA(c);
```
Stack Trace

Registers for the frame:

%eax: 0xbfeae558 (-107512)
%ecx: 0xbfeae854 (-107512)
%edx: 0x00000001 (1)
%ebx: 0xb7e17ff4 (-120995)
%esp: 0xbfeae4e0 (-107512)
%ebp: 0xbfeae568 (-107512)
%esi: 0xb7ff5ce0 (-120800)

Unknown

0x0000000e: ???
0x0000000f: ???
0x00000010: ???
0x00000011: ???
0x00000012: ???
0x00000013: ???
0x00000014: ???
0x00000015: ???
0x00000016: ???
0x00000017: ???
0x00000018: ???
0x00000019: ???
0x0000001a: ???

Action Points  Processes  Threads
Function `funcB` in `ReplayEngine_demo.cxx`

```c++
44:    int v[MAXDEPTH];
45:    int *p;
46:    c=b+2;
47:    p=&c;
48:
49:    if( c<MAXDEPTH )
50:      c=funcA(c);
51:    
52:    for (i=arraylength-1; i>0; i--){
53:      v[i]=*p;
54:    }
```
LAB 8: REVERSE DEBUGGING WITH REPLAY ENGINE
New capabilities in TotalView 8.11

- CUDA 4.2 support
- Intel Xeon Phi (MIC) early access
  - Native mode (native processes or parallel style MPI jobs)
  - Directive based offload support (LEO)
- OpenACC support for Cray CCE 8 compilers
- Early access preview of MRNet infrastructure
What’s coming in TotalView 8.12
(target ISC13)

- Xeon Phi Support
- Improved support for AVX instructions, registers and addressing modes
- New TotalView debugging session management capability
- Apple OS X Lion and Mountain Lion support
- Improved Cray XC series support
- New capabilities for searching and filtering multiple breakpoint addresses.
- New STLView support for STL set, multi-set and multi-map collectives.
What’s planned for TotalView 8.13
(target SC2013)

- Support for NVIDIA CUDA 5.5 capabilities such as dynamic parallelism
- Enhancements to TotalView’s support for Xeon Phi
- Disassembly support for more rarely-used AVX instructions
- Improved performance at scale
- One or more of the following (tbc)
  - Support for Fortran 2008 Coarray functionality
  - CAPS-HMPP OpenACC
  - PGI OpenACC
  - PGI CUDA Fortran
  - OpenCL
QUESTIONS?

Contact

dean.stewart@roguewave.com
neil.foster@roguewave.com

http://www.roguewave.com
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

Worldwide, except Japan

Engineers are available from 8:00 AM to 5:00 PM Eastern Time, Monday-Friday
US: 800-858-3760
Telephone: 508-652-7700
Fax: 508-652-7701
Email: tsvsupport@roguewave.com
Submit a Web Support Incident at: File a Support Request

IMSL Numerical Libraries

Worldwide, except Japan

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

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

TotalView is a GUI-based source code debug 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!