Getting Started with TotalView and CUDA

Chris Gottbrath, Principal Product Manager
Ed Hinkel, Senior Sales Engineer
Agenda

• Rogue Wave Software
• CUDA Challenges
• TotalView Debugger
• TotalView for CUDA
• Follow up
Rogue Wave Today

- **History**
  - Founded: 1989
  - Acquired by Battery Ventures: 2007
  - Acquired:
    - Visual Numerics: 2009
    - TotalView Technologies: 2010
    - Acumem: 2010

- **Customers**
  - 3,000+ in 36 countries
  - Financial services, telecoms, oil and gas, government and aerospace, research and academic

- **Pioneers in C++/object-oriented development**
- **Leading the way in cross-platform, parallel development**
Representative Customers
Rogue Wave Solution Portfolio

- Development
  - PyIMSL Studio
  - PV-Wave
  - SourcePro
  - IMSL Libraries

- Debugging
  - TotalView
  - ReplayEngine
  - MemoryScape

- Optimization
  - ThreadSpotter
GPU architecture

Used in conjunction with conventional CPUs
  Acts as an accelerator to a host process
  Or, perhaps the host processor acts to support the GPU

Distinct architecture
  Distinct processor architecture from the CPU

Many more cores than an SMP
  Multiple streaming multiprocessors
  Potentially 10k+ thread contexts
Programming for the GP-GPU

• **CUDA**
  – Function-like kernels are written for calculations to be performed on the GPU
    • Data parallel style, one kernel per unit of work
  – Presents a 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 to and from device memory using streams
Programming challenges

- Coordinating CPU code + device code
- Understanding what is going on in each kernel
  - Exceptions
- Understanding memory usage
- Understanding performance characteristics
What is TotalView?

• **Application Analysis and Debugging Tool: Code Confidently**
  - Debug and Analyze C/C++ and Fortran on Linux, Unix or Mac OS X
  - Laptops to supercomputers (BG, Cray)
  - 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 Support available
  - Includes a Remote Display Client freeing users to work from anywhere
  - Includes Memory Debugging with MemoryScape
  - Reverse Debugging available with ReplayEngine
  - Includes Batch Debugging with TVScript and the CLI
How can TotalView help you?

Effective Debugging requires the capability to control and examine specific instances of program execution in detail.

- **Threads and/or MPI**
  - When you have
    - Deadlocks and hangs
    - Race conditions
  - It provides
    - Asynchronous thread control
    - Powerful group mechanism

- **Fortran and/or C++**
  - Complex data structures
    - Diving and recursive dive
  - STL Collection Classes
    - STLView
  - Rich class hierarchies
    - Powerful type-casting features

- **Memory Analysis**
  - Leaks and Bounds Errors
    - Automatic error detection tools
  - Out of Memory Errors
    - Analysis of heap memory usage by file function and line

- **Data Analysis**
  - Numerical errors
    - Extensible data visualization
    - Slicing and filtering of arrays
    - Powerful expression system
    - Conditional watchpoints

TotalView provides an answer to the question: “What is my program really doing?”
TotalView Debugging Ecosystem

Debugging with TotalView

Reverse Debugging with ReplayEngine

Memory Debugging with MemoryScape

Remote Display Window

Easy Secure Fast

Batch Debugging with TVScript

Optimizing with ThreadSpotter

---

<table>
<thead>
<tr>
<th>Action</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Print</td>
<td></td>
</tr>
<tr>
<td>Process</td>
<td></td>
</tr>
<tr>
<td></td>
<td>.server (Debugger Process ID: 1, System ID: 12110)</td>
</tr>
<tr>
<td>Thread</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Debugger ID: 1.1, System ID: 3083946656</td>
</tr>
<tr>
<td>Time Stamp</td>
<td>06-26-2008 14:04:09</td>
</tr>
<tr>
<td>Triggered from event</td>
<td>actionpoint</td>
</tr>
<tr>
<td>Results</td>
<td></td>
</tr>
<tr>
<td></td>
<td>foreign_addr = {</td>
</tr>
<tr>
<td></td>
<td>sin_family = 0x0002 (2)</td>
</tr>
<tr>
<td></td>
<td>sin_port = 0x1fb6 (8118)</td>
</tr>
<tr>
<td></td>
<td>sin_addr = {</td>
</tr>
<tr>
<td></td>
<td>s_addr = 0x6658a8c0 (1717086400)</td>
</tr>
<tr>
<td></td>
<td>}</td>
</tr>
</tbody>
</table>

---

Optimizing with ThreadSpotter
Starting TotalView

Open a Core File
TotalView Startup
TotalView Root Window

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

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

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

A single point of control for the process and other related processes.
Tabbed Pane

Action Points Tab
all currently defined action points

Processes Tab
all current processes

Threads Tab:
all current threads, ID’s, Status
Stepping Commands

Based on PC location
Diving on Variables

Example: Dive on Variable “j” from Stack Frame or Source Panes
Viewing Arrays

Data Arrays

Structure Arrays
Slicing Arrays

Slice notation is \([\text{start}:\text{end}:\text{stride}]\)
Filtering Arrays
Multi-Dimensional Array Viewer

- See your arrays on a “Grid” display
- 2-D, 3-D... N-D
- Arbitrary slices
- Specify data representation
- Windowed data access
  - Fast

![Multi-Dimensional Array Viewer](image-url)
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
- $\text{visualize()}$ is a directive in the expression system, and can be used in evaluation point expressions.
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
  - Support for CUDA C++
  - Reports memory access errors
  - Handles CUDA exceptions and assert
  - Full Multi-Device Support
  - Can be used with MPI

- TV 8.9.2 supports CUDA 3.2 and 4.0
- TV 8.10 will support CUDA 4.1
TV 8.10 support for CUDA 4.1 specific features

- Works with the CUDA 4.1 SDK and Runtime
  - New Compiler Front End
  - New Debug API
- Support for no copy pinned memory
  - This was broken at the driver level in 4.0
- New support for CUDA device assertions
- New support for multiple CUDA contexts from the same process on the same device
- Support for CUDA on the Cray XK environment
Starting TotalView

- You can debug the CUDA host code using the normal TotalView commands and procedures

When a new kernel is loaded you get the option of setting breakpoints.
TotalView CUDA Debugging Model

Linux-x86_64 CUDA process (1)

- Linux process address space
  - Linux thread (1.1)
  - Linux executable and shared libraries

CUDA thread (1.1)
- Device address space
- GPU focus thread
- GPU image

CUDA thread (1.2)
- Device address space
- GPU focus thread
- GPU image
Debugging CUDA

CUDA host threads have a positive TotalView thread ID
CUDA GPU threads have a negative TotalView thread ID
GPU focus thread selector for changing the block (x,y) and thread (x,y,z) indexes of the CUDA thread
Select a line number in a box to plant a breakpoint

Thread (x,y,z)

Block (x,y,z)
Running to a Breakpoint in the GPU code

- GPU focus thread logical coordinates
- Stack backtrace (3.2) and inlined functions (3.1)
- PC arrow for the warp
- CUDA grid and block dimensions, lanes/warp, warps/SM, SMs, etc.
- Parameter, register, local and shared variables
- Dive on a variable name to open a variable window
Stepping GPU Code

• single-step operation advances all of the GPU hardware threads in the same warp

• To advance the execution of more than one warp, you may either:
  – set a breakpoint and continue the process, or
  – select a line number in the source pane and select “Run To”.
## 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**

### Table Example

<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>00000000000000001</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>00000000015aa94f0</td>
</tr>
<tr>
<td>Lane 01/32</td>
<td>Thread (1,0,0)</td>
</tr>
<tr>
<td>LPC</td>
<td>00000000015aa94f0</td>
</tr>
<tr>
<td>Lane 02/32</td>
<td>Thread (2,0,0)</td>
</tr>
<tr>
<td>LPC</td>
<td>00000000015aa94f0</td>
</tr>
<tr>
<td>Lane 03/32</td>
<td>Thread (3,0,0)</td>
</tr>
<tr>
<td>LPC</td>
<td>00000000015aa94f0</td>
</tr>
<tr>
<td>Lane 04/32</td>
<td>Thread (4,0,0)</td>
</tr>
<tr>
<td>LPC</td>
<td>00000000015aa94f0</td>
</tr>
<tr>
<td>Lane 05/32</td>
<td>Thread (5,0,0)</td>
</tr>
<tr>
<td>LPC</td>
<td>00000000015aa94f0</td>
</tr>
<tr>
<td>Lane 06/32</td>
<td>Thread (6,0,0)</td>
</tr>
<tr>
<td>LPC</td>
<td>00000000015aa94f0</td>
</tr>
<tr>
<td>Lane 07/32</td>
<td>Thread (7,0,0)</td>
</tr>
<tr>
<td>LPC</td>
<td>00000000015aa94f0</td>
</tr>
<tr>
<td>Lane 08/32</td>
<td>Thread (8,0,0)</td>
</tr>
<tr>
<td>LPC</td>
<td>00000000015aa94f0</td>
</tr>
<tr>
<td>Lane 09/32</td>
<td>Thread (9,0,0)</td>
</tr>
<tr>
<td>LPC</td>
<td>00000000015aa94f0</td>
</tr>
<tr>
<td>-SM Type</td>
<td>sm_20</td>
</tr>
<tr>
<td>-SMs</td>
<td>14</td>
</tr>
<tr>
<td>-Warps</td>
<td>48</td>
</tr>
</tbody>
</table>

---

**Example of Divergent GPU threads**

- Different PC for two groups of Lanes
- State of Lanes inside the warp
## CUDA Memory Types

<table>
<thead>
<tr>
<th>Memory</th>
<th>Scope</th>
<th>Locality</th>
</tr>
</thead>
<tbody>
<tr>
<td>Global</td>
<td>Device</td>
<td>External</td>
</tr>
<tr>
<td>Shared</td>
<td>Block</td>
<td>Chip</td>
</tr>
<tr>
<td>Local</td>
<td>Thread</td>
<td>Chip</td>
</tr>
<tr>
<td>Constant</td>
<td>Device</td>
<td>Chip (Cache)</td>
</tr>
<tr>
<td>Texture</td>
<td>Device</td>
<td>Chip (Cache)</td>
</tr>
<tr>
<td>Register</td>
<td>Thread</td>
<td>Chip</td>
</tr>
</tbody>
</table>
• Hierarchical memory with many layers
  – Local (thread)
  – Shared (block)
  – Global (GPU)
  – System (host)
TotalView Type Storage Qualifiers

@parameter Address is an offset within parameter storage.

@local Address is an offset within local storage.

@shared Address is an offset within shared storage.

@constant Address is an offset within constant storage.

@global Address is an offset within global storage.

@register Address is a PTX register name.
HRL Case Study

- Center for Neural and Emergent Systems at HRL
- Using a CUDA accelerated cluster to model the brain

- “In the first full day of using TotalView, we were quickly able to solve the bug that had us stumped for weeks. With TotalView we were able to step into a specific thread, and then into specific CUDA kernels to identify what went wrong. We could resolve the bugs quickly, and focus our development effort on adding features.”

- “We noticed a dramatic drop in our development cycle – what used to take us more than two weeks to develop and fully test now takes less than one week. By scaling down the development cycle we were able to add more features, even going beyond the requirements of our release cycle. Most important, we were able to focus on the performance of our code, resulting in much better utilization of our existing hardware and allowing us to scale past 100 GPUs.”

- For more information look at the HRL case study on the following page
  
  http://www.roguewave.com/resources/case-studies.aspx
Thanks!

• Contact me
  – Chris.Gottbrath@RogueWave.com
  – Ed.Hinkel@RogueWave.com

• or for more information

Check out: www.roguewave.com

Email: sales@roguewave.com
Register for GPU Tech Conference 2012
May 14-17 | San Jose, CA

By the numbers...

- 4 full days
- 1000s of developers, computational scientists, and researchers
- 3 keynotes
- 275 sessions
- 30 topic areas
- 150 research posters
- 2 superb co-located events - Los Alamos HPC Symposium & InPar 2012
- 1 Emerging Companies Summit
- Limitless opportunities for formal and informal networking

Register at www.gputechconf.com
Thank You