Facing the challenges of

Programming weather, climate, and earth-system models on heterogeneous multi-core platforms

or

New Approaches To
Debugging Complex Codes

September 7-8, 2011 at the National Center for Atmospheric Research in Boulder, Colorado

Ed Hinkel, Sales Engineer
Rogue Wave Software
Agenda

• Introduction – Rogue Wave

• TotalView

• Approaching the Debugging Challenge
  1  TVScript – Automate Your Debugging
  2  MemoryScape – Don’t Forget the Memory!
  3  Replay Engine – Shift into Reverse if You Want!
  4  CUDA – Accelerate your Development Schedule
  5  Performance – Memory Cache Usage Efficiency
Rogue Wave Today

The largest independent provider of cross-platform software development tools and embedded components for the next generation of HPC applications

Visual Numerics
Leader in embeddable math and statistics algorithms and visualization software for data-intensive applications.

Acumem
Leading provider of intelligent software technology which analyzes and optimizes computing performance in single and multicore environments.

TotalView
Industry-leading interactive analysis and debugging tools for the world's most sophisticated software applications.
Rogue Wave Product Offerings

IMSL
SourcePro C++
PV-WAVE
PyIMSL
ThreadSpotter
TotalView
ReplayEngine
MemoryScape
What is TotalView?

A comprehensive debugging solution for demanding applications on parallel, multi-core and hybrid systems

- Wide compiler & platform support
  - C, C++, Fortran 77 & 90, UPC
  - Unix, Linux, OS X
- Handles Concurrency
  - Multi-threaded Debugging
  - Multi-process Debugging
- Integrated Memory Debugging
- Reverse Debugging available
- Supports Multiple Usage Models
  - Powerful and Easy GUI – Highly Graphical
  - CLI for Scripting
  - Long Distance Remote Debugging
  - Unattended Batch Debugging
Facing the Debugging Challenge

# 1
Unattended Debugging

tvscript
Unattended Debugging

- tvscript provides for unattended, straightforward TotalView batch debugging
  - As an adjunct to interactive debugging
  - Usable whenever jobs need to be submitted or batched
  - Provides a tool more powerful and flexible than Printf-style debugging
  - Can be used to automate test/verify environments

Think of tvscript as “Printf on steroids”!
Debugging with TVScript

- TVScript
  - Define events
    - Breakpoints, memory errors, etc..
  - Provide actions to take in response to these events
    - Print variables or create memory reports
  - Run an MPI or serial program toward completion
    - With no user interaction

- More powerful and flexible than Printf-style debugging
  - Use to prepare and guide interactive debugging
  - Use whenever jobs need to be submitted into a managed environment
  - Can be used to automate test/verify environments
Unattended Debugging

- Using tvscript, multiple debugging sessions can be run without the need for recompiling, unlike with printf

- A single compile is all that’s needed, i.e.,
  - gcc -g -o server-dbgs server.c

- tvscript syntax:
  - tvscript [ options ] [ filename ] [ -a program_args ]
TVscript uses a simple, Event/Action interface

<table>
<thead>
<tr>
<th>Typical Events</th>
<th>Typical Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Action_point</td>
<td>• Display_backtrace [-level level-num]</td>
</tr>
<tr>
<td>• Any_memory_event</td>
<td>• List_leaks</td>
</tr>
<tr>
<td>• Guard_corruption_error</td>
<td>• Save_memory</td>
</tr>
<tr>
<td></td>
<td>• Print [-slice {slice_exp} {variable</td>
</tr>
</tbody>
</table>

- A single compile is all that’s needed, i.e.,
  
  gcc -g -o server-dbgs server.c

- tvscript syntax:
  
  tvscript [ options ] [ filename ] [ -a program_args ]
Unattended Debugging with Tvscript

Example
The following tells tvscript to report the contents of the foreign_addr structure each time the program gets to line 85
-create_actionpoint "#85=>print foreign_addr"

Typical output sample with tvscript:

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! Print
! 
! Process:
!   ./server (Debugger Process ID: 1, System ID: 12110)
! Thread:
!     Debugger ID: 1.1, System ID: 3083946656
! Time Stamp:
!   06-26-2008 14:04:09
! Triggered from event:
!   actionpoint
! Results:
!   foreign_addr = {
!       sin_family = 0x0002 (2)
!       sin_port = 0x1fb6 (8118)
!       sin_addr = {
!         s_addr = 0x6658a8c0 (1717086400)
!       }
!       sin_zero = ""
!   }
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
Summary

- Provides process, thread and timestamp information
- Provides a single output file, even with multiple processes
- Provides many event/action descriptions, including memory debugging events
- Supports external script files, utilizing TCL within a CLI file
- Allows the generation of even more complex actions
Facing the Debugging Challenge

#2
Memory Debugging

Don’t Forget the Memory!
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
  - Over running array bounds
Why Are Memory Bugs Different?

• 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 patterns change
  • Even the fallout is not always fatal. It can result in data corruption which may or may not result in a subsequent crash
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
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/reallocation notification
- Memory Corruption Detection - Guard Blocks
- Memory Comparisons between processes
- Collaboration features
Enabling Memory Debugging

**CONFIGURABILITY**

Memory Event Notification

<table>
<thead>
<tr>
<th>Event</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Allocation Failed</td>
<td>An allocation call failed or the address returned is NULL: probably out of memory</td>
</tr>
<tr>
<td>Double allocation</td>
<td>Allocator returned a block already in use: heap may be corrupted</td>
</tr>
<tr>
<td>Double free</td>
<td>Program attempted to free an already freed block</td>
</tr>
<tr>
<td>Free interior pointer</td>
<td>Program attempted to free a block incorrectly, via an address in the middle of the block</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>Program attempted to free an address not in the heap</td>
</tr>
<tr>
<td>Guard corruption</td>
<td>The guard areas around a block have been overwritten, suggesting a bounds error</td>
</tr>
<tr>
<td>Invalid aligned allocation request</td>
<td>Program supplied an invalid alignment argument to the heap manager</td>
</tr>
<tr>
<td>Misaligned allocation</td>
<td>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>Program attempted to reallocate an address not in the heap</td>
</tr>
<tr>
<td>Termination notification</td>
<td>The target is terminating, memory analysis can be performed</td>
</tr>
<tr>
<td>Unknown error</td>
<td>Some unknown error has occurred</td>
</tr>
</tbody>
</table>

Help  OK  Cancel
Heap Graphical View
Leak Detection

- 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
Memory Corruption Detection (Guard Blocks)
Memory Usage Statistics
Visualize Your Program's Environment…
Facing the Debugging Challenge

#3
ReplayEngine

Reverse Debugging
What is ReplayEngine?

A separately licensed add-on product that providing reverse debugging

- **Enhances debugging experience**
  - Add-on to TotalView
- **Captures execution history**
  - Record all external input to program
  - Records internal sources of non-determinism
- **Replays execution history**
  - Examine any part of the execution history
  - Step as easily back through code as you do forwards
  - Jump to points of interest
- **Simple extension to TotalView**
  - No recompilation or instrumentation
  - The user just says where they want to go
  - Explore data and state in the past just like a live process
- **Supported on Linux x86 and x86-64**
- **Supports MPI, Pthreads, and OpenMP**
ReplayEngine: Debug in Forward and Reverse!

ReplayEngine's deterministic replay capability records the execution history of your program and allows bi-directional diagnosis and debugging with TotalView.

- Step freely forward and backwards through a program’s execution
- Use Breakpoints and Watchpoints from either direction
- Examine historical data and execution paths deterministically
- Works with integrated memory debugging
- Allows for “rolling history capture” of long-running programs
  - Stop at out of memory
  - Set maximum history buffer size

ReplayEngine is a separately licensed product add-on that extends the capabilities of TotalView on Linux-86 and Linux-86-64 machines.
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
- 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

[Logo: Rogue Wave Software]
ReplayEngine

No Replay? (or not running)

No Active Buttons!
Facing the Debugging Challenge

#4
CUDA Debugging
CUDA Debugging TotalView

- **Characteristics**
  - Debugging of application running on the GPU device (not in an emulator)
  - Full visibility of both Linux threads and GPU device threads
    - Device threads shown as part of the parent Unix process
    - Correctly handle all the differences between the CPU and GPU
  - Fully represent the hierarchical memory
    - Display data at any level (registers, local, block, global or host memory)
    - Making it clear where data resides with type qualification
  - Thread and Block Coordinates
    - Built in runtime variables display threads in a warp, block and thread dimensions and indexes
    - Displayed on the interface in the status bar, thread tab and stack frame
  - Device thread control
    - Warps advance Synchronously
  - Handles CUDA function inlining
    - Step in to or over inlined functions
  - Reports memory access errors
    - CUDA memcheck
  - Can be used with MPI
A Linux-x86_64 CUDA process consists of:

- A Linux process address space, containing:
  - A Linux executable and a list of Linux shared libraries.
- A collection of Linux threads, where a Linux thread:
  - Is assigned a positive debugger thread ID.
- A collection of CUDA threads, where a CUDA thread:
  - Is assigned a negative debugger thread ID.
  - Has its own separate address space
Storage Qualifiers

- **Denotes location in hierarchical memory**
  - Part of the type – using “@” notation
  - Each memory space has a separate address space so 0x00001234 could refer to several places

<table>
<thead>
<tr>
<th>Storage Qualifier</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>@parameter</td>
<td>Address is an offset within parameter storage.</td>
</tr>
<tr>
<td>@local</td>
<td>Address is an offset within local storage.</td>
</tr>
<tr>
<td>@shared</td>
<td>Address is an offset within shared storage.</td>
</tr>
<tr>
<td>@constant</td>
<td>Address is an offset within constant storage.</td>
</tr>
<tr>
<td>@global</td>
<td>Address is an offset within global storage.</td>
</tr>
<tr>
<td>@register</td>
<td>Address is a PTX register name.</td>
</tr>
</tbody>
</table>

- **Used throughout expression system**
  - You can cast to switch between different spaces
Debugging CUDA

- GPU focus thread selector for changing the block (x,y) and thread (x,y,z) indexes of the CUDA thread.
- CUDA grid and block dimensions, lanes/warp, warps/SM, SMs, etc.
- Parameter, register, local and shared variables.
- CUDA host threads have a positive TotalView thread ID.
- CUDA GPU threads have a negative TotalView thread ID.
- Select a line number in a box to plant a breakpoint.
- Thread (x,y,z)
**GPU Device Status Display**

- 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

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Device 0/3</td>
<td></td>
</tr>
<tr>
<td>Device Type</td>
<td>gf100</td>
</tr>
<tr>
<td>Lanes</td>
<td>32</td>
</tr>
<tr>
<td>SM 2/1</td>
<td></td>
</tr>
<tr>
<td>Valid Warps</td>
<td></td>
</tr>
<tr>
<td>Warp 00/48</td>
<td>Block (0,0,0)</td>
</tr>
<tr>
<td>Lane 00/32</td>
<td>Thread (0,0,0)</td>
</tr>
<tr>
<td>LPC</td>
<td>0000000019aa94f0</td>
</tr>
<tr>
<td>Lane 01/32</td>
<td>Thread (1,0,0)</td>
</tr>
<tr>
<td>LPC</td>
<td>0000000019aa94f0</td>
</tr>
<tr>
<td>Lane 02/32</td>
<td>Thread (2,0,0)</td>
</tr>
<tr>
<td>LPC</td>
<td>0000000019aa94f0</td>
</tr>
<tr>
<td>Lane 03/32</td>
<td>Thread (3,0,0)</td>
</tr>
<tr>
<td>LPC</td>
<td>0000000019aa94f0</td>
</tr>
<tr>
<td>Lane 04/32</td>
<td>Thread (3,0,0)</td>
</tr>
<tr>
<td>LPC</td>
<td>0000000019aa94f0</td>
</tr>
<tr>
<td>Lane 05/32</td>
<td>Thread (5,0,0)</td>
</tr>
<tr>
<td>LPC</td>
<td>0000000019aa94f0</td>
</tr>
<tr>
<td>Lane 06/32</td>
<td>Thread (6,0,0)</td>
</tr>
<tr>
<td>LPC</td>
<td>0000000019aa94f0</td>
</tr>
<tr>
<td>Lane 07/32</td>
<td>Thread (7,0,0)</td>
</tr>
<tr>
<td>LPC</td>
<td>0000000019aa94f0</td>
</tr>
<tr>
<td>Lane 08/32</td>
<td>Thread (8,0,0)</td>
</tr>
<tr>
<td>LPC</td>
<td>0000000019aa94f0</td>
</tr>
<tr>
<td>Lane 09/32</td>
<td>Thread (9,0,0)</td>
</tr>
<tr>
<td>LPC</td>
<td>0000000019aa94f0</td>
</tr>
<tr>
<td>Valid/Active/Divergent</td>
<td>0000003f, 0000003f, 00000003</td>
</tr>
<tr>
<td>SMs</td>
<td></td>
</tr>
<tr>
<td>SMs Type</td>
<td>sm_20</td>
</tr>
<tr>
<td>Warps</td>
<td></td>
</tr>
<tr>
<td>Warps Type</td>
<td></td>
</tr>
<tr>
<td>Device 1/3</td>
<td></td>
</tr>
<tr>
<td>Device Type</td>
<td>gt200</td>
</tr>
<tr>
<td>Lanes</td>
<td>32</td>
</tr>
<tr>
<td>SM Type</td>
<td>sm_13</td>
</tr>
</tbody>
</table>
Facing the Debugging Challenge

#5
ThreadSpotter cache memory optimization tool

- Analyzes memory bandwidth and latency, data locality and thread communications
- Identifies specific issues and pinpoints troublesome areas in source code
- Provides guidance towards a resolution
- Increases productivity for experts and non-experts
Example: Why do more cores = less performance?

Poor parallelism?
Actually, this program is “embarrassingly parallel”
Poor memory usage $\Rightarrow$ super-linear slowdown
The Same Application Optimized

Optimization can be rewarding, but costly…

- Often requires expert knowledge
- Typically weeks of wading through performance data

➤ ThreadSpotter’s one-click advice: Change one line
How is the silicon used?
A rule of thumb

<table>
<thead>
<tr>
<th>Memory system level</th>
<th>Relative latency</th>
</tr>
</thead>
<tbody>
<tr>
<td>L1 cache</td>
<td>1x</td>
</tr>
<tr>
<td>Higher cache levels</td>
<td>10x</td>
</tr>
<tr>
<td>Main memory</td>
<td>100x</td>
</tr>
</tbody>
</table>

Source: AMD, *Michael Wall*
Simple modifications can make a big difference

<table>
<thead>
<tr>
<th>Program A</th>
<th>Program B</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>struct DATA</code></td>
<td><code>struct DATA</code></td>
</tr>
<tr>
<td>`{</td>
<td>`{</td>
</tr>
<tr>
<td>int a;</td>
<td>int a;</td>
</tr>
<tr>
<td>int b;</td>
<td>int b;</td>
</tr>
<tr>
<td>int c;</td>
<td>`int c;</td>
</tr>
<tr>
<td>int d;</td>
<td>`int d;</td>
</tr>
<tr>
<td>};</td>
<td>`};</td>
</tr>
<tr>
<td>DATA * pMyData;</td>
<td>DATA * pMyData;</td>
</tr>
<tr>
<td>for (long i=0; i&lt;10<em>1024</em>1024; i++) {</td>
<td>for (long i=0; i&lt;10<em>1024</em>1024; i++) {</td>
</tr>
<tr>
<td>pMyData[i].a = pMyData[i].b;</td>
<td>pMyData[i].a = pMyData[i].b;</td>
</tr>
<tr>
<td>}</td>
<td>}</td>
</tr>
</tbody>
</table>

Partially Used Structures
Partially Used Structures

Defined data structure includes a, b, c, d... but only uses a & b

Redefined data structure includes a, b, c, d

50%

100%
What’s the difference?
Alignment Characteristics

44%

75%
Test 4: Inefficient Loop Nesting

Explanation
A One-Click Report Generation

Click this button to create a report

Application to run

Input arguments

Working dir (where to run the app)

(Limit, if you like, data gathered here, e.g., start gathering after after 10 sec. and stop after 10 sec.)

Cache size of the target system for optimization (e.g., L1 or L2 size)
Acumem SlowSpotter™

Source:
C, C++, Fortran, OpenMP…

Mission:
Find the SlowSpots™
Assess their importance
Enable for non-experts to fix them
Improve the productivity of experts
/* Unoptimized Array Multiplication: x = y * z    N = 1024 */
for (i = 0; i < N; i = i + 1)
    for (j = 0; j < N; j = j + 1)
    {
        r = 0;
        for (k = 0; k < N; k = k + 1)
            r = r + y[i][k] * z[k][j];
        x[i][j] = r;
    }

/* Unoptimized Array Multiplication: x = y * z    N = 1024 */
for (i = 0; i < N; i = i + 1)
    for (j = 0; j < N; j = j + 1)
    {
        r = 0;
        for (k = 0; k < N; k = k + 1)
            r = r + y[i][k] * z[k][j];
        x[i][j] = r;
    }
Acumem ThreadSpotter

Acumem ThreadSpotter is a tool to quickly analyze an application for a range of performance problems, particularly related to multithread optimization.

Read more... Manual

Open the Report

Next Steps

The prepared report is divided into sections:

- Select the tab Summary to view global statistics for the entire application.
- Select the tabs Bandwidth Issues, Latency Issues and MT Issues to browse through the detected problems.
- Select the tab Loops to browse through statistics for detected problems and loop by loop.

The Issue and Source windows contain details and a related source code for the detected problems.

Resources

Manual

Table of Contents
Optimization Workflow
Reading the Report
Acumem Web Site

Issue Reference

Tutorials
### List of bad loops

<table>
<thead>
<tr>
<th>Loop</th>
<th>% of misses</th>
<th>% of fetches</th>
<th>Utilization</th>
<th>Issues</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>85.8%</td>
<td>62.3%</td>
<td>17.7%</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>9.9%</td>
<td>7.7%</td>
<td>22.8%</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0.0%</td>
<td>6.1%</td>
<td>34.1%</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>0.0%</td>
<td>4.4%</td>
<td>23.7%</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>0.0%</td>
<td>4.2%</td>
<td>36.9%</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>0.0%</td>
<td>4.2%</td>
<td>25.1%</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>4.2%</td>
<td>3.2%</td>
<td>18.4%</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>0.0%</td>
<td>1.1%</td>
<td>23.5%</td>
<td></td>
</tr>
</tbody>
</table>

### Spotting the crime

```c
/* Compute F1 - Q values */
tnorm = sqrt((double)tnorm);
for (ej=0; ej<numfile; ej++)
{
  fl_layer[tj].Q = fl_layer[tj].P;
}
```

### Explaining what to do

**Loop Focus Tab**
- Cache line utilization
- Inefficient loop nesting
- Random access pattern

**Loop 1**
- Loop statistics
- Loop instructions
- Instruction groups in this loop, summary of issues
  - Instruction group 1
  - Instruction group 2
  - Instruction group 3

**Copyright (c) 2007 Acumen AB. All rights reserved.**
Thank You