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

September 19-20, 2013 at the National Center for Atmospheric Research in Boulder, Colorado

Debugging with Accelerated Systems
Heterogeneous Multi-Core Debugging

Ed Hinkel
Senior Sales Engineer
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
  - CUDA GPU, Intel Xeon Phi

- **Handles Concurrency**
  - Multi-threaded Debugging
  - Parallel Debugging
    - MPI, PVM, OpenMP
  - Remote and Client/Server Debugging

- **Integrated Memory Debugging**

- **Reverse Debugging**

- **Variety of Usage Models**
  - Powerful and Easy GUI / Visualization
  - CLI for Scripting
  - Long Distance Remote Debugging
  - Unattended Batch Debugging
TotalView Debugging Ecosystem

Reverse Debugging with ReplayEngine
- Captures execution history
- Replays execution history
- Enable ‘on Demand’
- Step backwards!

Debugging with TotalView
- Parallel debugging
- Accelerator & coprocessor debugging
- Wide compiler and platform coverage
- Work Graphically
- Troubleshoot your most complex codes
- Develop Code Confidently!

Memory Debugging with MemoryScape
- Graphical View of Heap Memory
- Low Overhead
- Detect: Leaks, Buffer over/underflow
- MPI memory debugging

Remote Display Debugging with RDC
- Easy, Secure, Fast

Batch Debugging with TVScript
- Unattended TotalView debugging
-PRINTF on Steroids!

The Debugger of Choice for HPC and Enterprise
What Is ReplayEngine?

Reverse Debugging Tool: Radically simplify your debugging

- Captures and Deterministically Replays Execution
- Eliminate the Restart Cycle and Hard-to-Reproduce Bugs
- Step Back and Forward by Function, Line, or Instruction
- Simple extension to TotalView
  - No recompilation or instrumentation
  - Explore data and state in the past just like in a live process
- Supported on Linux x86 and x86-64
- Supports MPI, Pthreads, and OpenMP
Reverse Debugging

**Replay Engine – Total Execution Control**

- **Step forward over functions**
- **Step forward into functions**
- **Advance forward out of current Function, after the call**
- **Advance forward to selected line**
- **Run forward**
- **Step backward over functions**
- **Step backward into functions**
- **Advance backward out of current Function, to before the call**
- **Advance backward to selected line**
- **Run backward**
- **Advance forward to “live” session**
Enable ReplayEngine within a debugging session without restarting the program

- Recording starts when “Record” is selected
- Continue forward and then review and replay back to the start of the recording

**Most Requested Enhancement**

- Ease of use: you don’t have to remember to turn it on before you start up the program
- Skip initialization and other “known good” computations
- Manage runtime overhead so that you can use replay with late failures in longer-running jobs
- Precise control of where recording starts via breakpoints

**Second Most Requested Enhancement**

Memory-threshold Rolling History Capability
GPU Debugging

with

TotalView

CUDA
CUDA Port of TotalView

Full visibility of both Linux and GPU threads
- Device threads shown as part of the parent Unix process
- Handles 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 into or over inlined functions
- Functions show on stack trace

Reports memory access errors
- CUDA memcheck

Multi-Device Support
- Can be used with MPI
GPU Device Status Display

Provides the “high-level” view

• Values automatically update as you step through code
• Shows what hardware is in use
• Helps to map between logical and hardware coordinates
GPU Device Status Display

Provides detailed information for:

- Device and Type
- SMs
- Warps
- Lanes with PC

Information updates as you step

It also provides information for divergent GPU threads
Debugging CUDA

CUDA grid and block dimensions, lanes/warp, warps/SM, SMs, etc.

Parameter, register, local and shared variables
Debugging CUDA

GPU focus thread logical coordinates in the header...

```
// Matrix multiplication kernel called by MatrixMul()
__global__ void MatMulKernel(Matrix A, Matrix B, Mat
{
    // Block row and column
    int blockRow = blockIdx.y;
    int blockCol = blockIdx.x;
```
Debugging CUDA

... as well as in the Process Window
Debugging CUDA

PC arrow shows the Program Counter for the warp
Debugging CUDA

Dive on any variable name to open a variable window
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
Navigate through your CUDA code in the Process Window as you wish... Using either of two coordinate systems:
## Debugging CUDA - Navigation

### CUDA GPU threads have a negative TotalView thread ID

**Block** $(x,y,z)$

**Thread** $(x,y,z)$

User-controlled “spinboxes” allow selection and display of any part of your GPU execution.

GPU focus thread selector for changing the logical block and thread indexes of the CUDA thread.

- **Logical**: 2 or 3D Grid of Blocks, 3D Thread Within Grid
Debugging CUDA - Navigation

GPU focus selector for changing physical indexes of the CUDA thread.
- Physical: Device, SM, Warp, Lane

User-controlled “spinboxes” allow selection and display of any part of your GPU execution.
Executing GPU Code - Threads and Warps

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

- To advance the execution of more than one warp:
  - set a breakpoint and continue the process, or
  - select a line number in the source pane and select “Run To”.

- Warps advance synchronously
  - Warps share a PC

- Single stepping
  - Advances the warp containing the focus thread
  - Stepping over a `__syncthreads()` call advances all the relevant threads

- Continue and runto
  - Continues more than just the warp

- Halt
  - Stops all the host and device threads
CUDA Segmentation Faults

- TotalView displays segmentation faults as expected
  - Enable CUDA memory checking in New Program dialog window
OpenACC Support
What’s New

- Intel Phi Debugging
  - with
- TotalView
A Spectrum of Programming Use Models

Xeon-Centric

Xeon-native

Offload

General Purpose Serial and Parallel Codes

Scalar codes with highly parallel phases

Main() MPI_Foo()

Symmetric

Codes with balanced needs

Main() Foo()

Reverse

Parallel codes with scalar phases

Main() Foo()

Offload

Highly parallel codes

Main() MPI_Foo()

Offload<Xeon> Foo()
Spectrum of Execution Models

- CPU-Centric
  - Multi-core Hosted
  - Intel® Xeon Processor
  - General purpose serial and parallel computing

- Intel® Xeon Phi-Centric
  - Many-Core Hosted
  - Symmetric
  - Codes with highly-parallel phases
  - Codes with balanced needs
  - Highly-parallel codes

Productive Programming Models Across the Spectrum
Key Features:

- Full visibility of both host and coprocessor threads
- Full support of MPI programs
- Symmetric debugging of heterogeneous applications with offloaded code
- Remote debugging of Xeon Phi-native applications
- Asynchronous thread control on both Xeon and Xeon Phi
Debugging Applications with Offloaded Code

One debugging session for Phi-accelerated code
Debugging MPI Applications

• Attach to subset of processes on Phi
• Set breakpoints
• Debug MPI “as usual”
Remote Debugging of Applications on Phi

- Just run as `totalview –r hostN-micM <program>`
- Attach to running application
- See thread private data
- Investigate individual threads
- Analyze core crashes on Xeon Phi
- Kill stuck processes on Phi
Multi-host, Multi-card Phi-native MPI Debugging in TotalView 8.12

Each card has its own IP address and is accessible from front host node, running TotalView.
Multi-host, Multi-card MPI Debugging in TotalView 8.12

Single server launch (default)
- totalview -args mpiexec -np 240 -hosts host1-mic0,host1-mic1,host2-mic0,host2-mic1 ./tx_basic_mpi
- set env TVDSVRLAUNCHCOMMAND=<your ssh command to card> (ssh,micssh)
- Set TV::server_launch_string preference

MIC Native Launch
- totalview –mmic -args mpiexec -np 240 -hosts host1-mic0,host1-mic1,host2-mic0,host2-mic1 ./tx_basic_mpi
- Set: dset TV::mic_native_server_launch_string {
  ssh -n %R "/bin/rm -f /tmp/tvdsvrmain%K"; //1
  scp %B/tvdsvrmain%K %R:/tmp/tvdsvrmain_mic; //2
  ssh -n %R -n "/tmp/tvdsvrmain%K -callback %L -set_pw %P -verbosity %V %F" //3
}
  1. Removes your previous tvdsvrmain_mic
  2. Copies it from the installation directory to the /tmp/ directory on the coprocessor
  3. Starts the server on the Xeon Phi coprocessor.
Check Out the Web Site
roguewave.com/products/totalview/resources/

Videos

- Getting Started with TotalView
- Debugging MPI
- Introducing C++View
- Setting Breakpoint References for Threads
- Viewing Data Across Threads
- Threads Navigation
- Asynchronous Thread Control
- Debugging OpenMP
- C++ STL Type Transformations
- Memory Debugging with MemoryScape
- Memory Debugging with Red Zones
- Deterministic Replay with ReplayEngine
- 3 Part Series: Debugging on the Intel Xeon Phi Coprocessor with TotalView
Debugging Intel Xeon Phi Coprocessor with TotalView

Part 1 of Series: Debugging on the Intel Xeon Phi Coprocessor with TotalView

Debugging on the Xeon Phi with TotalView

Chris Gottbrath, Principal Product Manager

Part 2 of Series: Debugging Native Applications on the
Thank You