

TotalView Tutorial

Nikolay Piskun
Director of Continuing Engineering
Rogue Wave Software

Tutorial Agenda

- **Introduction to Rogue Wave and TotalView**
- **GPU accelerator architecture challenges**
- **Crash course on debugging CUDA with TotalView**

Rogue Wave Synergies

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 mathematical and statistical algorithms and visualization software for data-intensive applications



ROGUE WAVE
SOFTWARE

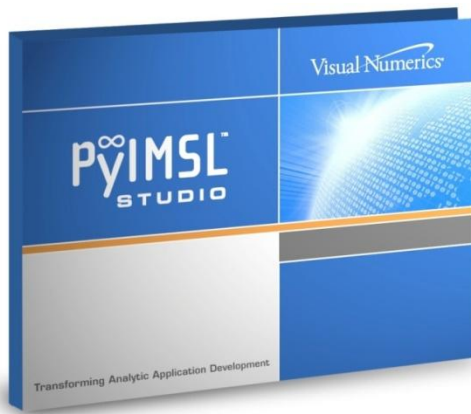
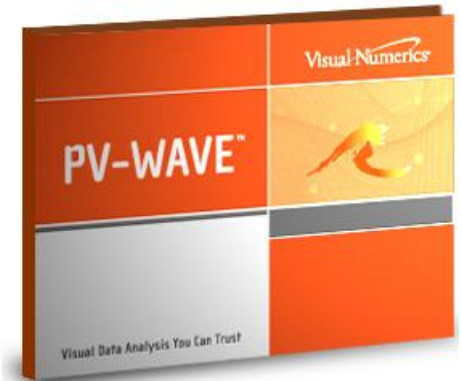
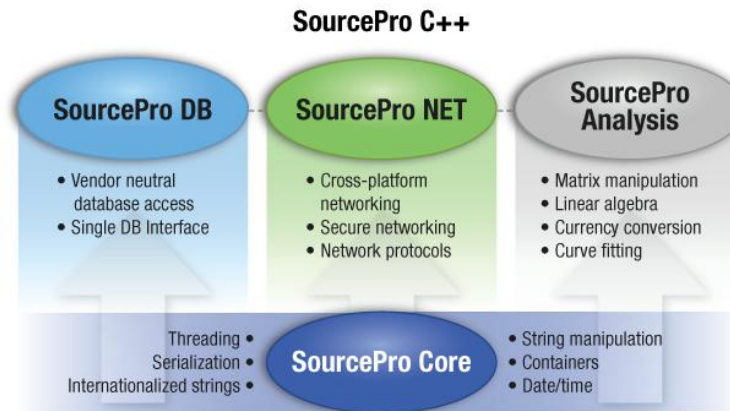
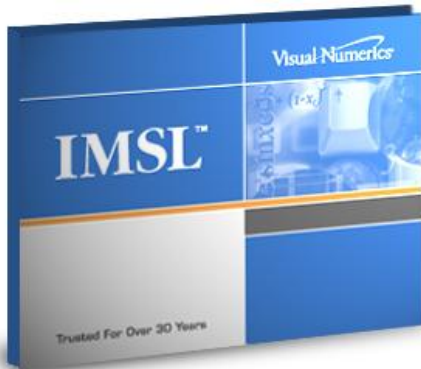
Leading provider of enterprise class C++ components and infrastructure for high performance applications



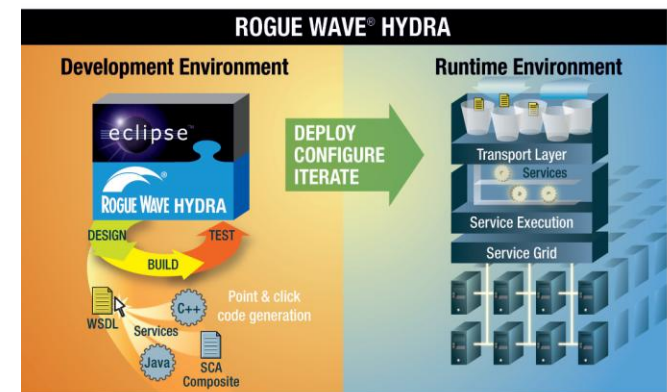
TOTALVIEW
TECHNOLOGIES

Industry-leading interactive analysis and debugging tools for the world's most sophisticated software applications.

Rogue Wave Major Product Offerings



O/S & Hardware
Most widely ported C++ framework available: Over 60 platform/compiler combinations



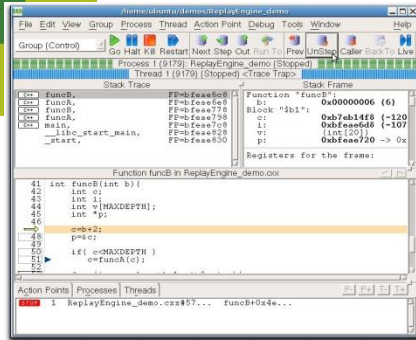
Eclipse is a trademark of the Eclipse Foundation, Inc.

TotalView Debugging Ecosystem

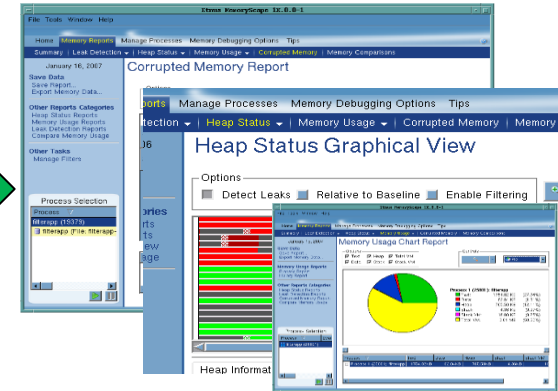
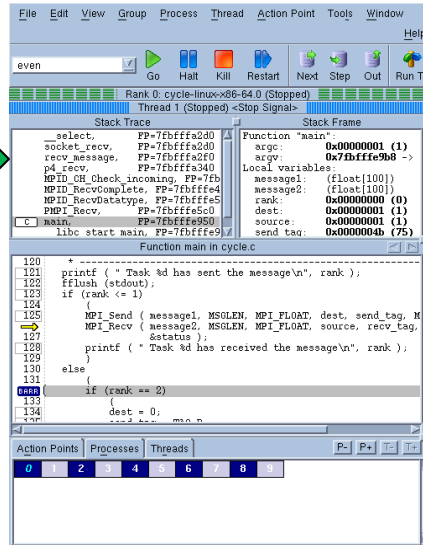
Reverse Debugging with **ReplayEngine** Debugging with **TotalView**

Memory Debugging with **MemoryScope**

Next
Step

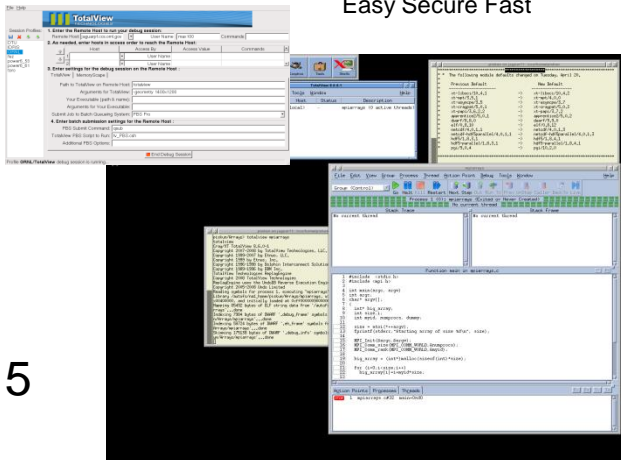


Prev
UnStep



Remote Display Window

Easy Secure Fast



Batch Debugging with TVScript

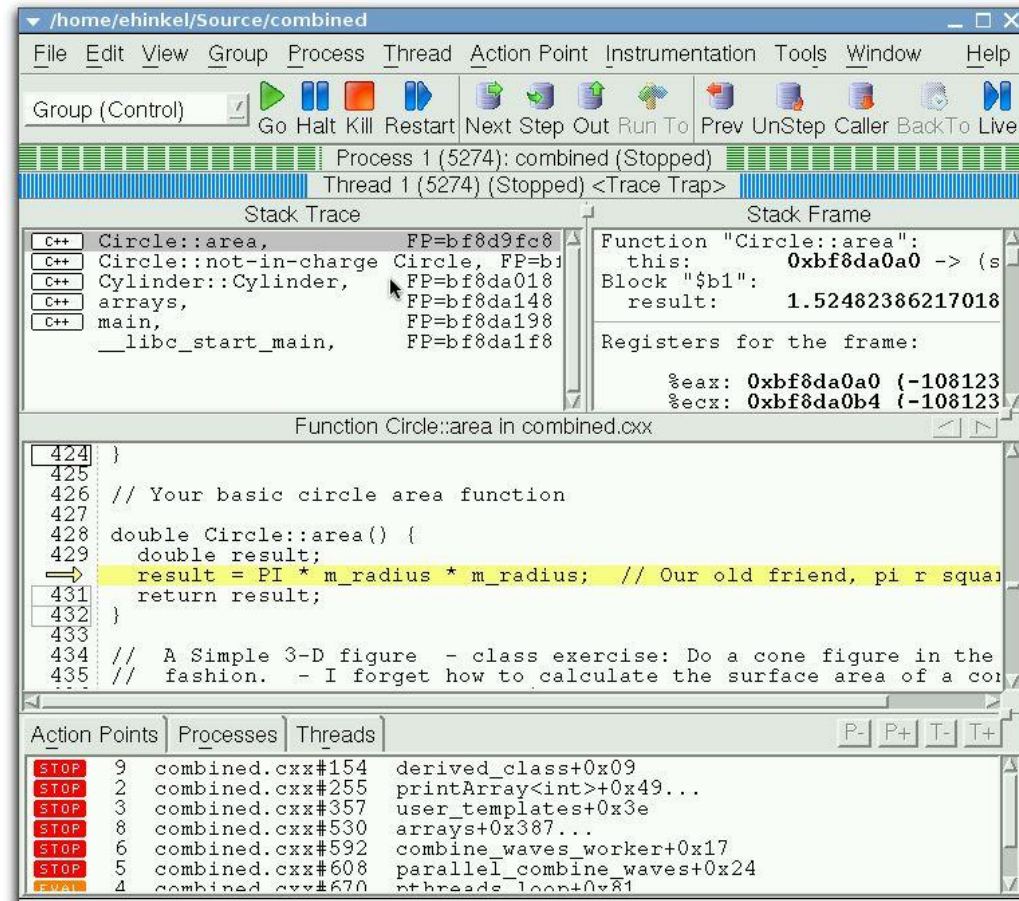
```

Print
Process:
./server (Debugger Process ID: 1, System ID: 12110)
Thread:
Debugger ID: 1.1, System ID: 3083946656
Time Stamp:
06-26-2008 14:04:09
Triggered from event:
actionpoint
Results:
foreign_addr = {
  sin_family = 0x0002 (2)
  sin_port = 0x1fb6 (8118)
  sin_addr = {
    s_addr = 0x6658a8c0 (1717086400)
  }
  sin_zero = ""
}
    
```

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
 - Parallel Debugging
 - MPI, PVM, OpenMP
 - Remote and Client/Server Debugging
- Integrated Memory Debugging
- Reverse Debugging available
 - ReplayEngine add on
- Supports a Variety of Usage Models
 - Powerful and Easy GUI
 - Visualization
 - CLI for Scripting
 - Long Distance Remote Debugging
 - Unattended Batch Debugging



What is MemoryScape?

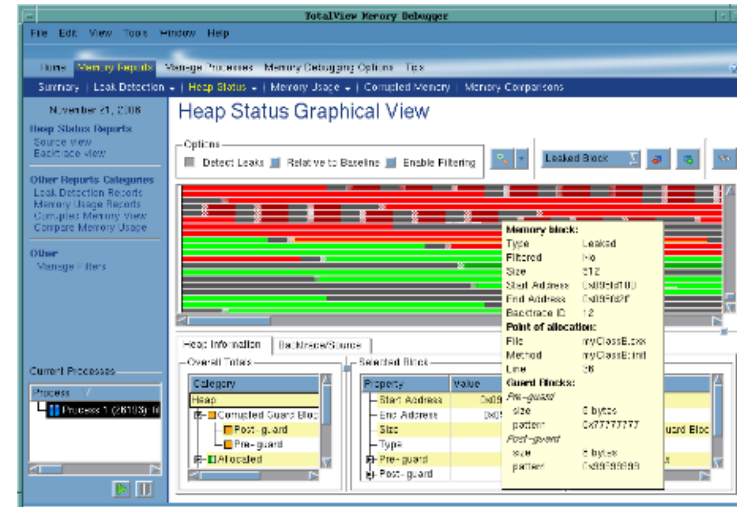
Simple to use, intuitive memory debugging

Memory Debugger

- Streamlined
- Lightweight
- Intuitive
- Collaborative

Features

- Shows
 - Memory errors
 - Memory status
 - Memory leaks
 - Buffer overflows
- MPI memory debugging
- Remote memory debugging



- Tech
 - Low overhead
 - No Instrumentation
- Interface
 - Inductive
 - Multi-process

TotalView and MemoryScape

You can use TotalView and MemoryScape together on an application.

More precise control

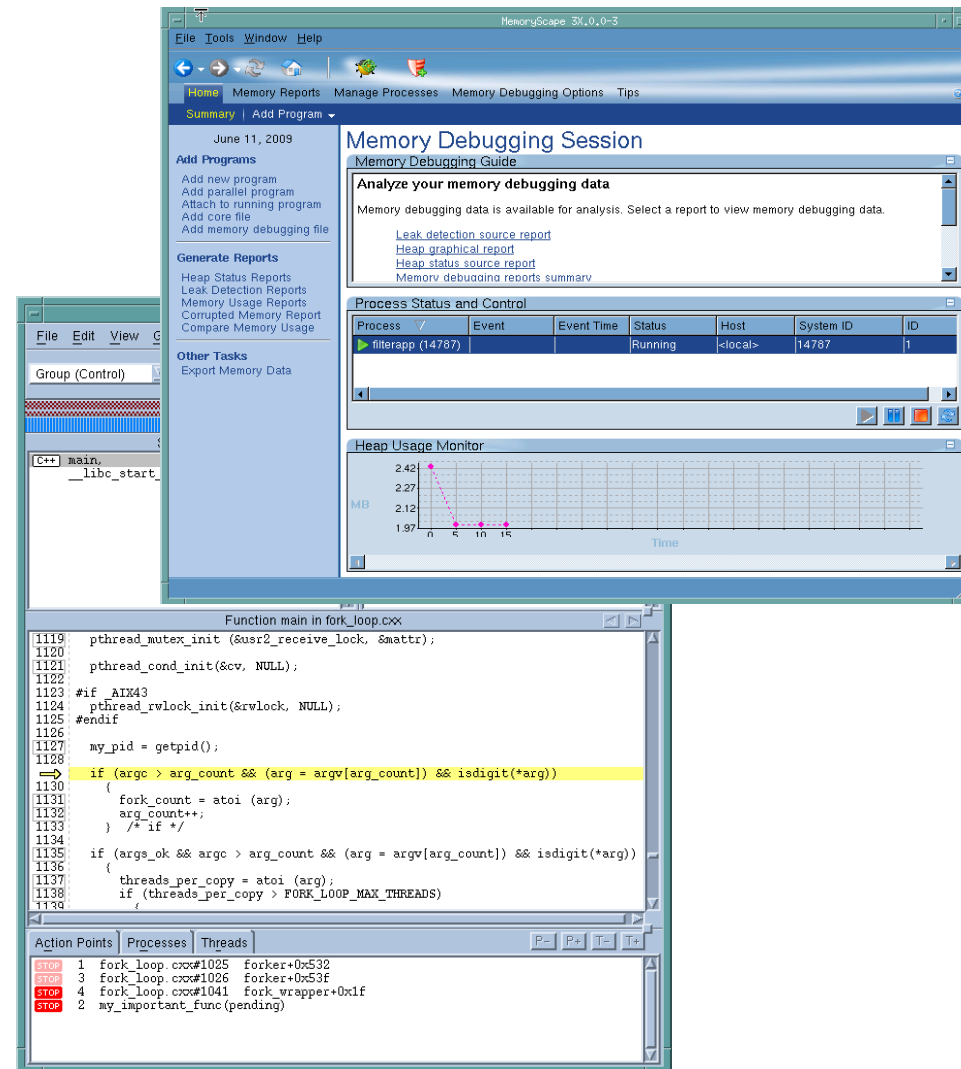
- Breakpoints
- Stepping

Visibility into

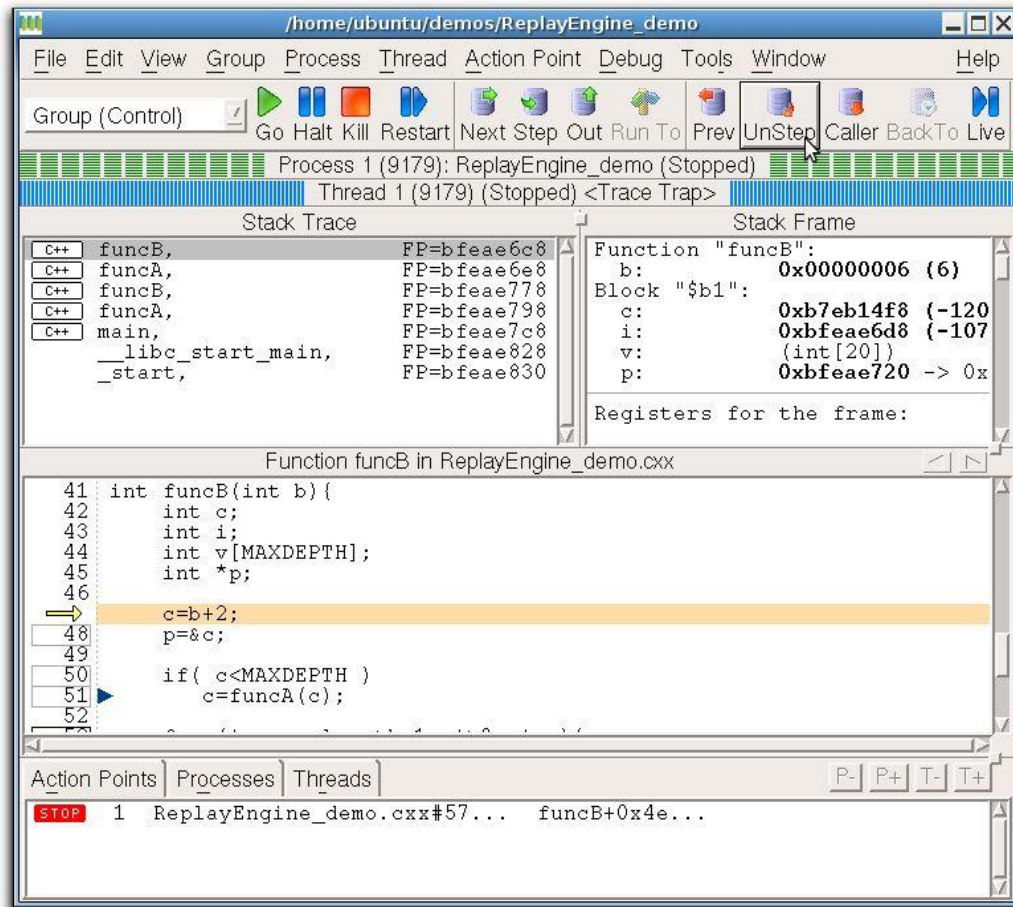
- Memory behavior
- Variable contents

Advanced features

- Heap baseline
- Dangling pointer annotation
- Memory painting



Replay Engine



- **Captures execution history**

- Records all external input to program
- Records internal sources of non-determinism

- **Replays execution history**

- Examine any part of the execution history
- Step back as easily as forward
- Jump to points of interest

- **An add-on product to TotalView**

- Support for
 - Linux/x86
 - Linux x86- 64

TotalView Debugger for CUDA

NVIDIA GPU accelerator architecture

- **Used in conjunction with conventional CPUs**
 - Acts as an accelerator to a host process
 - Host processes may be clustered together using MPI
- **Distinct processor architecture**
 - Compared to host CPU
 - Features vector instructions
- **Many more cores than an SMP**
 - Hundreds of streaming multiprocessors
 - Potentially 10k+ thread contexts
- **Hierarchical memory with more layers**
 - Local (thread)
 - Shared (block)
 - Global (GPU)
 - System (host)

GPU Compute Accelerators

● Technology Trends

- CPU Processors Multi-Core
- GPUs have very many extremely simple cores
- Leverages gaming/graphics market

● Multiple Vendors

- NVIDIA Tesla and Fermi
- AMD Firestream

● Multiple Potential Language/Runtime Choices

- NVIDIA CUDA for C
- OpenCL
- PGI CUDA for Fortran
- CAPS HMPP

Programming for the GP-GPU

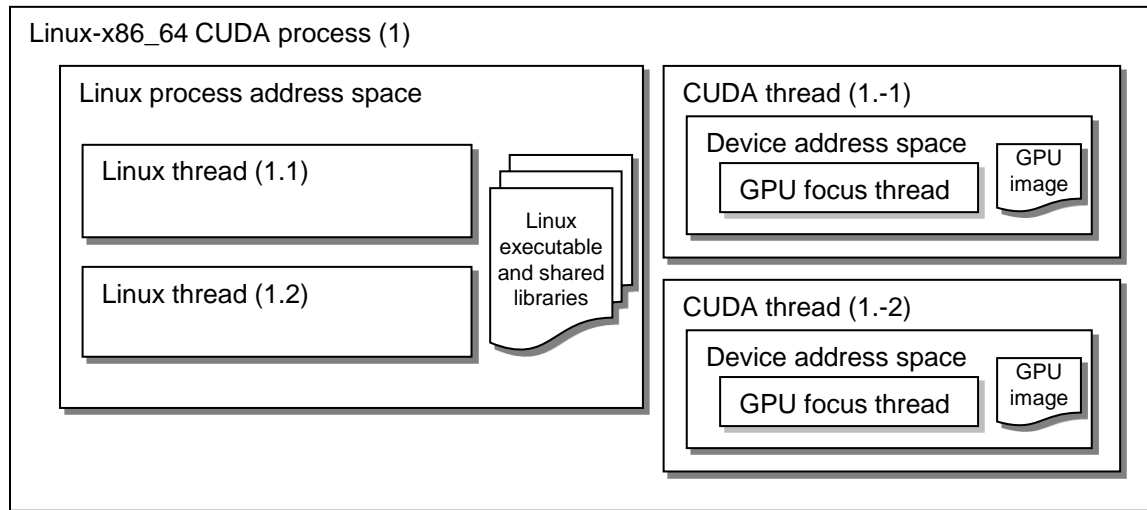
● CUDA

- Function-like kernels are written for the calculations to be performed on the GPU
 - Data parallel style, one kernel per unit of work
- Presents a hierarchical organization for thread contexts
 - 2D 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

TotalView CUDA Debugging Model



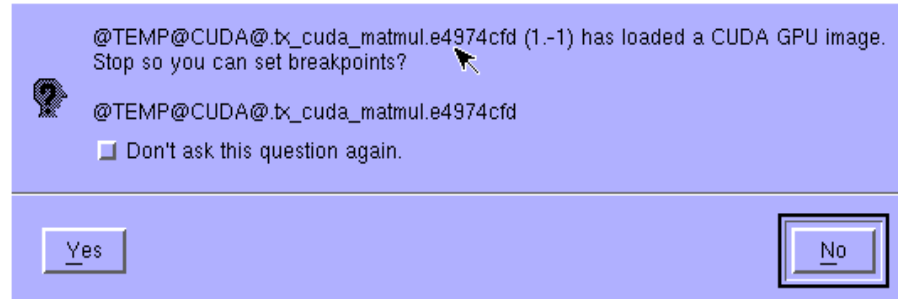
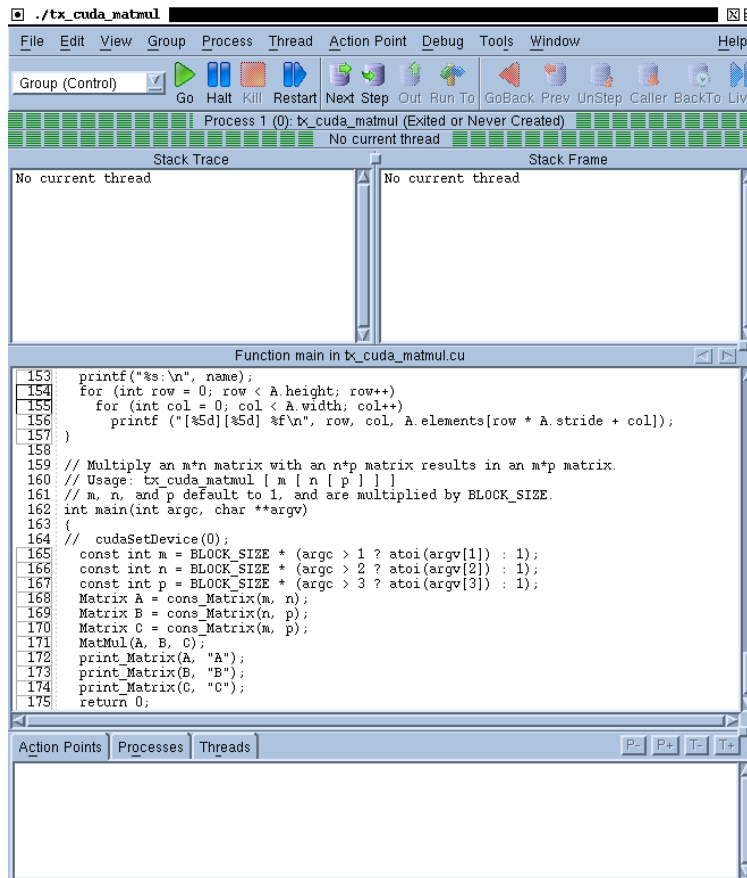
● A Linux-x86_64 CUDA process consists of:

- 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

Compiling for debugging (SDK 3.0)

- `nvcc -g -G -c tx_cuda_matmul.cu -o tx_cuda_matmul.o`
- `nvcc -g -G -Xlinker=-R/usr/local/cuda/lib64 \ tx_cuda_matmul.o -o tx_cuda_matmul`
- **Compiling for Fermi**
 - `-gencode arch=compute_20,code=sm_20`
- **Compiling for Fermi and Tesla**
 - `-gencode arch=compute_20,code=sm_20 -gencode arch=compute_10,code=sm_10`

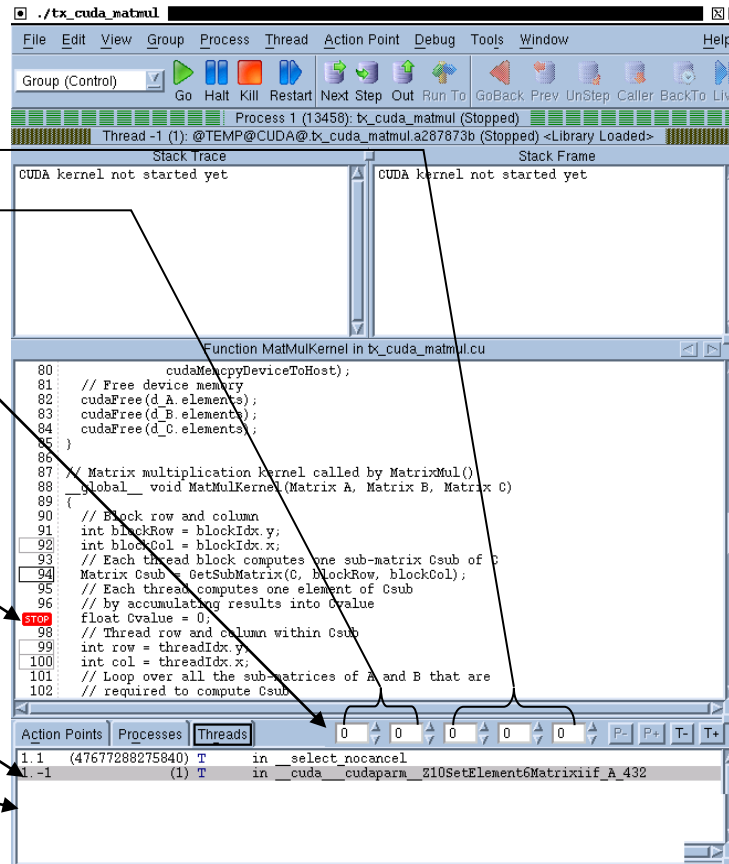
Starting TotalView



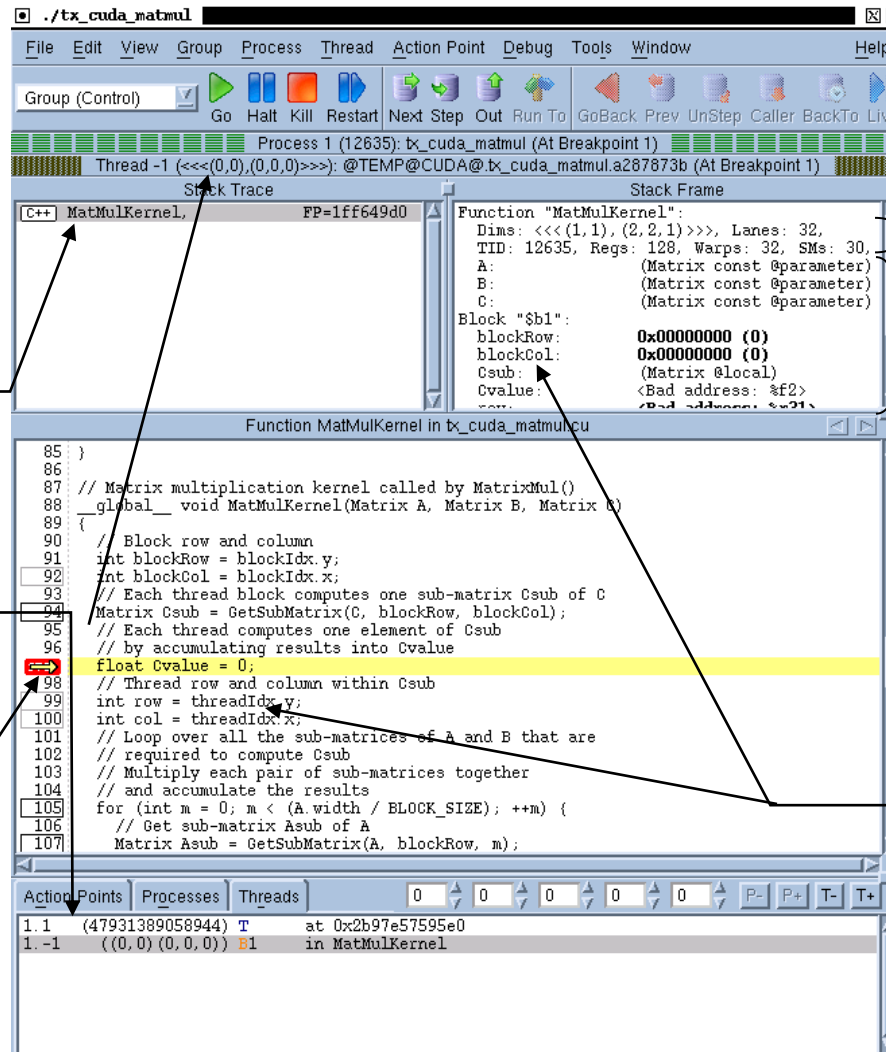
- When a new kernel is loaded you get the option of setting breakpoints

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

Debugging CUDA



Running to a Breakpoint in the GPU code



The screenshot shows the TotalView debugger interface for a GPU application. The main window displays the source code of the `MatMulKernel` function, which is currently paused at a breakpoint on line 97. The code includes comments and logic for matrix multiplication on the GPU, such as setting block dimensions and computing sub-matrices.

Annotations with arrows point to various parts of the interface:

- Stack backtrace and inlined functions:** Points to the 'Stack Trace' window showing the current thread and function.
- GPU focus thread logical coordinates:** Points to the thread list showing 'Thread -1' with coordinates `((0,0),(0,0,0))`.
- PC arrow for the warp:** Points to the red arrow on line 97 in the source code.
- Function details:** Points to the 'Function "MatMulKernel"' window showing metadata like dimensions, lanes, and registers.
- Parameter, register, local and shared variables:** Points to the 'Function MatMulKernel in tx_cuda_matmul.cu' window.
- Dive on a variable name to open a variable window:** Points to the `Csub` variable in the source code.

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

Stack backtrace and inlined functions

GPU focus thread logical coordinates

PC arrow for the warp

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”.

Storage Qualifiers

Denotes location in hierarchical memory

- Part of the type
- Each memory space has a separate address space so 0x00001234 could mean several places

Storage Qualifier	Meaning
@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 (see below).

Figure 7. Storage qualifier names

GPU Variables and Data Display

A - MatMulKernel - 1.-1

File Edit View Tools Window Help

1.-1

Expression: A Address: 0x00000010

Type: @parameter const Matrix

Field	Type	Value
width	int	0x00000002 (2)
height	int	0x00000002 (2)
stride	int	0x00000002 (2)
elements	float @global *	0x00110000 -> 0

“@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

CUDA Built-In Runtime Variable

● The supported **CUDA built-in runtime variables** are as follows:

- `struct dim3_16 threadIdx;`
- `struct dim2_16 blockIdx;`
- `struct dim3_16 blockDim;`
- `struct dim2_16 gridDim;`
- `int warpSize;`

Device Threads and Warps

- **Warps advance synchronously**

- They 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

Device Thread Navigation

● Two coordinate systems

● Logical and Hardware

- Logical: 2D Grid of Blocks, 3D Thread Within Grid

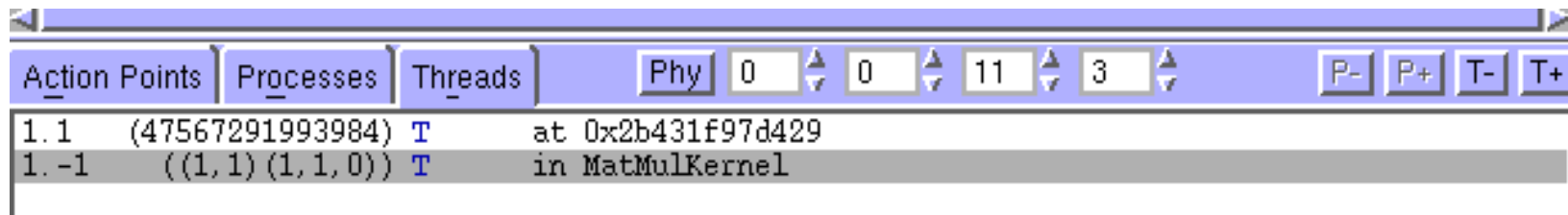
- Hardware: Device, SM, Warp, Lane

● Toggle to switch input

● Spinboxes

● Invalid selections are refused

● Logical coordinates are displayed elsewhere in the GUI



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

Name	Description
[-] Device 0/1	
[-] Device Type	gt206
[-] Lanes	32
[-] SM 0/1	
[-] Valid Warps	00000000000007c00
[-] Warp 10/32	Block (0,1)
[-] Lane 0/32	Thread (0,0,0)
LPC	000000001c97e540
[-] Lane 1/32	Thread (1,0,0)
LPC	000000001c97e540
[-] Lane 2/32	Thread (0,1,0)
LPC	000000001c97e540
[-] Lane 3/32	Thread (1,1,0)
LPC	000000001c97e540
[-] Valid/Active/Divergent	0000000f, 0000000f, 00000000
[-] Warp 11/32	Block (1,1)
[-] Lane 0/32	Thread (0,0,0)
LPC	000000001c97e360
[-] Lane 1/32	Thread (1,0,0)
LPC	000000001c97e360
[-] Lane 2/32	Thread (0,1,0)
LPC	000000001c97e360

GP-GPU Beta

Way for users to participate earlier in CUDA debugging with TotalView

- Provide feedback into development efforts
- Find out
 - How to manage and control GPU threads
 - How to see data from thousands threads
 - How to debug accelerated clusters using MPI and CUDA
- Sign up now
 - <http://www.totalviewtech.com/>
 - Contact :Nikolay.Piskun@roguewave.com

Thanks!

QUESTIONS?

www.roguewave.com

www.totalviewtech.com