Addressing the Increasing Challenges of Debugging on Accelerated HPC Systems
Agenda

Overview - Rogue Wave & TotalView
GPU Debugging with TotalView
Nvidia CUDA
Intel Phi
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.

Latest addition to the Rogue Wave family: Rogue Wave Visualization for C++ (Formerly IBM’s ILOG Visualization for C++ products)
Representative Customers
Rogue Wave Product Offerings
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
  - Multi-threaded Debugging
  - Multi-process Debugging
- Heterogeneous Systems Support
- Integrated Memory Debugging
- Reverse Debugging for Linux
- Supports Multiple Usage Models
  - Powerful and Easy GUI – Highly Graphical
  - CLI for Scripting
  - Long Distance Remote Debugging
  - Unattended Batch Debugging
GPU Debugging

with

TotalView
CUDA Port of TotalView

Debugging on the GPU device (not in an emulator)

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
Debugging CUDA applications

• **Applications can take advantage of**
  - Kernels execute asynchronously
    • Overlap of communication and computation
    • The same kernel can operate on multiple streams
  - Multi-process applications
  - Utilization of multiple GPUs at the same time
  - Multi-level parallelism
    • MPI + OpenMP + CUDA

• **Interface rapidly developing with each release of the SDK**
  - Rogue Wave is working closely with NVIDIA to take advantage of capabilities as they are introduced
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.
Once a new kernel is loaded TotalView provides the option to stop and set breakpoints.

TotalView automatically configures the GUI for CUDA debugging.

Debugging CUDA code is done by using normal TotalView commands and procedures.
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

<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></td>
</tr>
<tr>
<td>Lanes</td>
<td></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></td>
</tr>
<tr>
<td>Lane 00/32</td>
<td></td>
</tr>
<tr>
<td>Lane 01/32</td>
<td></td>
</tr>
<tr>
<td>Lane 02/32</td>
<td></td>
</tr>
<tr>
<td>Lane 03/32</td>
<td></td>
</tr>
<tr>
<td>Lane 04/32</td>
<td></td>
</tr>
<tr>
<td>Lane 05/32</td>
<td></td>
</tr>
</tbody>
</table>
GPU Device Status Display

<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></td>
<td>Valid Warps</td>
</tr>
<tr>
<td></td>
<td>Warp 00/48</td>
</tr>
<tr>
<td></td>
<td>- Lane 00/32</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Lane 01/32</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Lane 02/32</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Lane 03/32</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Lane 04/32</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Lane 05/32</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Lane 06/32</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Lane 07/32</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Lane 08/32</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Lane 09/32</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Valid/Active/Divergent</td>
</tr>
<tr>
<td>- SM Type</td>
<td>sm_20</td>
</tr>
<tr>
<td>- SMc</td>
<td>14</td>
</tr>
<tr>
<td>- Warps</td>
<td>48</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>

It also provides information for divergent GPU threads.

Different PC for two groups of Lanes

State of Lanes inside the Warp
Debugging CUDA

Information on GPU execution, location and data is readily available. ... the same as it is for Linux processes and threads.
Debugging CUDA

CUDA grid and block dimensions, lanes/warp, warps/SM,

Parameter, register, local and shared variables
Debugging CUDA

GPU focus thread logical coordinates in the header...

```
cudaFree(d_C.elements);
}

// 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

```c
87 // Matrix multiplication kernel called by MatMulKernel
88 __global__ void MatMulKernel(Matrix A, Matrix B)
89 {
90   // Block row and column
91   int blockRow = blockIdx.y;
92   int blockCol = blockIdx.x;
93   // Each thread block computes one sub-matrix
94   Matrix Csub = GetSubMatrix(C, blockRow, blockCol);
95   // Each thread computes one element of Csub
96   // by accumulating results into Cvalue
97   float Cvalue = 0;
98   // Thread row and column within Csub
99   int row = threadIdx.y;
100  int col = threadIdx.x;
101  // Loop over all the sub-matrices of A and B
```

---

<table>
<thead>
<tr>
<th>Action Points</th>
<th>Processes</th>
<th>Threads</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1 (47350702036240)</td>
<td>T</td>
<td>in cudbgApiInit</td>
</tr>
<tr>
<td>1.2 (1095072064)</td>
<td>T</td>
<td>in __select_nocancel</td>
</tr>
<tr>
<td>1.-1 ((0, 0, 0)(1, 1, 0))</td>
<td>B1</td>
<td>in MatMulKernel</td>
</tr>
</tbody>
</table>
Debugging CUDA

PC arrow shows the Program Counter for the warp
Debugging CUDA

Dive on any variable name to open a variable window
Debugging CUDA

"@parameter" type qualifier indicates that variable "A" is in parameter storage

Address 0x10 is an offset within parameter storage

Pointer value 0x110000 is an offset within global storage

"elements" is a pointer to a float in global storage
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

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

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

• 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

• 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
TV 8.10 support for CUDA 4.1 specific features

- Works with the CUDA 4.1 SDK and Runtime
  - New Compiler Front End
  - Slight differences in the 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
OpenACC

- Pragma Based
  - Similar to OpenMP
  - Supports C and Fortran
- OpenACC Vendors
  - Cray
  - PGI
  - CAPS
- TotalView early access with Cray CCE 8
- Working with PGI and CAPS towards 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
- Parallel codes with scalar phases

MIC-Centric

- MIC-native
- Offload

- Codes with balanced needs
- Highly parallel codes

Main()

MPI_Foo()
Intel MIC Port of TotalView

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

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

- Start application on Phi card
- Attach to running application
- See thread private data
- Investigate individual threads
- Kill stuck processes on Phi
Debugging Applications with Offloaded Code

## Xeon side

### Function `samples0` in `samples0.c`

```c
30 // Sample 0B
31 // This sample demonstrates how to offload placed in front of
32 // an OpenMP construct enables OpenMP on the target
33 // Effectively, this is heterogeneous OpenMP
34 void sample0B()
35 {
36     float pi = 0.0F;
37     int count = 10000;
38     int i;
39     // offload target (no)
40     // offload one parallel for reduction(+pi)
41     for (i = 0; (i++) < count) {
42         float t = (float)((++i));
43         pi += t;
44     }
45     // conditional
46     if (fabsf(pi-3.14F) < 0.0F)
47     {
48         printf("sample0B: pi =%.1F", pi);
49     }
50 }
```

## Phi side

### Debugging session for Phi-accelerated code

![Debugging session for Phi-accelerated code](image)
TotalView provides a full spectrum of debugging solutions

Code debugging
- Highly scalable interactive GUI debugger
- Powerful features for debugging multi-threaded, multi-process, and MPI parallel programs
- Compatible with wide variety of compilers across several platforms and operating systems

Memory Debugging
- Parallel memory analysis and error detection
  - Intuitive for both intensive and infrequent users
  - Easily integrated into the validation process

Reverse Debugging
- Parallel record and deterministic replay within TotalView
  - Users can run their program “backwards” to find bugs
  - Allows straightforward resolution of otherwise stochastic bugs

GPU CUDA Debugging
- Full Hybrid Architecture Support
- Asynchronous Warp Control
- Multi-Device and MPI Support

Intel Phi Debugging
- Multi-Model Support
- Early Experience Program Available
Summary, Conclusions, Next Steps and Questions

• Rogue Wave Software provides a suite of tools and libraries that can accelerate your software engineering efforts
• TotalView provides comprehensive hybrid debugging environment
• Initial support for Intel PHI has begun.
  – Including support for multiple workflows
  – We are interested in learning the workflows that you are going to apply
  – If you are interested in our early experience program, let us know.
• Questions?
Thanks!

Thank You

Download a TotalView evaluation at: www.roguewave.com/products

Academic License
tvexpress@roguewave.com