GPU Technology Conference 2015

Three Ways to Debug Parallel CUDA Applications: Interactive, Batch, and Corefile

RogueWave SOFTWARE

Accelerating Great Code

Three Ways to Debug Parallel CUDA Applications: Interactive, Batch, and Corefile

What do I see

- Expansion of the target domains
  - Not just numerical simulations
- Increased maturity of the software stack
- A lot more users
- A variety of programming paradigms
  - CUDA
  - OpenACC
  - OpenMP 4
  - OpenCL
- OpenPOWER and ARM64 as host processor architectures

Agenda

- Who are Rogue Wave software
- Challenges
- Techniques and strategies
- Tools

Rogue Wave Software

What we do

Rogue Wave helps organizations **simplify** complex software development, **improve** code quality, and **shorten** cycle times

Capabilities

APPLICATION SECURITY

Klocwork, OpenLogic, TotalView, IMSL, SourcePro

CODE REFACTORING

Klocwork

CODE REVIEW

Klocwork, OpenLogic

DEBUGGING COMPLEX CODE

Klocwork, TotalView

REUSABLE MATH ALGORITHMS

IMSL, SourcePro

OPEN SOURCE AUDITING

OpenLogic

OPEN SOURCE MANAGEMENT

OpenLogic

OPEN SOURCE SUPPORT

OpenLogic

CERTIFIED OPEN SOURCE

OpenLogic

STATIC CODE ANALYSIS

Klocwork

DEVELOPING USER INTERFACES

Visualization, Stingray, PV-WAVE

CODE MIGRATION

SourcePro, IMSL, HydraExpress

CODE BUILDING BLOCKS

SourcePro, IMSL, Stingray, Visualization

Global, diversified customer base

Used by 3,000 customers in over 57 countries across diverse industries to develop mission-critical applications and software

Financial Services

Bank of America, citibank, Credit Suisse, Liberty Mutual, Goldman Sachs, QUESTRADE, Bloomberg, PRUDENTIAL, Morgan Stanley, UBS, BARCLAYS, MIZUHO, E*TRADE, Deutsche Bank, JPMorganChase

Telecom

at&t, HUAWEI, BT, verizon, Alcatel·Lucent, GenCorp, Sprint, MOTOROLA, Polycom, QUALCOMM, ERICSSON, L3, BlackBerry

Gov't / Defense

BOEING, NASA, NORTHROP GRUMMAN, BAE SYSTEMS, Raytheon, Lawrence Livermore National Laboratory, LOCKHEED MARTIN, Sandia National Laboratories

Technology

CISCO, ca, ORACLE, IBM, bmcsoftware, DELL, EMC², RSA SECURITY, hp, SUNGARD, fiserv., SAP

Other Verticals

PIXAR, ExxonMobil, McKESSON, Schlumberger, Disney, DELPHI, INDUSTRIAL LIGHT & MAGIC, Shell, Cerner, OLYMPUS, SIEMENS, MARATHON, Saudi Aramco, DreamWorks, Schneider Electric, TOTAL, JOHN DEERE

Debugging Challenges

Target Domains

What are people doing with GPUs

- Physical Modeling
  - Chemical
  - Weather
  - Astrophysical
- Biological Modeling
  - Neural Networks
  - Proteins
- Financial Modeling
  - Derivatives
  - Porfolio Risk Analysis
- Deep Learning
  - Automotive
  - Speech
  - Image
  - Text
- Signal Processing
- Oil and Gas
  - Seismic
  - Reservoir Modeling

Common Challenges

What makes GPU debugging hard?

- Hardware differences
  - Vectors
  - Warps
  - Numerical behavior
- Data decomposition
  - Explicit mapping of data to arrays and grids
  - Data transfers to and from the device
  - Ghost regions
- Buffer overruns
  - MPI communication buffers
  - Data transfers to and from the device
- Pointers
  - Different memory spaces
- Concurrency and parallelism
  - Racy bugs
  - Needle in the haystack
  - Passing the bad apple

Challenges continued

Directive based languages

- What about directive based approaches?
  - OpenACC
  - OpenMP 4 target device and SIMD directives
  - OpenCL
- Compiler manages data migration with hints from the user
- Some challenges remain
  - Getting variable annotation correct
    - And understanding the performance impact of choices
  - Pointers
  - MPI
  - Needle in the haystack
  - Error propagation

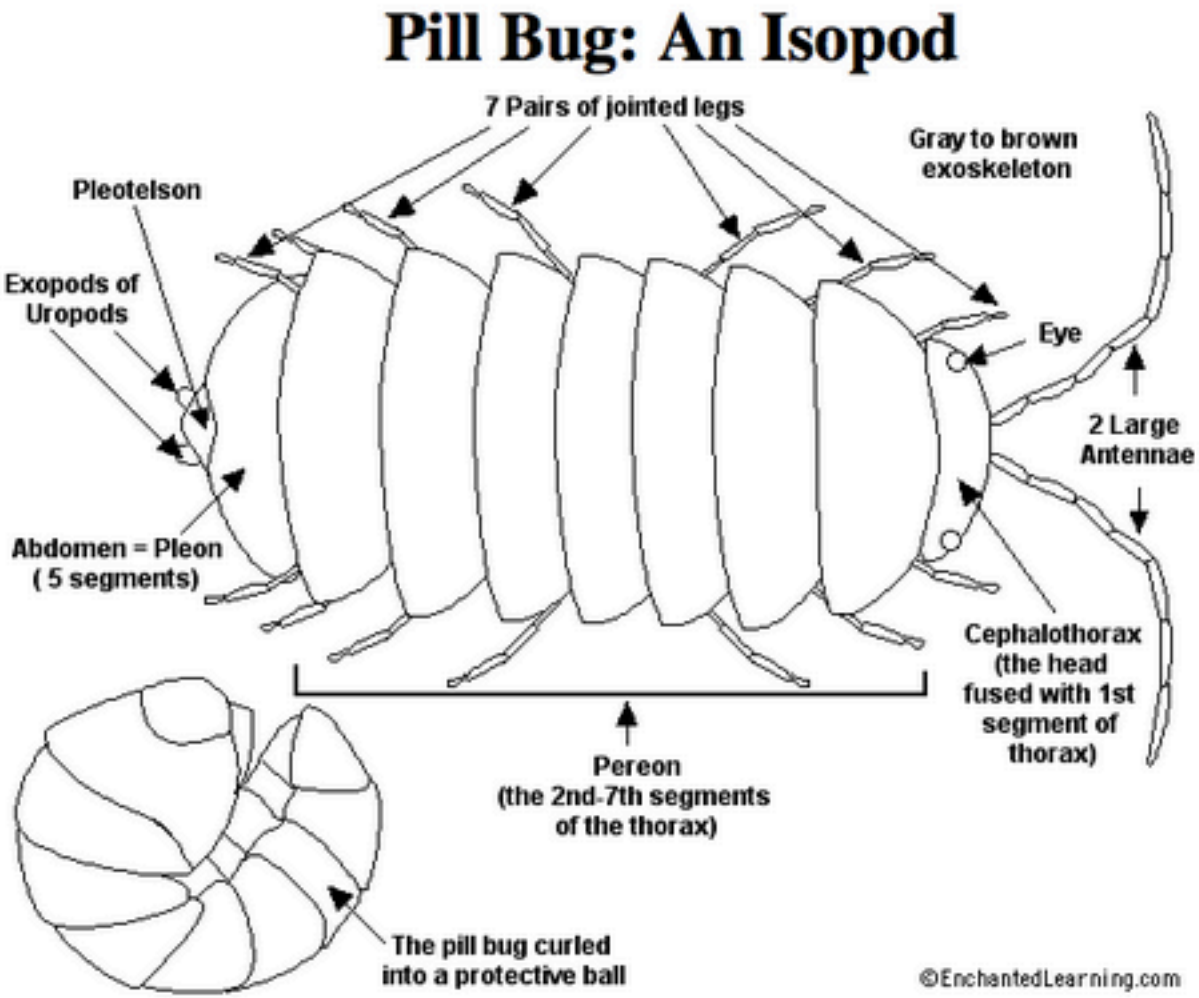
Strategies and Techniques

Debugging Strategies Generally

Thoughts about debugging strategies

- Design for clarity
  - You *will* need to debug
  - Modularity, testing
  - Check return codes
  - Documentation, etc..
- Capture what you need to reproduce the bug
  - Source code version
  - Input data set
  - Scale dependent
  - OS version
  - Libraries
  - Tool Chain
  - Device
  - Deterministic or racy
- Localize the symptoms
  - Where does it first go “off the rails”
  - Visible effect
- Hypothesis testing cycle
  - Develop a hypothesis
  - Find a way to test the hypothesis
  - Repeat till you find the defect
- Fix
- Validate the fix
  - If not then go back to hypothesis testing

Anatomy of a bug



Anatomy of a bug

Terminology: Bugs, root causes, defects, etc..

- **Sighting**: The situation in which the bug came to light: crash, hang, test failure, bug report
- **Defect**: The cause of a bug is one or more defect or mistake in the code
- **Effect**: What effect the defect has which might include things like
  - Incorrect state
  - Error propagation
  - Visible incorrect behavior or output
- **Trigger:** an input data or conditions in which the defect will cause the effect
- **Reproducer:** an articulation of the sighting that includes a clear trigger condition
- **Resolution:** the change to the code that eliminates the defect (ideally) or prevents the trigger condition (sometimes adequate)
- **Verification:** something that is done to ensure that the resolution is correct
  - Best practice includes things like a clean run of all regression tests along with a new a persistent regression test to watch for the bug and bugs like it.
- A **bug**
  - Encapsulates the sighting, defect, trigger, fallout, resolution and verification
- **Root cause** is the fundamental defect itself or the reason the defect made it into the code
  - Most bugs are “just mistakes” and the root cause = the defect itself
  - Sometimes it is valuable to try to push the root cause back to the architecture, specification, process or team dynamic that lead to the defect being written in the first place

Debugging multi-core applications

Observations about multi-threaded debugging

- Multithreaded applications and shared memory programming
  - Data can be shared (higher memory efficiency)
- Shared memory programming
  - Complexity: Only some memory is shared
- Multi-threaded programming
  - All threads share the same heap and global
  - Separate stacks (but mutually readable)
- Compared with multi-process parallelism
  - Similar challenges and techniques
  - Communication not as localized
  - Memory management is shared

Debugging multi-core applications

Techniques for multi-threaded debugging (1/2)

- Print debugging can work for some bugs but can be very confusing for others
  - Changes timing
- Look carefully at the thread capabilities of your debugger
- A good multi-thread debugger will give you
  - An asynchronous interface
    - Doesn't assume a simple running/stopped state
  - Easy access to all threads
  - Complete control over threads
    - Thread aware breakpoints
    - Ways to synchronize threads
    - Ways to hold threads
    - Thread groups
  - Display of thread states
  - Display of thread-private data
  - Display of data across threads

Debugging multi-core applications

Techniques for multi-threaded debugging (2/2)

- Try to reproduce problems without threads
- Vary the number of threads
- Try different interleaving patters
- Look at thread synchronization point (mutexes, semaphores, barriers)
- Use watchpoints (aka data breakpoints)
- Make sure resources are cleaned up before thread termination
- Capture the exact thread execution pattern with Record and Replay techniques

GPU Accelerated Codes

Techniques for GPU accelerated workstation / servers

- In addition to everything mentioned under thread debugging
- Memory is not protected on the GPU
  - Use CUDA memcheck capability of the runtime
- Compare data on both the host side and the device side
- Make sure that you are checking the return values on CUDAmalloc()
- Vary the domain size
  - Trigger bugs that are lurking in domain decomposition
  - Ideally during testing
- Variable initialization
- Review pointer memory types
- Race Conditions between device threads in the GPU
  - CUDA memcheck has a thread checking capability
- Race Conditions between the device threads and the host processor
  - Memcheck can't check these
  - Add synchronization
  - Use the debugger to change the ordering

Debugging distributed applications

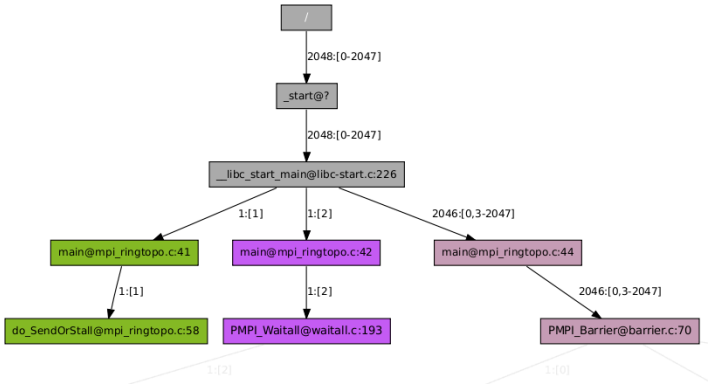
Techniques for debugging distributed apps (1/3)

- Print debugging doesn't scale
- You can debug 1 of N processes
  - Needle in the haystack problem
  - Passing the bad apple problem
- You can run N separate debuggers on N processes
  - Repetition
  - Coordination
  - Confusion
- You can use a parallel debugger
  - One debugger for N processes!

Debugging distributed applications

Techniques for debugging distributed apps (2/3)

- Parallel Debuggers and lightweight debug tools will
  - If any process fails you can focus on it
  - Allow you to synchronize your processes
  - Allow you to focus on any process
  - Allow you to compare processes
  - Give you ways to find outliers
  - Give you ways to group processes and work with those groups



LLNL STAT Tool

Debugging distributed applications

Techniques for debugging distributed apps (3/3)

- Re-run at different scales
  - Debug at lowest scale that exhibits defect
- Compare program flow in working and non-working cases
- Follow bad data back from the symptom to the cause
- Look closely at communication points and data decomposition
- Racy bugs
  - Try out different relative orders of execution
- Deadlocks & Live Locks
  - Examine all assumptions at sync points
  - Examine flow control across procedures around sync points
- As always when doing science: take careful notes

GPU Accelerated codes in Clusters

Techniques for GPU accelerated clusters

- Techniques from above ..

- GPUDirect
  - Make sure all memory used in inter device and internode communication buffers are pinned
  - Unified memory can't be used with GPUDirect
- Note that CUDA MPS device sharing can't be used with debugging.

Log file debugging

Locate and capture the context of the bug

- Recompile with print statements for a log file
- Compile in and toggle on/off with a runtime flag
- Trace with an external tool
  - System call tracing
  - Debugger assisted tracing (avoid the recompile step!)
- Tension & Trade-off
  - Capture enough context to understand what is happening
  - Manage the large volume of output that may be required
- Tips & Techniques
  - Binary search to find the site of the error
  - Consider file system / file size
  - Flush the pipe, otherwise file writing is asynchronous
  - The presence of a call sometimes changes the behavior (compiler bugs, optimization, race conditions)
  - Print debugging can be hairy with multi-thread or multi-process
    - Externally driven tracing tools may be preferable to ensure logging happens

Dynamic memory analysis

Pinpoint leaks and analyze memory use

- Dynamic memory tools help catch hard to identify bugs
  - memory leaks can lurk in a code base
  - bounds violations can corrupt data
    - can be an open door for malicious agents
  - dangling pointers lead to racy, hard to reproduce symptoms
- Dynamic memory tools can also be used to inspect what is happening in the heap memory
  - Normally quite hard to visualize and understand
  - Critical for optimizing for low memory environments
- Tips & techniques
  - Maintain a policy of eliminating 100% of leaks
  - Use with a testing system to make sure you exercise different kinds of input and different code paths
  - Compare heap behavior over time to make sure OS and library changes don't introduce problems

Reverse debugging

Get “racy” bugs “on tape”

- Record and deterministically replay execution trajectory through the code
  - Record non deterministic inputs
  - Replay those as needed to access any point in the execution
- Note: NOT currently implemented on GPU hardware
  - But you can use reverse debugging on the host side of a GPU accelerated code to debug host side bugs
- If you can get a racy bug to reproduce you can examine it at leisure
  - Give yourself the full benefit of hindsight
  - What steps led to it happening?
  - Where did the program go wrong?
- Tips & Techniques
  - Use watchpoints (data breakpoints) to find the source of corrupt data
  - Wait till you are close to the bug before activating the recording to avoid paying overhead for the entire runtime
  - Capture recordings and save them to a file as part of bug reports
  - Review recordings of defects in unfamiliar parts of the code with subject matter experts

Remote Debugging / Cross Debugging

Reserve Device Resources for your App

- Remote Debugging
  - debugger core runs on your workstation (host) system
  - lightweight agent process runs on the device (target) system
- The agent process is very lightweight
- The debugger core holds all the complex analysis data structures
- Tips & Techniques
  - Start with a debug target on the host machine
    - Copy and strip the version that goes on the device
  - You can start the server and then choose the target process
  - Sources may need to be accessible on the host
  - Use a tool that does the right thing with host/target library mismatch
  - Be aware of security

Core file debugging

A corefile is a good place to start

- The corefile isn't always sufficient
  - It can be trashed
  - It represents the consequence of the defect, but not the cause
- Examine the site of the crash
- Look for 'suspicious' variables
- Tips & Techniques
  - Compile with debug information
  - You can sync up a pre-stripped executable with a corefile generated by its stripped counterpart
  - Check the more than one stack frame
- Enabling corefiles
  - ulimit –c <size> (limit coredumpsize <size> on csh shells)
  - CUDA_ENABLE_COREDUMP_ON_EXCEPTION = 1
    - The file name is : core.cuda.HOSTNAME.PID
  - Override with CUDA_COREDUMP_FILE env variable

Batch mode debugging

Techniques for use in batch submission environments

- Examples
  - Batch submissions in HPC
  - Testing
    - Regression, Unit, Integration
    - Continuous Integration
  - Computationally intensive components in larger distributed systems
    - Cloud computing and scalable infrastructures
  - Embarrassingly parallel jobs
- Techniques
  - Print statements
  - Error logging
  - Scripted Debugging

Tools

Tools for debugging GP-GPU

What are some of your options

- Print Statements
- CUDA Memcheck
- CUDA GDB
- Allinea's DDT
- Eclipse Nsight
- TotalView

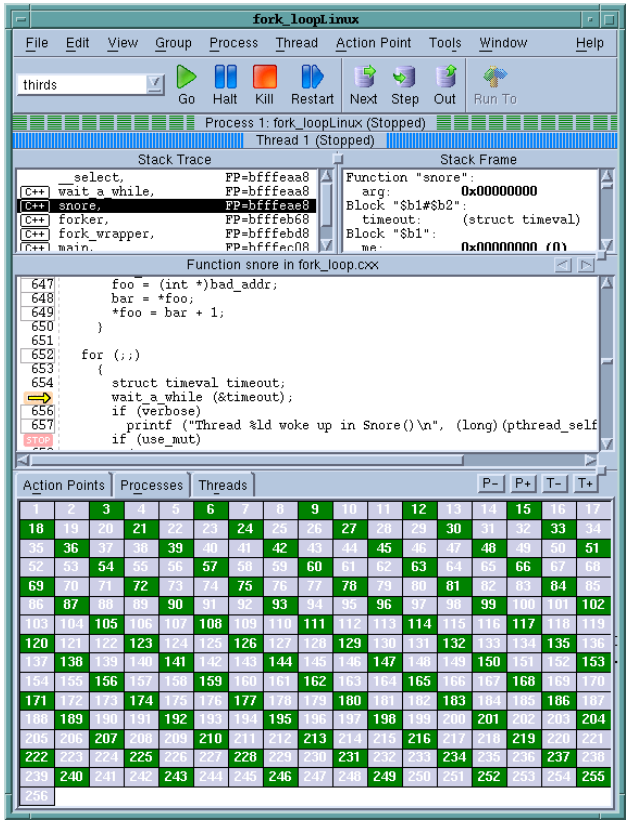
What is TotalView®?

Application Analysis and Debugging Tool: Code Confidently

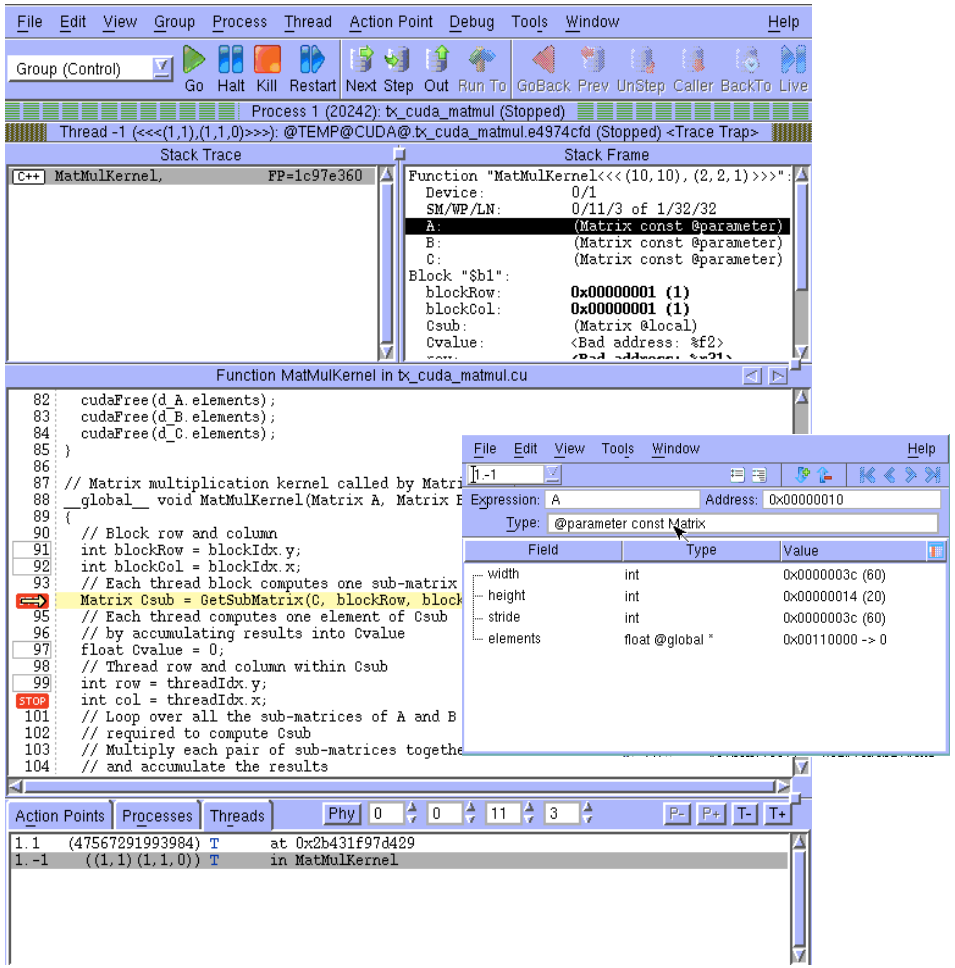
- Debug and Analyse C/C++ and Fortran on Linux™, Unix or Mac OS X
- Laptops to supercomputers
- Makes developing, maintaining, and supporting critical apps easier and less risky

Major Features

- Easy to learn graphical user interface with data visualization
- Parallel Debugging
  - MPI, Pthreads, OpenMP™, GA, UPC
  - CUDA™, OpenACC®, and Intel® Xeon Phi™ coprocessor
- Low tool overhead resource usage
- Includes a Remote Display Client which frees 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



TotalView for the NVIDIA ® GPU Accelerator



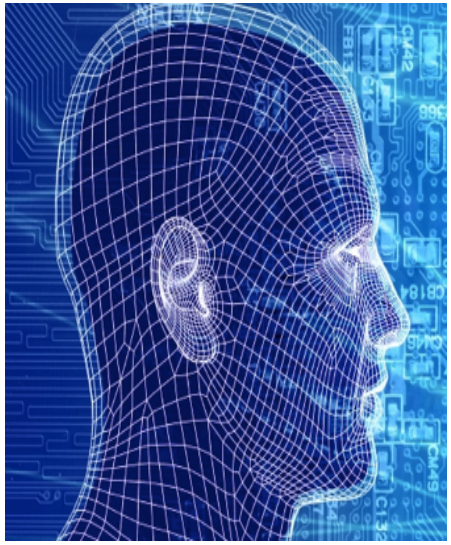
- NVIDIA Kepler, Fermi, Maxwell
- NVIDIA CUDA 5.5, 6.0, 6.5
  - With support for Unified Memory
- Cray CCE OpenACC
- Features and capabilities include
  - Support for dynamic parallelism
  - Support for MPI based clusters and multi-card configurations
  - Flexible Display and Navigation on the CUDA device
    - Physical (device, SM, Warp, Lane)
    - Logical (Grid, Block) tuples
  - CUDA device window reveals what is running where
  - Support for types and separate memory address spaces
  - Leverages CUDA memcheck

HRL Laboratories

- “In the first full day of using TotalView, we were quickly able to solve the bug that had us stumped for weeks. “

  Kirill Minkovich, HRL Laboratories

- Neural Network Simulation in the Defense Industry
- Need to adopt new technology *and* achieve massive scaling on a deadline
- Project went from failing to over-achieving



To learn more

- Visit

www.roguewave.com

  - Sign up for an evaluation
  - Documentation
  - Videos
  - Case Studies
  - Forum

- Email

Chris.Gottbrath@roguewave.com
sales@roguewave.com

Resource Slides