TotalView

Agenda

• Introduction
• Startup
• UI Navigation and Process Control
• Action Points
• Data Monitoring and Visualization
• Debugging for Parallel Applications

Agenda

• Remote Display Debugging
• CUDA Debugging
• Memory Reporting with MemoryScape
• Memory Debugging with MemoryScape

Agenda

• Batch Debugging
• Reverse Debugging with ReplayEngine
• Advanced Asynchronous Control for Parallel Applications
• Type Transformations
• Support and 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, UPC
  - Unix, Linux, OS X
- Handles Concurrency
  - Multithreaded Debugging
  - Parallel Debugging
  - MPI, PVM, Others
  - Remote and Client/Server Debugging
- Integrated Memory Debugging
- Reverse Debugging available
- ReplayEngine
- Supports a Variety of Usage Models
  - Powerful and Easy GUI
  - Visualization
  - CLI for Scripting
  - Long Distance Remote Debugging
  - Unattended Batch

Supported Compilers and Architectures

- Platform Support
  - Linux x86, x86-64, ia64, Power
  - Mac Intel
  - Solaris Sparc and AMD64
  - AIX
  - Cray XT, XE, XK
  - IBM BGL, BG/P
  - 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 – Standard I/O redirection

Attach to Process

Attach to Process – Enable Replay Engine

Open a Core File
Via Command Line

Normal
`totalview [ tv_args ] prog_name [-a prog_args ]`

Attach to running program
`totalview [ tv_args ] prog_name -pid PID# [-a prog_args ]`

Attach to remote process
`totalview [ tv_args ] prog_name -remote name [-a prog_args ]`

Attach to a core file
`totalview [ tv_args ] prog_name corefile_name [ -a prog_args ]`

Interface Concepts

**Root Window**

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

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

TotalView Root Window

- Host name
- Expand - Collapse Toggle
- TotalView Thread ID #
- Process Status
- Action Point ID number

**TotalView Root Window**

- **Action Point**
  - Host name
  - Expand/Collapse Toggle
  - Rank # (if MPI program)
  - Process status
  - TotalView Thread ID #

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

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

- **Stack Trace Pane**: Provides detailed state of one process, or a single thread within a process.
- **Stack Frame Pane**: A single point of control for the process and other related processes.
- **Source Pane**: View as Source - or Assembly - or Both!
- **Tabbed Area**: Action Points Tab, Processes Tab, Threads Tab:
  - Action Points: all currently defined action points
  - Processes: all current processes
  - Threads: all current threads, ID’s, Status

Stack Trace and Stack Frame Panes

- **Language**
- **Name**
- **Frame Pointer**
- **Local Variables**
- **Register Values**

- **Click to refocus source pane**
- **Click to modify**
- **Dive for variable window**
Process Status

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

Search Paths

- Search Path Variable You Can Set
  - SOURCE_SEARCH_PATH
  - OBJECT_SEARCH_PATH
  - SHARED_LIBRARY_SEARCH_PATH
  - EXECUTABLE_SEARCH_PATH
  - Each is a colon-separated list of paths

- Search Mappings – applied to paths before searching
  - SOURCE_SEARCH_MAPPINGS
  - OBJECT_SEARCH_MAPPINGS
  - SHARED_LIBRARY_SEARCH_MAPPINGS
  - EXECUTABLE_SEARCH_MAPPINGS
  - Each is a colon-separated list of RE=replacement mappings
    - \+regular_expression\+replacement\+:\+re\+:\+repl\+

- Search Path Variables TotalView Sets (Read-only)
  - COMPILATION_DIRECTORY_COMPONENT
  - COMPILATION_WORKING_DIRECTORY
  - COMPILATION_DIRECTORY
  - EXECUTABLE_DIRECTORY_COMPONENT
  - EXECUTABLE_WORKING_DIRECTORY
  - EXECUTABLE_DIRECTORY
  - You can use these in setting *PATH variables for search
Search Paths

• TotalView Built-in Functions
  • $tree(/dir/to/search1:/dir/to/search2)
    • Searches each colon-separated directory and all subdirectories. The $tree directive cannot be first or last in the Source search path, but you may have multiple $tree entries
  • $link(/dir/to/search1:/dir/to/search2)
    • Searches symbolic links in each colon-separated directory. Fails if file is not a symbolic link and in the exact directory

Managing Signals
File > Signals

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

Preferences

Finding Functions, Variables, and Source Files
Stepping Commands

Based on PC location

Using Set PC to resume execution at an arbitrary point

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

Debug Menu

- Menu Items for extra features
  - Replay Engine
  - MemoryScape
  - CUDA Memcheck
  - More on these features later
Examining Your Programs

Examining Where Your Programs Are

Parallel back trace

Viewing TotalView Informational Messages
Setting Up Your Windows

- Update current or all windows
  - Momentarily pauses execution (if running) to determine current state and update
- Duplicate a window
- Memorize current or all windows
  - Remembers window positions and sizes for future use
  - Only works if Preference Force Window Position is selected
- Focus on Root window

ACTION POINTS

Action Points

Breakpoints
Barrier Points
Conditional Breakpoints
Evaluation Points
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
- Optional contextual menu access for all functions
- Action Points Tab
  - Lists all action points
  - Dive on an action point to focus it in source pane
  - Action point properties
    - In Context menu
  - Saving all action points
    - Action Point > Save All
Setting Breakpoints

- Breakpoint->At Location...
  - Specify function name or line #
  - Specify class name and break on all methods in class, optionally with virtuals and overrides

49

Setting Breakpoints

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

50

Conditional Breakpoint

Evaluation Breakpoint...
Test Fixes on the Fly!

- Test small source code patches
- Call functions
- Set variables
- Test conditions
- C/C++ or Fortran
- Can’t use C++ constructors
- Use program variables
- ReplayEngine records changes but won’t step through them

51

52
Setting Breakpoints With C++ Templates

TotalView understands C++ templates and gives you a choice ...

Boxes with solid lines around line numbers indicate code that exists at more than one location.

Watchpoints

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

Action Point -> Create Watchpoint...

Watchpoints

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

Watchpoints

- Can create from right-click on variable in Source pane
Watchpoints

- Watchpoints are set on a memory region, not a variable
- Watch the variable scope and disable watchpoints when a variable is out of scope
- Can be conditional, just like other action points
  - Use $newval and $oldval in your evaluation to find unexpected changes in value (such as a loop value changing by more than 1)

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

LAB 1: THE BASICS

DATA MONITORING AND VISUALIZATION
Diving on a Common Block in the Stack Frame Pane

Undiving

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

In a Variable Window: replace contents with the previous contents. You can also remove changes in the variable window with Edit > Reset Default.

The Variable Window

• Click once on the value
• Cursor switches into edit more
• Esc key cancels editing
• Enter key commits a change
• Editing values changes the memory of the program

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

Expression List Window

Add to the expression list using contextual menu with right-click on a variable, or by typing an expression directly in the window

• Reorder, delete, add
• Sort the expressions
• Edit expressions in place
• Dive to get more info

• Updated automatically
• Expression-based
• Simple values/expressions
• View just the values you want to monitor
Viewing Arrays

- Data Arrays
- Structure Arrays

Array Viewer

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

Slicing Arrays

- Slice notation is [start:end:stride]

Filtering Arrays
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.

Dive in All

Dive in All will display an element in an array of structures as if it were a simple array.

Looking at Variables across Processes

TotalView allows you to look at the value of a variable in all MPI processes
- Right Click on the variable
- Select the View > View Across
- TotalView creates an array indexed by process
- You can filter and visualize
- Use for viewing distributed arrays as well.

Typecasting Variables

- Edit the type of a variable
- View data as type...
- Often used with pointers

Type Casts Read from Right to Left
- int[10] * Pointer to an array of 10 int
- int*[10] Array of 10 pointers to int

- Cast float * to float [100] * to see a dynamic array’s values
- Cast to built-in types like $string$ to view a variable as a null-terminated string
- Cast to $void$ for no type interpretation or for displaying regions of memory

The Bottom Line
Give TotalView a starting memory address and you can tell TotalView how to interpret your memory from that starting location.
Typecasting a Dynamic Array

C++ Class Hierarchies

Variable Window shows class hierarchy using indentation

Note:
- Virtual public base classes appear each time they are referenced
- The vtable entry here is part of the C++ implementation but can provide useful information

Example:
- derived2 inherits from base1 and derived1
- derived1 inherits from base1

Fortran 90 Modules
Tools > Fortran Modules

STLView

STLView transforms templates into readable and understandable information

- STLView supports std::vector, std::list, std::map, std::string
- See doc for which STL implementations are supported
STLView transforms templates into readable and understandable information.

LAB 2: VIEWING, EXAMING, WATCHING AND EDITING DATA

DEBUGGING FOR PARALLEL APPLICATIONS

TotalView Startup with MPI
TVT Launch

In the Parallel tab, select:

your MPI preference, number of tasks, and number of nodes.

… then add any additional starter arguments
TotalView Startup with MPI

The order of arguments and executables is important, and differs between platforms.

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

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

- 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

User Defined Groups

- Group > Custom Groups, to create a process group of some other specification
- Group Membership shown in Processes Tab
- User defined groups appear in the “Go” drop-down menu

Parallel Back Trace

- Look for outliers

User Defined Groups

- Group > Custom Groups, to create a process group of some other specification
- Group Membership shown in Processes Tab
- User defined groups appear in the “Go” drop-down menu

Preferences
Subset Attach

- 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 Graph

Message Queue Debugging

- Filtering
  - Tags
  - MPI Communicators
- Cycle detection
  - Find deadlocks
Strategies for Large Jobs

- **Reduce N**
  - **Problem:** Each process added requires overhead
  - **Strategy:** Reduce the number of processes TotalView is attached to
    - Simply reducing N is best, however data or algorithm may require large N
  - **Technique:** subset attach mechanism

- **Focus Effort**
  - **Problem:** Some debugger operations are much more intensive than others, and when multiplied by N this could be significant
  - **Strategy:** Reduce the interaction between the debugger and the processes
  - **Technique:** Use TotalView’s process control features to
    - Avoid single stepping
    - Focus on one or a small set of processes

LAB 3: EXAMINING AND CONTROLLING A PARALLEL APPLICATION

REMOTE DISPLAY DEBUGGING

- 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
  - Connection info can be saved for reuse
  - 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

- On Windows, there is a setup wizard
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

Security

- Remote Display Client uses SSH
- Remote Display Server allows only RFB (Remote Frame Buffer) connections
- No incoming access to the server is allowed
- Server can only connect back to the client viewer via SSH
- Only one viewer connection allowed per server
- No password information is saved, user is prompted every time

CUDA DEBUGGING

TotalView for CUDA

- Characteristics
  - Full visibility of both Linux threads and GPU device threads
  - Fully represent the hierarchical memory
  - Detailed device status display
  - Supports Unified Virtual Addressing and GPUDirect
  - Thread and Block Coordinates
  - Device thread control
  - Handles both inlined functions and CUDA callstack
  - Reports memory access errors
  - Full Multi-Device Support
  - Can be used with MPI
**TotalView for CUDA**

- **Supports:**
  - CUDA 3.2
  - CUDA 4.0
  - CUDA 4.1
  - Linux x86-64
  - NVIDIA Tesla or Fermi hardware
  - Support for no copy pinned memory
  - CUDA device assertions
  - Multiple CUDA contexts from same process on same device
  - CUDA on Cray, early access* support of OpenACC

*Not officially supported

---

**NVIDIA GPU Architecture**

- **Used in conjunction with conventional CPUs**
  - Acts as an accelerator to a host processor
  - Or, perhaps the host processor acts to support GPU
- **Distinct architecture**
  - Distinct processor architecture from the CPU
- **Many more cores than SMP**
  - Multiple streaming multiprocessors
  - Potentially tens of thousands of thread contexts

---

**GPU Memory Hierarchy**

- **Hierarchical memory**
  - Local (thread)
    - Local
    - Register
  - Shared (block)
  - Global (GPU)
    - Global
    - Constant
    - Texture
  - System (host)

---

**TotalView Type Storage Qualifiers**

- **@parameter Address**
  - Offset in parameter storage
- **@local Address**
  - Offset in local storage
- **@shared Address**
  - Offset in shared storage
- **@constant Address**
  - Offset in constant storage
- **@global Address**
  - Offset in global storage
- **@register Address**
  - PTX register name
CUDA Variables

- Storage qualifiers appear in the data type

![CUDA Variables Table]

CUDA

- Function-like kernels are written for calculations to be performed on the GPU
  - Data parallel style, one kernel per unit of work
- Presents hierarchical organization for thread contexts
  - 2D or 3D grid of blocks
  - 3D block of thread
- Exposes memory hierarchy explicitly to the user
- Includes routines for managing device memory and data movement using streams

CUDA Programming Challenges

- Coordinating CPU code and device code
- Understanding what is going in each kernel
- Exceptions
- Understanding memory usage
- Understanding performance characteristics

Debugging CUDA in TotalView

- When a new kernel is loaded, you get the option of setting breakpoints
- Once breakpoints are set, you can turn off the dialog and say no

![Debugging CUDA in TotalView Dialog]
**Debugging CUDA in TotalView**

- CUDA threads are considered part of the initiating process
- CUDA threads are given a negative TotalView thread id to distinguish them
- Normal TotalView controls work on CUDA code
- Underneath Toolbar is a GPU focus thread selector for changing block and thread indices

**Control of Threads and Warps**

- Warps advance synchronously
  - They share a PC
- Single step operation advances all GPU threads in the same warp
- Stepping over a __syncthreads() call will advance all relevant threads
- To advance more than one warp
  - Continue, possibly after setting a new breakpoint
  - Select a line and “Run To”

**GPU Device Status**

- Display of PCs across SMs, Warps and Lanes
- Updates as you step
- Shows what hardware is in use
- Helps you map between logical and hardware coordinates

**CUDA Segmentation Faults**

- TotalView displays segmentation faults as expected
  - Must enable CUDA memory checking
CUDA Built-in Runtime Variables

• Supported built-in runtime variables are:
  • struct dim3_16 threadIdx;
  • struct dim2_16 blockIdx;
  • struct dim3_16 blockDim;
  • struct dim2_16 gridDim;
  • int warpSize;

MEMORY REPORTING WITH MEMORYSCAPE

What is a Memory Bug?

• A Memory Bug is a mistake in the management of heap memory
  • Failure to check for error conditions
  • Leaking: Failure to free memory
  • Dangling references: Failure to clear pointers
  • Memory Corruption
    • Writing to memory not allocated
    • Overrunning array bounds

Why Are Memory Bugs Hard to Find?

• 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
**TotalView HIA Technology**

- **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
  - Support wide range of platforms and compilers

**Memory Debugger Features**

- Automatically detect allocation problems
- View the heap
- Leak detection
- Block painting
- Memory Hoarding
- Dangling pointer detection
- Deallocation/redeallocation notification
- Memory Corruption Detection - Guard Blocks
- Memory Comparisons between processes
- Collaboration features
Enabling Memory Debugging
Memory Event Notification

- Heap graphical view

Memory Event Details Window

- 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

Memory Corruption Report

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

MEMORY DEBUGGING WITH MEMORYSCAPE

MEMORY DEBUGGING: MEMORYSCAPE

- Preview: Debugging Memory with MemoryScape
  - Startup
    - Integrated and Bundled with TotalView
    - Typically started from the TotalView gui
  - Multi-threaded and multi-process programs
    - Setup from TotalView or stand alone.
    - The multi-process and multi-threaded GUI interface is very similar to TotalView.
- Automation Support
- Block painting
- Memory Corruption Detection - Guard Blocks
- Memory Hoarding
Memory Debugging: MemoryScape

Multi-Process and Multi-Thread

- 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

Red Zones instant array bounds detection for Linux

- Red Zones provides:
  - 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 programs execution and raise an event alerting you to the illegal access. You will be able to see exactly where your code overstepped the bounds.

Menu Selections:

- Block painting
- Guard block
- Hoarding
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.

Red Zones instant array bounds detection for Linux

- Red Zones support in the CLI
  - The Command Line Interface also provides support for RedZones
  - Scripting support of new commands and command qualifiers
    - TVScript
    - MemScript

Configuring 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 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
malloc zones Support on Mac OS X

Mac OS X provides a mechanism for multiple pools of memory called malloc zones. MemoryScape now tracks both the allocator and owner of all heap allocations. These properties can be displayed and used for filtering.

- Display allocator and owner information
- Filter vs allocator and owner

Improved Memory Hoarding support

The use of memory hoarding in MemoryScape increases the risk of running out of available memory. MemoryScape now has the capability to manage this condition and alert you when you are at risk.

- 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.
- Hoard Low Memory scripting and CLI support
  - TVScript
  - MemScript

LAB 4, 5, 6: MEMORY LABS
Using scripts for unattended debugging

BATCH 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

Sample Output

- Simple interface to create an action point
  - `create_actionpoint "#85==>print foreign_addr"

- Sample output with all information

```
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 = 100000000a (10)
    almost_pi = 3.1424259850011
    delta = 0.000833243988525023
  }
```


Events

- General
  - any_event
- Source code debugging events
  - actionpoint
  - error
- Memory events (just a few, all are listed in Chapter 4 of TotalView Reference Guide)
  - any_memory_event
  - free_not_allocated
  - guard_corruption
  - rz_overrun, rz_underrun, rz_use_after_free

Actions

- Source code
  - display_backtrace [-level num] [numlevels] [options]
  - print [-slice {exp}] (variable | exp)
- Memory
  - check_guard_blocks
  - list_allocations
  - list_leaks
  - save_html_heap_status_source_view
  - save_memory_debugging_file
  - save_text_heap_status_source_view

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
- 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
Command syntax

- Memory debugging options
  - -memory_debugging (must use for debugging memory)
  - -mem_detect_leaks
  - -mem_detect_use_after_free
  - -mem_guard_blocks
  - -mem_hoard_freed_memory
  - -mem_hoard_low_memory_threshold nnnn
  - -mem_paint_all
  - -mem_paint_on_alloc
  - -mem_paint_on dealloc

- Memory debugging red zone options
  - -mem_red_zone_overruns
  - -mem_red_zones_size_ranges min:max[,min:max]…
    - Ranges can be
      - min:max
      - min:
      - :max
      - fixed
  - -mem_red_zones_underruns

Script Files

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

- Action point and event functions
  - tvscript_create_actionpoint source_loc_expr
    - [[#image#]filename#]line_number
    - function_name
    - class class_name
    - virtual class:signature
  - tvscript_add_actionpoint_handler id handler
  - tvscript_add_event_handler event handler
    - Passes an array to handler, event will either be error or actionpoint
    - Other information relevant to event
- Handlers are written in TCL
LAB 7: BATCH MODE DEBUGGING WITH TVSCRIPT

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

REVERSE DEBUGGING WITH REPLAYENGINE

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
**ReplayEngine**

- 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

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

**Example**

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
LAB 8: REVERSE DEBUGGING WITH REPLAY ENGINE

Why Asynchronous Control

- Parallel codes are very difficult to debug
- Breaking down the problem to smaller pieces helps narrow down issues
- Stepping individual processes, threads, or groups can help narrow down a problem

ADVANCED ASYNCHRONOUS CONTROL
TotalView Asynchronous Control Features

• Built in control groups
• User-defined control groups
• Action points can target threads, processes or groups
• Typical debugging commands can target groups or individual processes and threads (Next, Step, etc.)

Groups

• By default, TotalView defines the following groups:
  • Control Group: everything
  • Share Group: all processes and their threads with same image
  • Workers Group: all threads in all control group processes
  • Lockstep Group: all threads at the same breakpoint
  • Process: current process with debugger focus
  • Process Workers: all threads in the process
  • Process Lockstep: all threads at the same breakpoint in one process
  • Thread: current thread with focus
  • Only the Workers group can be modified by the user
  • CLI, use dworker 0 to remove from the workers group or dworker 1 to add

Customizing Groups

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

Creating a Custom Group

• Enter the group name
• Select processes to be members of the group
• Add... button to create more groups
Custom Groups in the CLI

- In the CLI, use the dgroups command to create & modify groups
  
dgroups –new [t/p –g groupname] [id_list]
dgroups –add [–g groupname] [id_list]
dgroups –remove [–g groupname] [id_list]
dgroups –intersect [–g groupname id_list]
dgroups –delete [–g groupname]
  
t or p – can also use thread or process, is it a thread or process group
  
groupname is your name for the new group
  
id_list is a TCL list of ids to add to the new group

- You can also use dworker to add/remove threads from the process workers group
dfocus t1.1 dworker 0

- can also use thread or process, is it a thread or process group
**Breakpoints**

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

**Breakpoints in UI**

**Eval Breakpoints in UI**

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

• Control where they are planted, defaults to the Share Group
  • Uses the SHARE_ACTION_POINT variable, true plants in the Share Group, false plants in the focus process only
• Control what is stopped by hitting the breakpoint, the group, the process, or just the thread
  • Uses the BARRIER_STOP_ALL variable set to: group, process, or none
  • Use --stop_when_hit flag in CLI to override default

Barriers

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

Barriers Satisfaction Group in UI

• Satisfaction Group determines how many times barrier needs to be reached before it is satisfied and can release all threads that have reached it.
  • In the UI, you can select from Control group, Process, or Workers
  • If you have created custom groups, they should also appear in the drop down list in the UI
  • CLI uses the intersection of the current focus and the share group to determine the satisfaction group
  • BE SURE YOUR ENTIRE SATISFACTION GROUP CAN REACH THE BARRIER OR YOU CAN BE DEADLOCKED
  • Barriers can also create deadlocks if a thread held by the barrier is holding a lock or another thread is dependent on a held thread’s output, etc.
Asynchronous Controls

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

Holds

- Group, Process, Thread can all be held
- Anything that is held won’t run or step again until it is unheld
- Hold status is indicated in dstatus, in the Process Window, and also under the toolbar in the UI
- Hold status also applies to anything that is held at a barrier prior to the satisfaction group completing the barrier
Holds

- When something is held, you must "un-hold" it
- Set focus to the held thread/process, then release the hold
LAB 9: ASYNCHRONOUS CONTROL

Why Type Transformations?
- Debugger shows only what is in the immediate class, including unexpanded base classes
- Dive into base classes, but have to do it for each object
- Types aren’t always obvious
  - Abstract base class pointers may be common, would like to automate downcasting
- Sometimes class data is unimportant to debugging and only serves to clutter the screen
- Internal data storage may be abstracted or difficult to interpret in its raw form

TotalView Type Transformations
- Allow programmers to write their own functions to interpret the data
  - Display base class members as if they are members of the derived class
  - Choose not to display some data members
  - Interpret data at runtime to display it in a more readable fashion
  - Functions can be written in TCL using TTF or in C++ using C++ Views
In $HOME/.tvsrc:

```cpp
::TV::TTF::RTF::build_struct_transform {
  name ("class\struct x1")
  members {
    pmonth (month )
    pName (xbase upcast (* pName ) )
    pStreet (xbase upcast(* pStreet ) )
    pVoid ("string *" cast v )
    pVoid2 ("class x2 " cast q )
  }
}
```

Meta Language:
- `{member}`
- `{* expr}`
- `{expr . Expr}`
- `{expr -> expr}`
- `{datatype case expr}`
- `{baseclass upcast expr}`

---

**TotalView TTF**

**C++ Views**

- Implement the display function for any type you want to use

```cpp
int ::TV_ttf_display_type(const display_type_t *object);
```

- **Precedence for the functions**
  1. Class-qualified static function matching the type
  2. Global function at file scope matching the type
  3. Global function matching the type

---

**C++ Views**

- How do I implement the display function?
  1. Include `tv_data_display.h` from TOTALVIEW/include
  2. Make calls to display data using:
     ```cpp
     int TV_ttf_add_row( const char *field_name, const char *type_name, const void *address);
     ```
  3. Call once for each piece of data to be displayed

---

**C++ Views**

- Be sure to check return value for calls to add row
  - It returns one of these values:
    - `TV_ttf_ec_ok`: Success
    - `TV_ttf_ec_not_active`: `TV_add_row` was called outside of a valid calling context
    - `TV_ttf_ec_invalid_characters`: The field or type name contains invalid characters, such as newline or tab
    - `TV_ttf_ec_buffer_exhausted`: `TV_add_row` has been called too many times, filling up the buffer. Generated types cannot be unlimited in length. For arrays, consider returning array type with a dynamic length, rather than iterating over all of its members. For non-arrays, the display will need to be truncated.
C++ Views

- After displaying your chosen data in your chosen fashion, return the appropriate code:
  - TV_ttf_format_ok: successful, show what was sent by this function
  - TV_ttf_format_ok_elide: ok, but elide type, used to keep display smaller
  - TV_ttf_format_failed: failure on known type
  - TV_ttf_format_raw: display the data in TotalView’s default fashion
  - TV_ttf_format_never: same as raw, but don’t ever call this function again

- When building, link to tv_data_display.o
  - Source file tv_data_display.c is in $TOTALVIEW/src
  - Optionally build it into its own library

Example 1 header

```c
struct Point { int importantValueX;
               int importantValueY;
               float meaninglessInternal;};

struct Circle{ Point pos; //poorly named center
               int r; //poorly named radius};

struct Square{ Point pos; //poorly named bottom
               left position
               int w; //poorly named width};
struct Rectangle : public Square { int h; //poorly named height};
```

TTF Example

```c
::TV::TTF::RTF::build_struct_transform {
  name {^class|struct Circle$}
  members {
    { centerX { pos . importantValueX } }
    { centerY { pos . importantValueY } }
    { radius { r } }
  }
}
```

TTF Example continued

```c
::TV::TTF::RTF::build_struct_transform {
  name {^class|struct Square$}
  members {
    { centerX { pos . importantValueX } }
    { centerY { pos . importantValueY } }
    { sideLength { w } }
  }
}
```
TTF Example continued

```cpp
::TV::TTF::RTF::build_struct_transform {
    name "class|struct Rectangle$"
    members {
        { centerX { Square upcast { pos .
            importantValueX } } }
        { centerY { Square upcast { pos .
            importantValueY } } }
        { width { Square upcast { w } } }
        { height { h } }
    }
}
```

C++ Views

Example 1 Transform ...

```cpp
#include <tv_data_display.h>

int TV_ttf_display_type(const Circle* c){
    int retVal = TV_ttf_add_row("centerX", "int",
            &(c->pos.importantValueX));
    if(TV_ttf_ec_ok == retVal)
        retVal = TV_ttf_add_row("centerY", "int",
            &(c->pos.importantValueY));
    if(TV_ttf_ec_ok == retVal)
        return TV_ttf_format_ok;
    return TV_ttf_format_failed;
}
```

```cpp
int TV_ttf_display_type(const Square* s){
    int retVal = TV_ttf_add_row("centerX", "int",
            &(s->pos.importantValueX));
    if(TV_ttf_ec_ok == retVal)
        retVal = TV_ttf_add_row("centerY", "int",
            &(s->pos.importantValueY));
    if(TV_ttf_ec_ok == retVal)
        retVal = TV_ttf_add_row("side length", "int",
            &(s->w));
    if(TV_ttf_ec_ok == retVal)
        return TV_ttf_format_ok_elide;
    return TV_ttf_format_failed;
}
```

```cpp
int TV_ttf_display_type(const Rectangle* r){
    int retVal = TV_ttf_add_row("centerX", "int",
            &(r->pos.importantValueX));
    if(TV_ttf_ec_ok == retVal)
        retVal = TV_ttf_add_row("centerY", "int",
            &(r->pos.importantValueY));
    if(TV_ttf_ec_ok == retVal)
        retVal = TV_ttf_add_row("width", "int", &(r->w));
    if(TV_ttf_ec_ok == retVal)
        retVal = TV_ttf_add_row("height", "int", &(r->h));
    if(TV_ttf_ec_ok == retVal)
        return TV_ttf_format_ok;
    return TV_ttf_format_failed;
}
```
struct Point {
  int importantValueX;
  int importantValueY;
  float meaninglessInternal;
};

class Shape {
public:
  Point getPosition() const;
  void getPosition(const Point& pos) { pos_ = pos; }
private:
  Point pos_;
protected:
  bool isClosed_
  Shape(Point position, int isClosed = true) : pos_(position),
    isClosed_(isClosed) {} Shape() : isClosed_(true){
    pos_.importantValueX = 0;
    pos_.importantValueY = 0;
  }
friend int TV_ttf_display_shape(const Shape* s, const char*,
  const char*);
};

class Circle : public Shape {
private:
  int r; //poorly named radius
public:
  Circle() : Shape(), r(0) {} Circle(int radius) : Shape(), r(radius) {}
  Circle(Point center, int radius) :
    Shape(center), r(radius) {}
friend int TV_ttf_display_type(const Circle* c);
};

class Square : public Shape {
protected:
  int w; //poorly named width
public:
  Square() : Shape(), w(0) {}
  Square(int width) : Shape(), w(width) {}
  Square(Point lowerLeft, int width) :
    Shape(lowerLeft), w(width) {}
friend int TV_ttf_display_type(const Square* s);
};
class Rectangle : public Square {
    private:
        int h;  // poorly named height
    public:
        Rectangle() : Square(), h(0) { }
        Rectangle(int width, int height) :
            Square(width), h(height) {} 
        Rectangle(Point lowerLeft, int width, int height) :
            Square(lowerLeft, width), h(height) { }
        
        friend int TV_ttf_display_type(const Rectangle* s);
};

int TV_ttf_display_type(const Square* s){
    int retVal = TV_ttf_display_shape((const Shape*)s, 
        "lowerLeftX", "lowerLeftY");
    if(TV_ttf_ec_ok == retVal) retVal = 
        TV_ttf_add_row("side length", "int", &s->w);
    if(TV_ttf_ec_ok == retVal) return TV_ttf_format_ok;
    return TV_ttf_format_failed;
}

int TV_ttf_display_type(const Rectangle* r){
    int retVal = TV_ttf_display_shape((const Shape*)r, 
        "lowerLeftX", "lowerLeftY");
    if(TV_ttf_format_ok == retVal) retVal = 
        TV_ttf_add_row("width", "int", &(r->w));
    if(TV_ttf_format_ok == retVal) retVal = 
        TV_ttf_add_row("height", "int", &(r->h));
    if(TV_ttf_ec_ok == retVal) return TV_ttf_format_ok;
    return TV_ttf_format_failed;
}
class FreeFormShape : public Shape {
private:
  std::vector<Point> ptArray_;  
public:
  FreeFormShape() : Shape() { }  
  FreeFormShape(const std::vector<Point>& points,
                bool closedShape)
    : Shape(points[0], closedShape),
      ptArray_(points) { }  
  virtual ~FreeFormShape() {}  
  friend int TV_ttf_display_type(const FreeFormShape* c); 
};

class Circle : public Shape {
private:
  int r; // poorly named radius
public:
  Circle() : Shape(), r(0) { }  
  Circle(int radius) : Shape(), r(radius) { }  
  Circle(Point center, int radius) : Shape(center), r(radius) { }  
  virtual ~Circle() {}  
  friend int TV_ttf_display_circle(const Circle* c);  
  friend int TV_ttf_display_type(const Circle* c); 
};  
// Square and Rectangle are unchanged

#include "complex.h"  
#include <tv_data_display.h>  
#include <typeinfo>  
int TV_ttf_display_shape(const Shape* s, const char* xName, const char* yName){  
  int retVal = TV_ttf_add_row(xName, "int", &(s->pos_.importantValueX));  
  if(TV_ttf_ec_ok == retVal) retVal = TV_ttf_add_row(yName, "int", &(s->
>pos_.importantValueY));  
  return retVal; 
}

int TV_ttf_display_circle(const Circle* c){  
  int retVal = TV_ttf_display_shape((const Shape*)c, "centerX", "centerY");  
  if(TV_ttf_ec_ok == retVal) retVal = TV_ttf_add_row("radius", "int", &(c->r));  
  return retVal; 
}

int TV_ttf_display_type(const Circle* c){  
  int retVal = TV_ttf_display_circle(c);  
  if(TV_ttf_ec_ok == retVal) return TV_ttf_format_ok;  
  return TV_ttf_format_failed; 
}
Example “Complex” Transform continued

```cpp
int TV_ttf_display_type(const Square* s){
    int retVal = TV_ttf_display_shape((const Shape*)s, "lowerLeftX", "lowerLeftY");
    if(TV_ttf_ec_ok == retVal) return TV_ttf_add_row("side length", "int", &(s->w));
    if(TV_ttf_ec_ok == retVal) return TV_ttf_format_failed;
}
```

Example “Complex” Transform continued

```cpp
int TV_ttf_display_type(const Rectangle* r){
    int retVal = TV_ttf_display_shape((const Shape*)r, "lowerLeftX", "lowerLeftY");
    if(TV_ttf_format_ok == retVal) return TV_ttf_add_row("width", "int", &(r->w));
    if(TV_ttf_format_ok == retVal) return TV_ttf_add_row("height", "int", &(r->h));
    if(TV_ttf_ec_ok == retVal) return TV_ttf_format_failed;
}
```

Example “Complex” Transform continued

```cpp
int TV_ttf_display_type(const FreeFormShape* f){
    int retVal = TV_ttf_add_row("closedShape", "bool", &(f->isClosed_));
    char type[32];
    sprintf(type, "Point[%ld]", (long) f->ptArray_.size());
    if(TV_ttf_format_ok == retVal) return TV_ttf_add_row("points", type, &(f->ptArray_.front()));
    if(TV_ttf_format_ok == retVal) return TV_ttf_format_failed;
}
```

Example “Complex” Transform continued

```cpp
int TV_ttf_display_type(const Shape* s){
    char buf[64];
    snprintf(buf, 64, "%s", (const Shape*)s);
    int retVal = TV_ttf_add_row("type", type, s);
}
```
Example “Complex” Transform continued

```c++
int TV_ttf_display_type(const Shape* s) {
...
    if (typeid(*s) == typeid(Rectangle)) {
        snprintf(buf, 64, "Rectangle");
        retVal = TV_ttf_add_row("type", "Rectangle",
                                s);
        if(TV_ttf_ec_ok == retVal)
            return TV_ttf_format_ok_elide;
    }
    if (typeid(*s) == typeid(FreeFormShape)) {
        snprintf(buf, 64, "FreeFormShape");
        retVal = TV_ttf_display_type((const
                                    FreeFormShape*)s);
    }
    if(TV_ttf_ec_ok == retVal)     retVal =
                                    TV_ttf_add_row(buf, TV_ttf_type_ascii_string,
                                                   "type");
    if(TV_ttf_ec_ok == retVal) return
                                    TV_ttf_format_ok;
    return TV_ttf_format_ok;
}
```

C++ Views - caveats

- The data pointer passed to add row must remain in scope beyond the display function
- Don’t do any data allocation in display function
- Objects passed in may not be valid objects, be careful
  - Break point during constructor or destructor, only partially valid
  - Dangling pointer
  - Break before a constructor
- Beware of threading issues
  - Locks may already be acquired
  - Shared data may change if not locked
  - Not all threads are guaranteed to always be stopped
TotalView Customer Support

- Email: tvsupport@roguewave.com
- Use our web site for documentation, demos, FAQs and to contact support

TotalView Customer Support

http://www.roguewave.com

TotalView Documentation
QUESTIONS?