Porting Legacy Code of a Large, Complex Brain Simulation to a GPU-Enabled Cluster

Chris Gottbrath, Principal Product Manager
March 21st, 2013
Agenda

- Who are Rogue Wave Software
- HRL CNES Neural Network Simulation
- Other Case Studies
- Overview of TotalView for CUDA
- Latest Updates and Futures
- Conclusion
Rogue Wave Software

- **History**
  - Founded: 1989
  - Acquired by Battery Ventures: 2007
  - Acquired:
    - Visual Numerics: 2009
    - TotalView Technologies: 2010
    - Acumem: 2010
    - ILOG Visualization Tech 2012

- **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**
Leveraging Leading Edge Analytics

Challenges

• Understanding the data is difficult
• Knowing when and how to apply mathematical and statistical methods requires an advanced skill set
  – Which method for particular data sets and situations
  – Must go through several rounds of prototyping and testing
• Developing the algorithms is even more difficult
  – Making sure they are correct
  – Making sure they are correct when they execute in parallel environments

Rogue Wave Solutions

• Build or extend the application using IMSL numerical libraries
  – Cross-platform, multi-platform
  – Already optimized for parallel deployment
• IMSL C Numerical Library
• IMSL C# Numerical Library for .NET Applications
• JMSL™ Numerical Library for Java™ Applications
• IMSL Fortran Numerical Library
IMSL Numerical Library Methods

Mathematics
- Basic Types
- Linear Algebra
- Eigensystems
- Interpolation & Approximation
- Quadrature
- Differential Equations
- Nonlinear Equations
- Optimization
- Special Functions
- Finance & Bond Calculations
- Genetic Algorithm

Statistics
- Basic Statistics
- Time Series & Forecasting
- Nonparametric Tests
- Correlation & Covariance
- Data Mining
- Regression
- Analysis of Variance
- Transforms
- Goodness of Fit
- Distribution Functions
- Random Number Generation
- Neural Networks
• **IMSL C and Fortran libraries both support CUDA**
  – BLAS Linear Algebra routines
  – Uses CUDA where appropriate
    • Just a link time change .. No code changes whatsoever!
  – Contact Rogue Wave for more details.
IMSL Fortran GPU Benchmarks

The chart shows the speedup of SGEMM using NVIDIA CUBLAS compared to IMML BLAS and High Perf BLAS. The x-axis represents the matrix size (500, 1000, 2000, 4000, 8000), and the y-axis represents the speedup. The graph indicates that as the matrix size increases, the speedup also increases for all tested configurations.
What is TotalView?

Application Analysis and Debugging Tool: Code Confidently

- Debug and Analyze C/C++ and Fortran on Linux, Unix or Mac OS X
- Debugs host and accelerator together
- 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 and OpenACC
- Includes a Remote Display Client freeing 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
What is ThreadSpotter?

- **Runtime Cache Performance Optimization Tool: Tune into the Multi-Core Era**
  - Realize More of the Performance Offered by Multi/Many-Core Chips
  - Quickly Detects and Prioritizes Issues -- and then Provides Usable Advice!

- **Features**
  - Supports Windows and Linux x86/x86-64
  - Any compiled code
  - Runtime Analysis
    - Low overhead
  - Cache Modeling
    - Prioritizes Issues
    - Identifies Problem Lines of Code
  - Provides Advice
    - Explanations
    - Examples
    - Detailed statistics (if desired)
Case Study: HRL

- **Who:** Center for Neural and Emergent Systems (CNES) at HRL
- **Project:** Scaling up neural network technology.
- **Technology Chosen:** Using a CUDA accelerated cluster to model the brain.
- **Challenge:** New technologies, ambitious goals, and short timelines put everyone under pressure. They were unable to resolve bugs fast enough to meet timelines.
- **Solution:** TotalView which radically shortened bug diagnosis and allowed them to meet their project goals.
The Project

• **Cortical Simulation Project**
  – Biomorphic Networks
  – Modeling the brain’s structure
  – Large numbers of neurons
  – Massive numbers of connections
  – Lots of short connections, fewer longer range connections

• **Lots of parallelism**

• **200 Node cluster**
  – NVIDIA GPU acceleration
“One of the tasks of the CNES team at HRL is simulating large scale networks in real time. The only possible way to simulate such massive networks without moving to a full-scale supercomputer is with a large GPU cluster. “
- Kirill Minkovitch, HRL
Business Challenges

• Ambitious Goals
  – Really “scaling out” the problem in terms of the size of the simulated network
  – Simulation has to generate results quickly
  – Prohibitively expensive with traditional processors

• Cutting edge technology
  – First major foray into CUDA based GPU programming

• Demanding project schedule
  – Productivity is key
  – Bug related delays have business impact
Technical Challenges

• Architecture
  – Mapping the problem effectively to the parallelism in the hardware
  – Balancing computation and communication

• Tools
  – Tried with open source tools and failed
  – Needed a single polished approach for MPI and CUDA

• Project schedules and goals at risk
Finding the right balance

- Communication vs Computation

- Overlapping communication was critical to achieving results
  - Main computation occurs on the GPU device
  - The host processor is used to facilitate long range communications
Expressing multiple levels of parallelism

- The system being modeled is fundamentally parallel
- Matching that parallelism to the hardware hierarchy
- Minimizing communication overhead
- Frequent short range connections handled on the device
- Less frequent longer range communication over MPI
What did they do before TotalView?

- Used multiple flavors of GDB, for different tasks
- Struggled with configuration, managing multiple debuggers
- Wished they could easily use printf in kernels
  - This issue has been resolved
Not adequate at all

- “For a short time we tried using existing tools, but setting up the environment, even before debugging began, became more and more daunting. So we looked at the time we were devoting to debugging, and realized if we were going to meet our release deadline, something had to change.”
  - Kirill Minkovitch, HRL
Unified Accelerated Cluster Debugging Environment

• Need to debug CUDA in context of MPI
  – Attach to all the MPI processes that contribute to a specific job
  – Control MPI processes and compare their behavior

• Can’t always isolate CUDA issues to a specific instance
  – Failures may happen anywhere within the job

• Using non-MPI aware CUDA tools required heroic set up in the cluster environment
Radical step forward with TotalView

“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.”

- Kirill Minkovitch, HRL
Aside: Vote for your favorite bug!

- **Nominations include:**
  - Failure to Launch
  - Failure to check return codes
  - Error Code 30
  - Abuse of Malloc
  - Pointer Arithmetic
  - Seg faults
  - Memory alignment problems
  - Typos/People make mistakes

- Favorite can be most frequently encountered, hardest, most memorable

- We’re doing drawings daily for 15 dollars and once at the end of the show for 50 dollars

- A business card or entry card

- Make sure there is an email address so we can contact you
Impact of using the right tools

• Not just:
  – Easier to fix bugs
  – More productive

• Accomplished more
  – Hit critical goals
  – Added more features
  – Added more ambitious features like host-side multi-threading
Success

• “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.”
Not a replacement for good coding style..

- “Although using TotalView is no replacement for good coding style and error checking, with it we have been able to quickly add complex features like multithreading; that would have been difficult without such a tool.”

  - Does avoiding certain tools make you a better developer?
  - It isn’t about “debugger or good coding style”
  - They were successful with the right tools and good coding style
Time saved debugging can be spent optimizing

- Optimization is a time consuming activity

- In an actively developed code base optimization comes .. Last
  - More often than it should

- So, quite understandably it gets squeezed.

- With this project scalability was critical, the goals could only be achieved by putting the right kind of priority on optimization
### Other Case Studies

<table>
<thead>
<tr>
<th>Environment</th>
<th>C, C++ &amp; Fortran on 12,800 processor Blue Gene/L, P, Q</th>
</tr>
</thead>
<tbody>
<tr>
<td>Issues</td>
<td>Complex code, debug across thousands of processors, gigabytes of memory</td>
</tr>
<tr>
<td>Solution</td>
<td>TotalView</td>
</tr>
<tr>
<td>Result</td>
<td>Reduced development time by 20%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Environment</th>
<th>C, C++ on x86 based Linux Servers and Linux Clusters using MPI</th>
</tr>
</thead>
<tbody>
<tr>
<td>Issues</td>
<td>Mysterious issues evaded testing &amp; traditional debugging</td>
</tr>
<tr>
<td>Solution</td>
<td>MemoryScape</td>
</tr>
<tr>
<td>Result</td>
<td>Faster QA, proactive bug discovery and resolution leading to Improved product quality</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Environment</th>
<th>C, C++ and Fortran ported from Legacy Unix to an Intel Linux environment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Issues</td>
<td>Porting a mission critical simulation from Fortran on Unix to C++ on Linux</td>
</tr>
<tr>
<td>Solution</td>
<td>TotalView</td>
</tr>
<tr>
<td>Result</td>
<td>Development time reduced by 60-70%, software costs reduced by 40%</td>
</tr>
</tbody>
</table>

“Since OpenGeoSolutions began using TotalView, our development time has been reduced by months. Without TotalView’s capabilities, our programmers would be taken back to the days of print statements and generating reams of paper output in order to understand, locate and repair broken code. The time spent on this style of error assessment can be astronomical.”

Paul Garossino, Geoscientist at OpenGeoSolutions
Widely varied applications running on IBM BlueGene/L, P, Q
- LLNL’s BlueGene/Q has 98K nodes, 1.6M cores.
- Complex applications use C, C++, Fortran and MPI and process terabytes of data

Solution: TotalView debugger
- Parallel Debugger able to handle > 1M threads.
- Asynchronous thread control, conditional breakpoints and memory debugging with MemoryScape

Benefits:
- Simplified the task of scaling up applications to take advantage of the BlueGene/L architecture
- Reduced development time by more than 20%
**Challenge:** Porting to New HPC Platforms & Languages

**Solution:**

- High Fidelity Simulations of Weapon Systems
  - Running on a Variety of Older HPC Platforms
  - Ported to Intel C++ Running on Multi-core Linux Platforms
- Solution: TotalView Debugger
  - Debugs Multi-threaded Apps
  - Wide Platform and Language Support
  - Streamlined User Interface
- Benefits:
  - Cut Development Time
  - Lowered Costs
  - Improved Software Reliability

"The ability to port code, and to watch and test on two different platforms side-by-side in real-time using the TotalView Debugger, dramatically cut our development time. The TotalView Debugger didn’t only meet all of our needs, it was also a very robust tool that was easy to use and gave us great confidence."

Jim Knoblach, Manager, Modeling and Simulation Department CAS, Inc.
Challenge: Hard to Analyze Bugs Slowing Development

Solution: MemoryScape – Memory Debugging and Analysis

- Develops Abaqus Finite Element Analysis Suite & other tools
  - C, C++ and Fortran
  - Multi-threaded and Multi-Process
- Struggling with intermittent errors
  - Due to buffer overruns manifesting far from cause
- Solution: MemoryScape
  - Gives a clear view of the source of the problem
  - Faster and easy to work with
- Benefits:
  - Reduced effort on bug fixing & Better product quality

“MemoryScape enabled us to identify memory issues, and by using its scripting interface, we were able to automate the evaluation process. Now, the system automatically uncovers any hidden latent errors in our code with every build, allowing our developers to proactively fix potential errors prior to release.”

Nick Monyatovsky, Software Engineer at SIMULIA
What is TotalView?

Application Analysis and Debugging Tool: Code Confidently

- Debug and Analyze C/C++ and Fortran on Linux, Unix or Mac OS X
- Debugs host and accelerator together
- 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 and OpenACC
- Includes a Remote Display Client freeing 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
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
  - MPI, pthreads, OMP, and remote apps
- Low runtime overhead
- Easy to use
  - Works with vendor libraries
  - No recompilation or instrumentation
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
Cambridge Study

• Survey conducted by the Judge Business School at Cambridge University concluded that Reverse Debuggers allow users, on average, to spend 13% less of their programming time debugging.
  – Programming was 50% of total work week on average
  – Debugging was 50% of programming time without reverse debugging
  – Debugging was 37% of programming time with reverse debugging
  – That frees up 130 hours (>3 work weeks, 6.5% total time) per developer per year for design and new feature development

• The survey looked at total value (salaries & overhead) of debugging as a task and they determined that this savings could, across the whole world economy, be work $41 billion in increased productivity.
  – The productivity improvement should be worth $2,500 per developer per year (salary only) or $5,000 per year with overhead.

TotalView for CUDA

- Full visibility of both Linux threads and GPU device threads
- Fully represent the hierarchical memory
- Supports Unified Virtual Addressing and GPUDirect
- Thread and Block Coordinates
- Device thread control
- Handles CUDA function inlining and CUDA stacks
- Support for C++ and inline PTX
- Reports memory access errors
- Handles CUDA exceptions
- Multi-Device Support
- Can be used with MPI
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 thread (1:1)

- 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

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.

CUDA host threads have a positive TotalView thread ID.

CUDA GPU threads have a negative TotalView thread ID.

Thread (x,y,z)

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

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

GPU focus thread logical coordinates

Parameter, register, local and shared variables

Dive on a variable name to open a variable window

© 2013 Rogue Wave Software | All Rights Reserved
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

### Example of Divergent GPU threads

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Device 0/3</td>
<td>gF100</td>
</tr>
<tr>
<td>SM 2/1</td>
<td></td>
</tr>
<tr>
<td>Valid Warps</td>
<td>0000000000000001</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>0000000019a39d85</td>
</tr>
<tr>
<td>Lane 01/32</td>
<td>Thread (1,0,0)</td>
</tr>
<tr>
<td>LPC</td>
<td>0000000019a39d85</td>
</tr>
<tr>
<td>Lane 02/32</td>
<td>Thread (2,0,0)</td>
</tr>
<tr>
<td>LPC</td>
<td>0000000019a39f0</td>
</tr>
<tr>
<td>Lane 03/32</td>
<td>Thread (3,0,0)</td>
</tr>
<tr>
<td>LPC</td>
<td>0000000019a39f0</td>
</tr>
<tr>
<td>Lane 04/32</td>
<td>Thread (4,0,0)</td>
</tr>
<tr>
<td>LPC</td>
<td>0000000019a39f0</td>
</tr>
<tr>
<td>Lane 05/32</td>
<td>Thread (5,0,0)</td>
</tr>
<tr>
<td>LPC</td>
<td>0000000019a39f0</td>
</tr>
<tr>
<td>Lane 06/32</td>
<td>Thread (6,0,0)</td>
</tr>
<tr>
<td>LPC</td>
<td>0000000019a39f0</td>
</tr>
<tr>
<td>Lane 07/32</td>
<td>Thread (7,0,0)</td>
</tr>
<tr>
<td>LPC</td>
<td>0000000019a39f0</td>
</tr>
<tr>
<td>Lane 08/32</td>
<td>Thread (8,0,0)</td>
</tr>
<tr>
<td>LPC</td>
<td>0000000019a39f0</td>
</tr>
<tr>
<td>Lane 09/32</td>
<td>Thread (9,0,0)</td>
</tr>
<tr>
<td>Valid/Active/Divergent</td>
<td>00000003Ff, 000003Ff, 00000003</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>
<tr>
<td>Device 1/3</td>
<td>gt200</td>
</tr>
<tr>
<td>- Device Type</td>
<td></td>
</tr>
<tr>
<td>- Lanes</td>
<td>32</td>
</tr>
<tr>
<td>- SM Type</td>
<td>sm_13</td>
</tr>
</tbody>
</table>
GPU Memory Hierarchy

- Hierarchical memory
  - Local (thread)
    - Local
    - Register
  - Shared (block)
  - Global (GPU)
    - Global
    - Constant
    - Texture
  - 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.
CUDA Variables

- Storage qualifiers appear in the data type
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 Segmentation Faults

- TotalView displays segmentation faults as expected
- Must enable CUDA memory checking
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;`
TotalView for OpenACC

- Step host & device
- View variables
- Set breakpoints

Compatibility with Cray CCE OpenACC
TotalView General Updates and Futures

• Major Projects & Initiatives
  – Scalability
  – User interface

• Current release is TotalView 8.11
• TotalView 8.12 beta will start in just a few weeks
  – Sign up for the beta!
Scalability Project

- Goal is Petascale Parallel Debugger Scalability
- Collaboration with LLNL & Tri-lab partners
- MRNet – infrastructure
  - Tree overlay network (multicast and reduction)
- Multi-platform:
  - BlueGene/Q
  - Cray XT/XE/XK/XC
  - Linux Cluster with Infiniband
- Application Driven Tuning
- GUI improvements for representing data and working with groups in a scalable way
TotalView debugs 786,432 cores.
Climb with Rogue Wave towards exacale.
Some more details on the 786,432 core test

- The test was performed on 48 racks of Sequoia
- The test code
  - Implements a Jacobi Linear Equation Solver
  - The test code is a hybrid MPI + OpenMP code
  - 16 threads per process, one process per node
- The test operations
  - Start up
  - Setting breakpoints / removing breakpoints
  - Single stepping all threads
- Tests performed at a variety of scales to understand scalability
Second test - Oversubscription

- **Same framework**
  - same code
  - same machine

- **Oversubscription**
  - Scheduled more than one thread per physical core
  - This is a reasonable use case since the BG/Q supports 4 logical threads per core

- **TotalView Debugged 1,048,576 threads**
TotalView 8.11

• This was released at SC12

• Features
  – Blue Gene/Q Support
    • Includes Async Thread Control, Memory Debugging, Dynamic Library Support, etc...
  – Early Access to Intel Xeon Phi (MIC) Debugging
    • Support for both Native and Offload mode
    • Multi-device support
  – CUDA 4.2
  – Cray CCE 8 OpenACC

• For select customers
  – Limited Early Access for Scalable MRNet Infrastructure
Features Planned for ISC 2013 release

• Will be TotalView 8.13
• Headline Feature
  – Productized Xeon Phi (MIC architecture) support
• Other Features
  – AVX Support
  – New Program Dialogue
  – Cray ATP
  – Apple OS X Mountain Lion
  – Improved support for Cray XC series
  – Improved performance with breakpoints and C++ programs
Summary

- HRL’s experience suggests
  - CUDA accelerated clusters can achieve great results
  - The right tools, including TotalView, were critical for achieving their ambitious project goals and demanding timelines
  - Productivity gains in debugging pay dividends in
    - Hitting schedules
    - More features
    - Freedom to focus on optimization
  - Best practice couples tools with good coding practices
  - Other sites have seen similar value in TotalView
Thanks!

- Vote for your favorite bug here or at our booth!
- Please come and talk to us at our booth over lunch
  See a demo of TotalView for CUDA
- For more information about HRL you can find the case study online
  http://www.roguewave.com/resources/case-studies.aspx
- Learn more about TotalView and sign up for an evaluation at
- Sign up for our beta program!
- Contact me at chris.gottbrath@roguewave.com
  Or our sales team at sales@roguewave.com