Agenda

• Introduction
• TotalView Debugger
• Demo
• Debugging MPI / OpenMP Hybrid Codes
• Memory Debugging
• Debugging Accelerators and Coprocessors
• Batch Debugging
• Reverse Debugging
• Running on ANL systems
Hybrid and Accelerated Applications

• What do we see
  – NVIDIA Tesla GP-GPU computational accelerators
  – Intel Xeon Phi Coprocessors
  – Complex memory hierarchies (numa, device vs host, etc)
  – Custom languages such as CUDA and OpenCL
  – Directive based programming such as OpenACC and OpenMP
  – Core and thread counts going up

• A lot of complexity to deal with if you want performance
  – C or Fortran with MPI starts to look “simple”
  – Everything is Multiple Languages / Parallel Paradigms
  – Up to 4 “kinds” of parallelism (cluster, thread, heterogeneous, vector)
  – Data movement and load balancing
How does Rogue Wave help?

TotalView debugger

• Troubleshooting and analysis tool
  – Visibility Into
  – Control Over

• Scalability

• Usability

• Advanced features/functionality

• Support for HPC platforms and languages
TotalView Overview
Application Analysis and Debugging Tool: Code Confidently

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

Major Features

- Easy to learn graphical user interface with data visualization
- Parallel Debugging – MPI, Pthreads, OpenMP™, GA, UPC
  - CUDA™, OpenACC®, and Intel® Xeon Phi™ coprocessor
- Low tool overhead resource usage
- Includes a Remote Display Client which frees you to work from anywhere
- Memory Debugging with MemoryScape™
- Deterministic Replay Capability Included on Linux/x86-64
- Non-interactive Batch Debugging with TVScript and the CLI
- TTF & C++View to transform user defined objects
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
What is new in 8.13 and 8.14

• 8.13 (Nov 2013)
  – CUDA 5.0 and 5.5
    • Dynamic Parallelism
  – Xeon Phi Symmetric
  – MemoryScape Xeon Phi support
    • Native and symmetric
  – OS X Mavericks
  – Performance
    • Setting breakpoints
    • Scalable dwhere & dstatus
  – Platform updates

• 8.14 (July 2014)
  – CUDA 6.0
    • Unified Memory
  – Early Access ReplayEngine Save/Load functionality (CLI)
  – STLView for unordered_X
    • GCC only, for now
    • Unordered set/multiset & map/multimap
  – Performance improvements
    • Startup performance
    • Complex C++ codes
    • Handling dlopen()
Multi-phase R&D Projects Underway

- Massive Scalability
  - Collaboration with LLNL and Tri-lab partners
  - Targeting Cray, Blue Gene and Linux Clusters

- Shiny new GUI
  - Sleek, Modern and Fast
  - Configurable
  - Improved Usability
  - Provides aggregation capabilities for big data and scale
  - Leveraging math and stat expertise from IMSL

- Working with customers through early access programs
  - Customer input is key to the success of both programs
• Implement an additional tree-based infrastructure using MRNet
• Parallelize debugger operations into a tree
  – Convert iteration
  – Convert Multicast down, and...
• Push debugger smarts, not the whole debugger, into the back-end
  – Operations previously handled by the debugger front-end must be pushed down into the debugger back-end or target application
  – Operations requiring symbol table information must send it with the request, or handled differently
  – The back-ends must get much smarter, but not much fatter
• Apply “classic optimization” techniques too
  – Caching, hoisting loop invariants, change algorithms/data structures, avoid bottlenecks, etc.
TotalView debugs 786,432 cores. Climb with Rogue Wave towards exacale.
New-Style Root Window (SEA2+)

- A prototype new-style root window w/ "-demo_ui"
- Displays aggregated program information
- Intended to eventually replace the old-style root window
- Menu items that are not yet implemented are disabled

![Diagram of the New-Style Root Window]

- Diving selects a representative of the group and refocuses the process window
- Current aggregations
- Hierarchical groupings planned
Compressed ptlist Syntax

- Aggregation requires a compact process/thread set representation (for both CLI and GUI output)
- General syntax of a ptlist

  \[
  \text{ptlist} : \ pcount ':\ tcount [' prange [ ',', prange ] ... ']
  \]

  \[
  \text{prange} : \ prange '':\ trange
  \]

  \[
  \text{range} : \ rank [ '-\ rank ]
  \mid \ 'p'\ dpid [ '-\ dpid ]
  \]

  \[
  \text{trange} : \ dtid [ '-\ dtid ]
  \]

- Inspired by STAT and previous TotalView implementations

- Example

  \[
  28:28[0-26.1,\ p1.1]
  \]
Call Graph vs. Call Tree (SEA3+)
Please give it try!

• We value your feedback
• Enable MRNet and the demo UI
  – totalview –mrnet –demo_ui …
• Many infrastructure changes are in place already
  – Though not all operations parallelized yet
• User interface changes in prototype phase
  – More improvements coming in existing UI
  – Remaining improvements coming in new UI
• Questions?
Demo
Debugging Hybrid MPI + OMP codes
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**:
  ```cpp
  #include "Circle.h"
  #include "arrays.h"
  #include "main.h"
  int main()
  {
      m_radius = radius;
      myarea = 2 * PI * m_radius * m_radius;
      m_area = area();
  }
  ```
- **Toolbar**: Serves as a single point of control for the process and other related processes.
- **Tabbed Area**: Facilitates management of multiple processes and threads.
Stepping Commands

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

- **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
Setting Breakpoints

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

```c
if (my_ptr == @id) { stop; }
```

Location: `/home/ehinkel/Source/combined.cpp`
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
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.
Diving

Diving on a Common Block in the Stack Frame Pane
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
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.
Array Viewer

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

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

• Connecting to a subset of a job reduces tokens and overhead
• Can change this during a run
• Groups->Subset Attach
Memory Debugging
What Is MemoryScape®?

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

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

Process

User Code and Libraries

Heap Interposition
Agent (HIA)

Malloc API

Allocation Table

Deallocation Table

TotalView
### Enabling Memory Debugging

#### Memory Event Notification

<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>
Memory Event Details Window

Process 1: filterapp-mpi.1 - 1  
Event: Double free - Error: Program attempted to free an already freed block

ID  | Function  | Line # | Source Information
---|-----------|-------|-------------------
D-108 | free      |       | 104malloc_wrappers可以更好block.c
    | double_free | 74    | main.c
    | main       | 287   | main.c
    | _libc_start_main | libso.c
    | _start     |       | filterapp-mpi

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

73  // Show that the deallocation stack is available now
74  junk = 0;
75  
76  // Now release the memory the second time - illegal
77  #ifdef USEMPI
78  if( rank == 1 )
79     #endif
80  free( p );
81
Generate:
Memory File

Close           View in Block Properties window           Help
Heap Graphical View
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

Expression: `addr`  Address: 0xbfffd1f4
Type: int *

Value
0x80496d0 (Dangling) -> 0x00000000 (0)

Expression: `misaddr`  Address: 0xbfffd1f0
Type: int *

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
Debugging Accelerators and Coprocessors
TotalView for the NVIDIA® GPU Accelerator

- NVIDIA Kepler
- NVIDIA CUDA 5.0, 5.5, and 6.0 (New in 8.14)
  - With support for Unified Memory
- Cray CCE OpenACC
- Features and capabilities include
  - Support for dynamic parallelism
  - Support for MPI based clusters and multi-card configurations
  - Flexible Display and Navigation on the CUDA device
    - Physical (device, SM, Warp, Lane)
    - Logical (Grid, Block) tuples
  - CUDA device window reveals what is running where
  - Support for types and separate memory address spaces
  - Leverages CUDA memcheck

© 2014 Rogue Wave Software, Inc. All Rights Reserved
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.

@TEMP@CUDA@.tx_cuda_matmul.e4974cfd (1,-1) has loaded a CUDA GPU image. Stop so you can set breakpoints?

@TEMP@CUDA@.tx_cuda_matmul.e4974cfd

☐ Don't ask this question again.

Yes

No
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”
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;`
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

Example of divergent GPU threads

Different PC for two groups of lanes

State of lanes inside warp
TotalView for the Intel® Xeon Phi™ coprocessor

Supports All Major Intel Xeon Phi Coprocessor Configurations

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

User Interface

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

Memory Debugging

- Both native and symmetric mode
Batch Debugging
TVScript Overview

• Gives you non-interactive access to TotalView’s capabilities
• Useful for
  – Debugging in batch environments
  – Watching for intermittent faults
  – Parametric studies
  – Automated testing and validation
• TVScript is a script (not a scripting language)
  – It runs your program to completion and performs debugger actions on it as you request
  – Results are written to an output file
  – No GUI
  – No interactive command line prompt
• A “better” printf()
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 = 0x0000000a (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, rzUnderrun, 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
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
Reverse Debugging
Deterministic Replay Debugging

- Reverse Debugging: Radically simplify your debugging
  - Captures and Deterministically Replays Execution
  - Not just “checkpoint and restart”
  - Eliminate the Restart Cycle and Hard-to-Reproduce Bugs
  - Step Back and Forward by Function, Line, or Instruction

- Specifications
  - A feature included in TotalView on Linux x86 and x86-64
    - No recompilation or instrumentation
    - Explore data and state in the past just like in a live process, including C++View transformations
  - Replay on Demand: enable it when you want it
  - Supports MPI on Ethernet, Infiniband, Cray XE Gemini
  - Supports Pthreads, and OpenMP
  - New: Save / Load Replay Information (CLI only)
Running on ALCF systems
Debugging on BG/Q with Totalview 8.14.0

Load .totalview in your .soft
Use the remote display client
Just add totalview -args before runjob

- totalview -args runjob --block $COBALT_PARTNAME -p 16 : demoMpi
- Add options from ~/chrisg/ATPESC/example.tvdrc to your .totalview/.tvdrc
to use the MRNet early access
- For memory debugging (from documentation):
  - Link statically as –L<path> -ltvheap -Wl,rpath,<path>
  - Link dynamically as –L<path> -Wl,@<path>/tvheap_bgqs.ld
- TotalView 8.14 will be available on Mira, Vesta, Cetus and Tukey for the
duration of the training.
Thanks!

• To learn more / sign up for the Scalability Early Experience Program please contact me: chris.gottbrath@roguewave.com

• Visit the website
  – Videos
  – Documentation
  – Sign up for an evaluation
  – Contact customer support & post on the user forum