

#### TotalView Training - NERSC

MAY 13, 2024

TotalView by Perforce © Perforce Software, Inc.

# LBL/NERSC Agenda – May 2024

- Introduction
- Latest TotalView Features
- TotalView Roadmap
- Remote Debugging Techniques
- Review of General Debugging Features
- TotalView Debugging at NERSC Best Practices
- GPU Debugging with TotalView on Perlmutter (10:00am)

- MPI and OpenMP debugging
- Reverse Debugging
- Memory Debugging
- Common TotalView Usage Questions
- Q&A

#### Introductions

• Bill Burns (Senior Director of Software Engineering and Product Manager)

bburns@perforce.com

• John DelSignore (TotalView Chief Architect)

jdelsignore@perforce.com

• Larry Edelstein (Manager, Sales Engineering, TotalView)

ledelstein@perforce.com

# PERFORCE

#### Solving the Hardest Challenges in DevOps



#### **Perforce Products**





Overview of TotalView Labs

## Overview of TotalView Labs

#### Nine different labs and accompanying example programs

- Lab 1 Debugger Basics: Startup, Basic Process Control, and Navigation
- Lab 2 Viewing, Examining, Watching, and Editing Data
- Lab 3 Examining and Controlling a Parallel Application
- Lab 4 Exploring Heap Memory in an MPI Application
- Lab 5 Debugging Memory Comparisons and Heap Baseline \*
- Lab 6 Memory Corruption discovery using Red Zones \*
- Lab 7 Batch Mode Debugging with TVScript
- Lab 8 Reverse Debugging with ReplayEngine
- Lab 9 Asynchronous Control Lab

#### Notes

- \* Labs 5 and 6 require use of TotalView's Classic UI
- Sample program breakpoint files were created with GNU compilers. If a different compiler is used, they may not load and will need to be recreated.
- Several example programs use OpenMPI so you will need to configure your environment beforehand.
- We do not have a lab specific to Python Debugging yet. There are good examples and instructions in the TotalView totalview.<version>/<linux-x86-64>/examples/PythonExamples directory.
- Use this slide deck for GPU specific debugging information

## TotalView Features

## What is TotalView used for?

- Provides interactive Dynamic Analysis capabilities to help:
  - Understand complex code
  - Improve code quality
  - Collaborate with team members to resolve issues faster
  - Shorten development time
- Finds problems and bugs in applications including:
  - Program crash or incorrect behavior
  - Data issues
  - Application memory leaks and errors
  - Communication problems between processes and threads
  - CUDA application analysis and debugging
- Contains batch and Continuous Integration capabilities to:
  - Debug applications in an automated run/test environment



## **TotalView Features**

- Multi-process/thread dynamic analysis and debugging
- Comprehensive C, C++ and Fortran Support
- Thread specific breakpoints with individual thread control
  - View thread specific stack and data
- View complex data types easily
- MPI, OpenMP, Hybrid support
- NVIDIA (CUDA) and AMD (HIP) GPU support
- Convenient remote debugging
- Integrated Reverse debugging
- Mixed Language Python C/C++ debugging
- Memory debugging
- Script debugging
- Linux, macOS and UNIX
- More than just a tool to find bugs
  - Understand complex code
  - Improve developer efficiency
  - Collaborate with team members
  - Improve code quality
  - Shorten development time

			Tł	nreadWo	rkers - P	rocess 1, Thread 1.4 (G	Thread) (Brea	(point) - TotalView	2020			_ = ×
<u>File Edit Group Process Thread</u>	Action Points	Debug	Window	Help								
Thread 🗧 🕨 🔢 📕 IÞ	2 2 2	E I			ReplayEngine			4				
Processes * Lookup File o * Doc	× )pe ×	Start I	⊃age ×	main.cpp	×poll_	nocancel × worker.cpp ×			Call St	tack 🗶		
1 2 3 =		38 39	void Wo	rker::doW	ork()				<b>T</b>			
Description	# P #	# 40 41	{ aDebu	a()<<"Sta	rting wor	ker process in Thread "	< <thread()->cu</thread()->	rentThreadId():	C++	Worker::doW	ork	
<ul> <li>ThreadWorkers (S3)</li> </ul>	1 .	1 42	90000	d(worker)		ter process in fineau			(CH)	Worker::at s	tatic metacall	
Tixed	1	<b>43</b> 1 44	do {	d(worker)	,					700000		
Breakpoint	1	45 1 46	mut boo	ex.lock() l doAbort	; = abort;					_ZN11QMeta	aObject8activateEP7Q	ObjectPKS_IPPv
Worker::doWork	1	47 1 48	mut	ex.unlock	();					_ZN7QThrea	d11setPriorityENS_8F	PriorityE
worker.cpp#49	1	49 1 50	swi cas	tch (work e 1:	er) {					start_thread		
1.4	1	51 52	q	Debug() <	< "Worker	1 doing work!";				clone		
Running	1	53	cas	e 2:	< "Worker	1 doing work!"						
<ul> <li><unknown address=""></unknown></li> </ul>	1	1 55	b	reak;	< WOLKET	I doing work: ,						
<ul> <li><unknown address=""></unknown></li> </ul>	1	1 57	q	e 3: Debug() <	< "Worker	1 doing work!";						
1.1	1	1 59	58 break; 59 case 4: 60 case () cc "Merchan 4 daine work:"									
Stopped	1	3 61	b	reak;	< worker	I doing work: ,						=
poll_nocancel	1	3 63										-
<ul> <li><unknown line=""></unknown></li> </ul>	1	3 Action	Points	× Data Vi	ew 🗶 Re	play Bookmarks 🗶 Commar	nd Line × Input/	Dutput ×	Local	Variables ×	Type	Value
1.2	1	1	ID❤	Туре	Stop	Location		Line		rauments	type	Value
1.3	1	1	1	Eval	N/A	/worker.cpp#49	work	er.cpp (line 49)		this	Worker * const	0x022241a0 -> (Worker)
1.5	1	1							T BI	ock at Line 45		
										doAbort	bool	false (0)
										loop	OEventleen	
										юор	QEvenirooh	(Opaque QEveniLoop)
	_		_	_								
Process: 1 (2911) ThreadWorkers T		3d414f970	)0) - Breal	kpoint	Frame: Wo	orker::doWork File: /hon	ne/bburns/Demos/T	nreadWorkers/worker.cop		ne: 49	_	Source Line: 49

Recent TotalView Features

# TotalView Remote Client for Windows

- TotalView 2024.1 adds native Windows remote client support
- Combine the convenience of establishing a remote connection to a cluster and the ability to run the TotalView GUI locally
- Front-end GUI architecture does not need to match back-end target architecture (macOS front-end -> Linux back-end)
- Secure communications
- Convenient saved sessions
- Once connected, debug as normal with access to all TotalView features
- Windows, macOS and Linux native front-ends



# TotalView 2024.1 Platform Updates

#### **Platform / Compiler Updates**

- macOS Sonoma
- AMD GPU ROCm 6.0 and MI300

#### **Other Updates**

- Various bug fixes and other minor enhancements
- Third-party open-source package updates security



# **Other Recent TotalView Updates**

- Assembly and Register View
- C++ Type Transformations
  - Iterator support
  - Additional container classes
- Array Debugging
  - Array View
  - Array Visualization
- Apple ARM M1/2/3 support
- Memory Debugging Additions
  - Hoarding and Painting
  - Buffer overwrite detection

Category	Class	Iterator kind	Transformed type
Sequence	array (C++11)	Random	Array (Dense)
	vector	Random	Array (Dense)
	deque	Random	Array (Sparse)
	forward_list (C++11)	Forward	List
	list	Bidirectional	List
Associative	set	Bidirectional	Tree
	multiset	Bidirectional	Tree
	map	Bidirectional	Tree
	multimap	Bidirectional	Tree
Unordered associative	unordered_set	Forward, Local	Hashtable
	unordered_multiset	Forward, Local	Hashtable
	unordered_map	Forward, Local	Hashtable
	unordered_multimap	Forward, Local	Hashtable
Adaptors	stack (deque,list,vector)		Struct
	queue (deque,list)		Struct
	priority_queue		Struct
	(deque,vector)		
General utilities	pair		Struct
	tuple (C++11)		Struct
Memory management	unique_ptr (C++11)		Struct
	shared_ptr (C++11)		Struct
	weak_ptr (C++11)		Struct
Numeric	complex		Struct
Strings	string		Struct

TotalView Memory Debugging

# What is a Memory Bug?

- A Memory Bug is a mistake in the management of heap memory
  - Leaking: Failure to free memory
  - Dangling references: Failure to clear pointers
  - Failure to check for error conditions
  - Memory Corruption
    - Writing to memory not allocated
    - Overrunning array bounds



# TotalView Heap Interposition Agent (HIA) Technology

**TotalView** 

- Advantages of TotalView HIA Technology
  - Use it with your existing builds
    - No Source Code or Binary Instrumentation
  - Programs run nearly full speed
    - Low performance overhead
  - Low memory overhead
    - Efficient memory usage



# The Agent and Interposition



# The Agent and Interposition



#### , , ,

## Memory Debugging in TotalView's New UI

-

#### **TotalView 2024.1 Features**

- Leak detection
- Dangling pointer detection
- Heap allocation overview
- Automatically detect allocation problems
- Memory Corruption Detection Guard Blocks
- Memory Block Painting
- Memory Hoarding

#### **Coming Features**

- Graphical heap view
- Memory Corruption Detection Red Zones
- Memory Comparisons between processes

	git - Process 1, Thread 1.1 (Br	reakpoint) - TotalView 2021	_ = ×
Group Process Thread Action Points	ebug Window Help		
ontrol) 🗧 🕨 🚺 📕 🕪 🛆 🖄			
	Start Page × wrapper.c × common-main.c × tv_heap_breakpoint.c ×	Leak Report <p1> ×</p1>	Call Stack ×
3 =	26         die("attempting to allocate %"PRIuMAX" over limit %"PRIuMAX           27         (uintmax_t)size, (uintmax_t)limit);	Process: git UPDATE	<b>Y</b>
#P #	20 } 29 return 0; 30 }	Drag tab to detatch. Report may be placed side-by-side with source.	TV_HEAP_notify_breakpoint_here
Breakpoint 1 1	<pre>31 32 try_to_free_t set_try_to_free_routine(try_to_free_t routine)</pre>	Process Bytes 💙 Count Begin Address End Address E	TV_HEAP_notify_tv
Breakpoint 1 1	33 { 34 try_to_free_t old = try_to_free_routine;	Process 1 (5095): git 1157.18 KB 8959	
TV_HEAP_notify_breakpoi 1 1	35 if (!routine) 36 routine = do_nothing;	vrapper.c 1157.18 KB 8959	TV_HEAP_malloc_interposer_fini
v_tv_heap_breakpoint.c#59 1 1	37 try_to_free_routine = routine; 38 return old;	▼ xstrdup 201 6	C TV_HEAP_linux_ia_fini
1.1 1 1	<b>39</b> } 40	Line 43 201 6	_dL_fini
	41 char *xstrdup(const char *str) 42 {	▶ xrealloc 332 6	run_exit_handlers
	43	b do_xmalloc 17.27 KB 133	exit
	45 try_to_free_routine(strlen(str) + 1); 46 ret = strdup(str); 47 life(ret)	Backtrace	handle_builtin
	48 die("Out of memory, strdup failed");	ID Function Line # Source Information	C run_argv
	50 return ret;	▶ 383	C cmd_main
	<b>51</b> } 52	▶ 93	C main
	53 static void "do_xmalloc(size_t size, int gentle) 54 {	▶ 92	libc_start_main
	55 void "ret; 56	▶ 85	<b>0</b>
	57 if (memory_limit_check(size, gentle)) 58 return NULL;	▶ 38	Local Variables ×
	59 ret = malloc(size); 60 if (!ret && !size)	▶ 36	Name Type Value
	61		<ul> <li>Arguments</li> </ul>
	Action Points * Data View * Replay Bookmarks * Command Line * Input/Output *		▶ event T 0x7fff8fb77840 -> (TV
	Linux x86_64 TotalView 2021.1.16		
	Thread 1.1 has appeared		
	Created process 1 (5005), named "git" Thread 1.1 has appeared Thread 1.1 has exited		
	Thread 1.1 has reported a heap tracking event (see dheap for more information) ${\rm dl}_* \hookrightarrow$		
(5095) git Thread: 1.1 (0x7f4f1be2a740) -	reakpoint Frame: TV_HEAP_notify_breakpoint_here File:works/totalview.2021.1.16/src/tv_heap_breakp	aint.c Line: 59	Source Line: 43

Memory Debugging Demo



Memory Debugging Demo

Debugging OpenMP Applications

## TotalView Support for OpenMP

- Source-level debugging of the original OpenMP code (C, C++, and Fortran)
- Debug code inside of OMP **parallel** and **task** regions
  - Set stop-thread breakpoints, single-step, etc.
  - View OMP **shared**, **private**, and **threadprivate** variables
- Debug code inside OMP target regions
  - On NVIDIA GPUs, similar to debugging CUDA code
  - On AMD GPUs, similar to debugging HIP code
- CORAL-2 OMP/OMPD support (scheduled for TotalView 2024.2 release)
  - Focuses on Clang, AMD Clang/Flang, and HPE CCE compilers
  - Evaluates quality DWARF debug information produced by the compiler
  - Adds TotalView OpenMP views to display the runtime state of regions, threads, control variables, and ICVs
  - OpenMP thread-stack transformations: filter-out OMP runtime frames, annotate parallel/task regions, and insert parent links
  - Demangles OpenMP outlined function names

## Debugging OpenMP Applications

- OpenMP programs are multi-threaded programs
  - Use normal debugging techniques for multi-threaded programs
  - Use stop-thread breakpoints inside parallel regions
  - Use thread-level execution controls: hold/unhold thread, single-step threads / thread groups
- OpenMP **parallel** and **task** regions are in outlined functions
  - A single line of source code can generate multiple block of machine code
  - Outlined function names are mangled by the compiler, but TV 2024.2 will support OMP name demangling
  - TV 2024.2 will support step into and out of OMP parallel region, if the compiler supports OMPD properly
  - Otherwise, set a breakpoint inside the parallel region and let the process run to it
- OpenMP target regions are offloaded to the GPU
  - Use normal debugging techniques for CUDA / HIP
  - Unfortunately, OMPD information is not available for target regions

# Enabling OMPD Support (TV 2024.2)

- TotalView OMPD support requires compiler support
  - Clang 15+, HPE CCE 17+, AMD Clang/Flang 17+
  - OMPD support is still maturing
  - Special linking rules might apply (check the doc)
- Set OMPD environment variable
  - OMP\_DEBUG=enabled
  - MPI + OpenMP codes require setting the environment variable in the MPI processes
  - Use to propagate OMP\_DEBUG setting
- Select "Enable OpenMP Debugging"
  - Good for non-MPI codes launched by the debugger
  - Does not work for processes that are attached to

<pre>• tx_omp_task - Process 1, No current</pre>	nt thu	hread - TotalView 2024X.2
File Edit Group Process Thread Action P	oints	Debug Window Help
Group (Control) 💠 🕨 📕 🕨 🖄	1	Enable ReplayEngine Create Replay Bookmark Ctrl+Shift+D
Processes & Threads × Lookup File or Function	×	Enable Memory Debugging Stop on Memory Events Generate Leak Report
	Mamb	Generate Heap Report
v tx_omp_task (S3)	p1	Memory Event Report Memory Options
Nonexistent	p1	Enable OpenMP Debugging     Enable CUDA Memcheck
		24setvbuf(stdout, 0, _IOLB25printf("%d: Hello from m26#pragma omp parallel num_t27{28printf("%d: Inside omp

# Example OpenMP Debugging Session (TV 2024.2)

- Example using AMD Clang 17
  - LLVM Clang works but has some issues
  - HPE CCE works but has some issues
  - gcc / gfortran works but OMPD support not tested
- Set stop-thread breakpoints inside parallel / task region
- Make sure the <u>whole</u> process is stopped so that the OpenMP runtime is in a consistent state

<pre>tx_omp_task - Process 1, 1</pre>	No current thread -	TotalView 2024X.2	図 巴
<u>File Edit Group Process Thread</u>	d Action Points Debug	Window Help	
Group (Control) 💠 🕨 📕		E 🖊 😰 ReplayEngine 💿 ┥ 🖆 🛃 E4 🕅 🗏 🕈 🗍 Memory	⊙ ♦ ♦ ₫ ♦
Processes & Threads * Lookup File	or Function 😕	Start Page * tx_omp_task.c *	Call Stack ×
1 2 3 =		12 void task1() 13 {	Ŧ
Description	Members	<pre>14 printf("%d: Hello from task1()\n", omp_get_thread_num()); /* STOP_WI 15 }</pre>	
▼ tx_omp_task (S3)	pl	16 void task2() 17 {	
Nonexistent	p1	<pre>18 printf("%d: Hello from task2()\n", omp_get_thread_num()); /* STOP_WI 19 }</pre>	No current thread
		20 21 int main(int argc, char **argv)	
		<pre>22 { 23 int nthreads = argc &gt; 1 ? atoi(argv[1]) : -1;</pre>	
		<pre>24 setvbuf(stdout, 0, _IOLBF, 0); 25 printf("%d: Hello from main()\n", omp_get_thread_num());</pre>	Info
		<pre>26 #pragma omp parallel num_threads(nthreads &lt; 1 ? return4() : nthreads) 27 {</pre>	
		<pre>28 printf("%d: Inside omp parallel\n", omp_get_thread_num()); 29 #pragma omp single /* STOP WITH ARGS(omp)</pre>	
		30 { printf("vdu Incide own single)all own get thread aum());	
		32 printi ( wat inside oup singletin , oup_get_tin ead_idum()),	
		Command Line * OcenMR * Data View * Action Points * Array View * Longer *	
			Local Variables X Degisters X
			Local variables · Registers ·
		OMPD not available or not initialized	
A Configure			
Process: 1 (0) tx_omp_task No_ci	urrent thread		

# OpenMP > Regions (TV 2024.2)

- Displays **parallel** and **task** regions
  - Aggregated view of all OpenMP threads
- "Regions" tab shows
  - Source-code line-number of OMP region
  - OMP implicit or explicit task function name
  - OpenMP threads that are in the region

Command Line * OpenMP * Data View	X Action Points X Array	View × Logger ×	
Task Line	Task Function	Members	
<ul> <li>/ (Parallel Regions)</li> </ul>		1:4[p1.1-4]	
tx_omp_task.c#26	.omp_outlined7	1:4[p1.1-4]	
/ (Task Regions)		1:4[p1.1-4]	
tx_omp_task.c#26	.omp_outlined7	1:4[p1.1-4]	
tx_omp_task.c#32	.omp_task_entry.	1:1[p1.1]	
tx_omp_task.c#34	.omp_task_entry6	1:1[p1.3]	
Regions Control Variables OMPD Info Three	ads ICVs		

# OpenMP Threads (TV 2024.2)

- Thread-oriented view of OMP threads
  - For the focus process
  - Current state plus nest of OMP generating task regions
- "Threads" tab shows
  - Debugger process/thread ID and OMP thread-num
  - Current state of OMP thread / region #
  - Wait ID / Parent (encountering thread) ID
  - Region flags
    - "i" implicit vs. explicit task
    - "p" active parallel region
    - "f" final task
  - Task function and source-code line-number
  - Runtime frame information (not shown)

Command L	ine × C	DpenMP × Da	ta View 🗶	Action F	Points 🕷 🛛 Array Vie	w × Logger ×
Thread ID	OMP	State/Region#	Wait Id/Pare	Flags	Task Function	Task Line
<b>v</b> p1.1	0	work_parallel	0x0	-p-	.omp_task_entry.	[tx_omp_task.c#32]
		region 0		-p-	.omp_task_entry.	[tx_omp_task.c#32]
		region 1	p1.4	ip-	.omp_outlined7	[tx_omp_task.c#26]
		region 2	p1.1	i	<unavailable></unavailable>	<unavailable></unavailable>
▶ p1.2	1	wait_barrier	0x0	ip-	.omp_outlined7	[tx_omp_task.c#26]
▶ p1.3	2	work_parallel	0x0	-p-	.omp_task_entry6	[tx_omp_task.c#34]
▶ p1.4	3	wait_barrier	0x0	ip-	.omp_outlined7	[tx_omp_task.c#26]
Regions C	ontrol Varia	ables OMPD Info	D Threads	ICVs		

# OpenMP > ICVs (TV 2024.2)

- Hierarchical view of OpenMP internal control variables
  - For the focus process
- Organized by OpenMP scope
  - Global / address space scope
  - Thread scope
  - Parallel region scope
  - Task / implicit task scope

Command Line * OpenMP * Data View * A	Action Points X Array View X Logger X
ICV Name	Value
<ul> <li>Process 1 (global / adddress space scope)</li> </ul>	
affinity-format-var	"OMP: pid %P tid %i thread %n b
cancel-var	0
debug-var	1
display-affinity-var	0
max-task-priority-var	0
num-procs-var	32
stacksize-var	8388608
tool-libraries-var	
tool-var	1
tool-verbose-init-var	
Thread 1.1 (thread scope)	
default-device-var	0
dyn-var	0
nthreads-var	"32"
thread-num-var	0
<ul> <li>Region 0 (task / implicit task scope)</li> </ul>	
bind-var	0
final-task-var	0
implicit-task-var	0
max-active-levels-var	1
run-sched-var	"static,0"
thread-limit-var	2147483647
Regions Control Variables OMPD Info Threads I	CVs

# **OpenMP Stack Transformations**



## **OpenMP Debugging Caveats**

- OpenMP support prior to TotalView 2024.2 is a prototype
  - Not fully supported w/ limited compiler support
  - Has bugs and other problems with OpenMP displays
- LLVM/Clang-based compilers do <u>not</u> do a good at generating DWARF debug information
  - Program variables inside regions are all marked artificial, so TotalView does not display them
  - Use TotalView "-compiler\_vars" option to display program variables, but compiler-generated variables are also displayed
  - Parallel "for" loop variables do not have correct values
  - Many other DWARF debug information problems exist
- GNU compilers seem to do a much better job in general
- Linking applications with OMPD support varies by compiler
  - Check the documentation

# Debugging NVIDIA GPUs and CUDA with TotalView

# TotalView for the NVIDIA <sup>®</sup> GPU Accelerator

- NVIDIA Tesla, Fermi, Kepler, Pascal, Volta, Turing, Ampere, Hopper
- NVIDIA CUDA 9.2, 10, 11 and 12
  - With support for Unified Memory
- NVIDIA and Cray OpenACC support
- 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
  - Support for types and separate memory address spaces
  - GPU Status view reveals what is running where


### TotalView CUDA Debugging Model



## GPU Memory Hierarchy

- Hierarchical memory
  - Local (thread)
    - Local
    - Register
  - Shared (block)
  - Global (GPU)
    - Global
    - Constant
    - Texture
  - System (host)



#### Supported Type Storage (aka, Address Space) Qualifiers

@generic @frame @global @local @parameter @iparam @oparam @shared @surface @texsampler @texture @rtvar @register @sregister

An offset within generic storage An offset within frame storage An offset within global storage An offset within local storage An offset within parameter storage Input parameter Output parameter An offset within shared storage An offset within surface storage An offset within texture sampler storage An offset within texture storage Built-in runtime variables A PTX register name A PTX special register name

### 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"

#### NVIDIA GPU and CUDA Parallelization

- CUDA uses the single instruction multiple thread (SIMT) model of parallelization.
- CUDA GPUs made up of many computing units called cores
  - Cores includes an arithmetic logic unit (ALU) and a floating-point unit (FPU).
- Cores collected into groups called streaming multiprocessors (SMs).
- Computing tasks are parallelized by breaking them into numerous subtasks called threads.
- Threads are organized into blocks.
- Blocks are divided into warps whose size matches the number of cores in an SM.
- Each warp gets assigned to a particular SM for execution. GPUs have one or more SMs.
- SM control unit directs each of its cores to execute the same instructions simultaneously for each thread in the assigned warp.

## Compiling for CUDA debugging

When compiling an NVIDIA CUDA program for debugging, it is necessary to pass the **-g -G** options to the nvcc compiler driver. These options disable most compiler optimization and include symbolic debugging information in the driver executable file, making it possible to debug the application.

```
% /usr/local/bin/nvcc -g -G -c tx cuda matmul.cu -o tx cuda matmul.o
```

```
% /usr/local/bin/nvcc -g -G -Xlinker=-R/usr/local/cuda/lib64 \
tx cuda matmul.o -o tx cuda matmul
```

```
% ./tx_cuda_matmul
A:
[ 0][ 0] 0.000000
...output deleted for brevity...
[ 1][ 1] 131.000000
```

## Compiling for a specific GPU architecture (avoids JIT'ing from PTX)

#### Compiling for Ampere

-gencode arch=compute\_80,code=sm\_80

#### **Compiling for Volta** -gencode arch=compute\_70,code=sm\_70

**Compiling for Pascal** -gencode arch=compute\_60,code=sm\_60

#### **Compiling for Kepler**

-gencode arch=compute\_35,code=sm\_35

#### **Compiling for Fermi and Tesla**

-gencode arch=compute\_20,code=sm\_20 –gencode arch=compute\_10,code=sm\_10

#### **Compiling for Fermi**

-gencode arch=compute\_20,code=sm\_20

#### A TotalView Session with CUDA

A standard TotalView installation supports debugging CUDA applications running on both the host and GPU processors.

TotalView dynamically detects a CUDA install on your system. To start the TotalView GUI or CLI, provide the name of your CUDA host executable to the totalview or totalviewcli command.

For example, to start the TotalView GUI on the sample program, use the following command:

#### % totalview tx\_cuda\_matmul

\* This example is just a single node, no MPI application

#### Source View Opened on CUDA host code

```
Start Page × tx_cuda_matmul.cu ×
        Matrix A;
 139
 140
        A.width = width ;
 141
        A.height = height_;
 142
        A.stride = width_;
 143
        A.elements = (float*) malloc(sizeof(*A.elements) * width_ * height_);
 144
        for (int row = 0; row < height_; row++)</pre>
 145
          for (int col = 0; col < width ; col++)</pre>
 146
            A.elements[row * width + col] = row * 10.0 + col;
 147
        return A;
 148
      3
149
150 static void
 151
      print_Matrix (Matrix A, const char *name)
152 {
153
        printf("%s:\n", name);
 154
        for (int row = 0; row < A.height; row++)</pre>
 155
          for (int col = 0; col < A.width; col++)</pre>
 156
            printf ("[%5d][%5d] %f\n", row, col, A.elements[row * A.stride + col]);
 157 }
 158
 159 // Multiply an m*n matrix with an n*p matrix results in an m*p matrix.
160 // Usage: tx_cuda_matmul [ m [ n [ p ] ] ]
161 // m, n, and p default to 1, and are multiplied by BLOCK_SIZE
162 int main(int argc, char **argv)
163 {
164
      // cudaSetDevice(0);
 165
        const int m = BLOCK_SIZE * (argc > 1 ? atoi(argv[1]) : 1);
 166
        const int n = BLOCK SIZE * (argc > 2 ? atoi(argv[2]) : 1);
 167
        const int p = BLOCK_SIZE * (argc > 3 ? atoi(argv[3]) : 1);
 168
        Matrix A = cons Matrix(m, n);
 169
        Matrix B = cons_Matrix(n, p);
 170
        Matrix C = cons_Matrix(m, p);
        MatMul(A, B, C);
 171
 172
        print_Matrix(A, "A");
        print_Matrix(B, "B");
 173
174
        print_Matrix(C, "C");
 175
        return 0;
176 }
 177
 178
       * Update log
 179
       30
 180
       * Feb 25 2015 NYP: Removed forceinline , it is making cli too fast
```

#### Set Breakpoints in CUDA Kernel Code Before Launch

Set breakpoints in the CUDA or OpenMP TARGET region code before you start the process.

Hollow breakpoint indicates a breakpoint will be set when the code is loaded onto the GPU.

#### Start Page × tx\_cuda\_matmul.cu ×

89

95

97

98

100

101

104

105

109

```
___global___void MatMulKernel(Matrix A, Matrix B, Matrix C)
 // Matrix multiplication kernel called by MatrixMul()
  // Block row and column
  int blockRow = blockIdx.y;
  int blockCol = blockIdx.x;
   // Each thread block computes one sub-matrix Csub of C
  Matrix Csub = GetSubMatrix(C, blockRow, blockCol);
  // Each thread computes one element of Csub
  // by accumulating results into Cvalue
  float Cvalue = 0;
  // Thread row and column within Csub
  int row = threadIdx.y;
  int
      col = threadIdx.x;
   // Loop over all the sub-matrices of A and B that are
  // required to compute Csub
  // Multiply each pair of sub-matrices together
  // and accumulate the results
   for (int m = 0; m < (A.width / BLOCK_SIZE); ++m) {</pre>
     // Get sub-matrix Asub of A
     Matrix Asub = GetSubMatrix(A, blockRow, m);
     // Get sub-matrix Bsub of B
     Matrix Bsub = GetSubMatrix(B, m, blockCol);
     // Shared memory used to store Asub and Bsub respectively
     __shared__ float As[BLOCK_SIZE][BLOCK_SIZE];
     ____shared__ float Bs[BLOCK_SIZE][BLOCK_SIZE];
     // Load Asub and Bsub from device memory to shared memory
     // Each thread loads one element of each sub-matrix
     As[row][col] = GetElement(Asub, row, col);
     Bs[row][col] = GetElement(Bsub, row, col);
     // Synchronize to make sure the sub-matrices are loaded
     // before starting the computation
     ___syncthreads();
     // Multiply Asub and Bsub together
     for (int e = 0; e < BLOCK_SIZE; ++e)
       Cvalue += As[row][e] * Bs[e][col];
     // Synchronize to make sure that the preceding
     // computation is done before loading two new
     // sub-matrices of A and B in the next iteration
     ____syncthreads();
   // Write Csub to device memory
```

## Stopped at a Breakpoint in CUDA Kernel Code

 Bold line numbers indicate source code lines where the compiler generated code, which are good places to set breakpoints



### CUDA thread IDs and Coordinate Spaces

rocesses & Thr × .ookup File or	Fun ×	locum.	. × Op
Description	#P	#T 🔻	Members
tx_cuda_matmul (S3)	1	1	p1
🐨 📕 Breakpoint	1	1	p11
WatMulKernel	1	1	p11
🔻 tx_cuda_matmul.c	1	1	p1-1
11	1	1 (	p11
💌 📕 Stopped	1	3	p1.1-3
poll_nocancel	1	1	p1.3
accept4	1	1	p1.2
cuVDPAUCtxCreate	1	1	p1.1
🔍 <unknown line=""></unknown>	1	1	p11
1.1	1	1	p1.1

Host thread IDs have a positive thread ID (p1.1)

CUDA thread IDs have a negative thread ID (p1.-1)

#### GPU Physical and Logical Focus Toolbars



Logical toolbar displays the Block and Thread coordinates.

**Physical** toolbar displays the Device number, Streaming Multiprocessor, Warp and Lane.

To view a CUDA host thread, select a thread with a positive thread ID in the Process and Threads view.

To view a CUDA GPU thread, select a thread with a negative thread ID, then use the GPU focus controls in the logical or physical toolbar to focus on a specific GPU thread or lane.

## Displaying CUDA Program Variables

Action	Points × Data Viev	Command Line	Input/Output	@local type qualifier indicates that variable	e A
Name		Туре	Thread ID	Value is in local storage.	
▼ A		Matrix @local	11	(Matrix @local)	
	width	int	11	0x0000002 (2)	
	height	int	11	0x0000002 (2)	
	stride	int	11	0x0000002 (2)	
W	elements	float @generic *	11	0x7f724e800000 -> 0	
	*(elements)	@generic float	11	0	
[A	dd New Expression]				
			"eler to a stora	nents" is a pointer float in @generic ge.	

- The identifier @local is a TotalView built-in type storage qualifier that tells the debugger the storage kind of "A" is local storage.
- The debugger uses the storage qualifier to determine how to locate A in device memory

## Stepping GPU Code

- Single-step operations advance all the GPU hardware lanes in the same warp
- Note that stepping operations Step and Next are slow in GPU code; the following is faster...
- 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 Status View**

Displays the state of all the GPUs being debugged.

Fully configurable to allow aggregating, sorting and filtering based on physical or logical attributes.

<pre>122 Cvalue += As[row][e] * Bs[e][col]; 123 // Synchronize to make sure that the preceding 124 // computation is done before loading two new 125 // sub-matrices of A and B in the next iteration 126syncthreads(); 127 } 128 // Write Csub to device memory 129 // Each thread writes one element</pre>	
Data View × Command Line × GPU Status × Input/Output × Logger ×	
Vanables Status	
Device(0)	20
▼ SM(0)	No
▼ Warp(0)	
T ▼ Lane(Slane) for lane in {0 1 2 3}	
Function = MatMulKernel	
State = stopped	
nel	
nel	V
Frame: MatMulKernel File: /home/dstewart/cuda/tx_cuda_matmul.cu Line: 115	

## Enabling CUDA Memory Checker Feature





#### From the Program Session Dialog

From the Debug Menu

#### Demo

■ @TEMP@CUDA@.b.out - Rank 1, Thread 12 (<<<(0	0,0,0),(0,0,0)>>>) (Breakpoint) - TotalView 2022			X E
<u>File Edit Group Process Thread Action Points</u>	Debug Window Help			
(Group (Control)	] [] ÞΞ // [] 🏟 (‡ ReplayEngine 💿 ┥ 🖆 [] 🗄 🗄 Ξ< ト/ 🗏 🕹 (‡ Memory 💿 📚 🌢			
GPU (Logical) Block 0 + 0 + 0 + Thread 0				
Processes & Th * Lookup File or Fu * Docu *	Start Page * _dl_debug_state * vecAdd.cxx * vecAddWrapperCXX.cu *	Call Stack ×		
ⓐ ② ③ 🗧 〓	1 #include <stdio.h></stdio.h>	i ÷		
Description #P #T ¥ M	2 #include <stdlib.h> 3 #include <math.h></math.h></stdlib.h>	C vecAdd		
▼ srun (S3) 1 1 p	4 #include <assert.h></assert.h>	VecAdd		
	6 // CUDA kernel. Each thread takes care of one element of c			
	7global Void VecAdd(Tloat *a, float *D, float *c, int n) 8 {			
Running 1 1 p	9 // Get our global thread ID 10 int id = blockIdx x*blockDim_x+threadIdx x:	🧃 Info		
<pre>vunknown ad 1 4 p</pre>		Function	vecAdd	
1.1 1 1 p	12 // Make sure we do not go out of bounds 13 if (id < n)	Source	u1/j/jdelsign/sup	port-50919/vecAddWrapperCXX.cu
1.2 1 1 p	14 c[id] = a[id] + b[id]; 15 }	Local Variables		
1.3 1 1 p	16 17 //void vecAdd wrapper()	Name	Туре	Value
1.4 1 1 р	18 void vecAdd_wrapper(int rank, int nprocs)	<ul> <li>Arguments</li> </ul>		
▼ b.out (S4) 4 4 0	<pre>char *cvd = getenv ("CUDA_VISIBLE_DEVICES"); char *cvd = getenv ("CUDA_VISIBLE_DEVICES");</pre>	▶ a	float @generic	0x15318b200000 -> 0
	<pre>21 printf ("kank %o sees CODA_VISIBLE_DEVICES=%s\n", 22 rank,</pre>	▶ b	float @generic	0x15318b261c00 -> 0.382683
🗢 Configure	<pre>23 cvd ? cvd : "(unset)"); 24 char *cdo = getenv ("CUDA_DEVICE_ORDER");</pre>		float @generic	0v15318b2c3800 > 0
Select process or thread attributes to group by:	25 printf ("Rank %d sees CUDA_DEVICE_ORDER=%s\n", 26 rank		ioar @generic	0x155100203000 => 0
Control Group	27 cdo ? cdo : "(unset)");	n	int @parameter	0x000186a0 (100000)
Share Group	<pre>29 int deviceCount = 0;</pre>	Block at Lin		
✓ Hostname	30 cudaGetDeviceCount(&deviceCount); 31	id	int @register	<bad (optimized="" address:="" out)=""></bad>
V Process State	32 printf("Rank %d out of %d processes: I see %d GPU(s).\n",	J		
↑ <b>つ</b> ↓	Data View * Command Line * Logger * GPU Status *			
	Control Group - 989 C			
Action Points ¥ Replay Bookmarks ×				
ID Type Stop Location	Variables Status			
✓ 1 Break ProcessperCXX.cu#10	Processes     Process(r1)			
1 Break ProcessperCXX.cu#10	Device(0)			
	Device(3)			
	SM(\$sm) for sm in [0 26]:			
	Warp(\$warp) for warp in [031]:			
	▼ Lane(\$lane) for lane in [031]:			
	State = breakpoint			
Pank: 1 (3351@128.55.6/ 105-mmet) @TEMD@CUDA@ h	out Thread: 1.2 (zzz/0.0.0) (0.0.0)>>>) Breaknoint Erame: vacAdd Eile: al\u1/i/i/dalcian/cunnad:50010\u004dd/bleanord/C	You Line: 10		
Talik. 1 (0001@120.00.04.100.00100) @TEMP@CODA@.D.	inclut. 1.2 (***(0,0,0)/(0,0,0)/2/) - breakpoint France. veckuu Frieavdrijueisigiiisuppoir-0.0919/veckuu/wiapperck	Line. 10		

Debugging on Perlmutter

#### Debugging on Perlmutter (Things to Know)

- If you bind processes to GPUs using srun, the debugger cannot determine which GPUs the processes are using
  - SLURM's use of Linux control groups make it impossible
  - Workaround Do not use the "--gpu-bind" option when debugging
- Watchpoints in GPU memory are not support on NVIDIA GPUs, but CPU watchpoints are supported
- On Perlmutter, the environment variable "TVD\_DISABLE\_CRAY=1" must be set to disable using Cray CTI
  - "module load totalview" sets TVD\_DISABLE\_CRAY=1 on Perlmutter
  - SSH is used to instantiate the TV/MRNet tree
  - Requires passwordless SSH between nodes

### Debugging on Perlmutter (Things to Know)

- Using SSH to between NERSC nodes can generate a lot of terminal output
  - Each SSH generates a long "NOTICE TO USERS" message
- The messages can be suppressed by adding the following lines to your "\$HOST/.ssh/config" file:

# The "LogLevel quiet" option stops the "NOTICE TO USERS" messages
Host \*
LogLevel quiet

• The above is <u>not</u> necessary, but it does reduce terminal output

## Debugging on Perlmutter (Supported Start-ups)

- TotalView supports <u>interactive</u> and <u>batch</u> debugging sessions
- <u>Interactive</u> debugging sessions
  - Use **salloc** to allocate interactive nodes
  - Start TotalView on **srun** within the allocation
  - Allows restarting srun multiple times within the same allocation
- <u>Batch</u> debugging sessions
  - Use **sbatch** to submit a batch job
  - Batch script uses **tvconnect srun** ... to request a "reverse connect" to TotalView
  - Start TotalView on a login node and accept the "reverse connect" request
  - To restart srun multiple times, invoke **tvconnect srun** in a loop in the script

### Debugging on Perlmutter (Interactive Start-up)

- Load the "totalview" module
  - module load totalview
- Allocate some nodes, for example
  - salloc -A ntrain7 -C gpu -N 2 -G 8 -t 60 -q interactive\_ss11
- An interactive shell (bash, csh, etc.) will start inside the allocation
- Start **totalview** on **srun**, for example
  - totalview -args srun -n 8 -G 8 -c 32 --cpu-bind=cores ./b.out
    - Remember, "--gpu-bind" does not work, so do not use it while debugging

## Debugging on Perlmutter (Batch Start-up)

• Example batch script using **tvconnect** 

#!/bin/bash -x
#SBATCH -A nvendor
#SBATCH -C gpu
#SBATCH -N 2
#SBATCH -G 8
#SBATCH -G 8
#SBATCH -t 30
#SBATCH --qos=debug

```
module load totalview
tvconnect srun b.out
```

- When the batch script starts, **tvconnect** blocks until a **totalview** accepts the reverse connect request
- On the login node, load the "totalview" module and start totalview

```
module load totalview
totalview
```

### Debugging on Perlmutter (Batch Start-up)

- TotalView will "Listen For Reverse Connections" by default, but make sure the option is enabled
- When the batch script executes the **tvconnect** command, TotalView will post a dialog
- Select "Yes" to connect TotalView to the batch job

	Debug Window Help	
3	🖸 🖂 🖊 🚺 🗱 🛛 ReplayEngine 💿 ┥ 🖆 🖆 🛃 🛤 🗮	
	Start Page ×	
	What do you want to do today? Recent	Sessio
	Debug a Program Debug a Parallel Program	
	No Rece	ent Sessi
2	Attach To Process Load Core or Replay Recording File	
	Listen For Reverse Question	<u>ן</u>
	The following reverse connection request was found.	port
	What's New Do you want to accept?	atact us at io/suppo
	New in TotalView 2 Dive Stacks for Data Del	
H	Data View × Command Line × Logger ×	
	$  \bullet \rightarrow$	

## Debugging on Perlmutter (Common to Interactive/Batch)

- Once TotalView starts-up on srun, the following steps are common to interactive / batch debugging
- Typically
  - Select "Go" to start srun
  - srun will launch the parallel program
  - TotalView detects the parallel program launch and attaches to the MPI processes
- When the jobs goes parallel,

TotalView will post a dialog



# Stop the job when it goes parallel?

- Click "Yes" to stop the parallel job, which is useful if you want to
  - Navigate to source files / functions
  - Set breakpoints
- Click "No" to allow the job to run, which is useful if you
  - Have saved breakpoints from a previous session
  - Know the program is going to crash (SEGV, etc.)



## TotalView will focus on main() in rank 0

	Start	Page ¥ _dl_debug_state ¥ vecAdd.cxx ¥
	1	<pre>#include <stdio.h></stdio.h></pre>
	2	#include <mpi.h></mpi.h>
	3	#include <unistd.h></unistd.h>
	4	#include <stdlib.h></stdlib.h>
	5	
	6	//void vecAdd_wrapper(void);
	7	<pre>void vecAdd_wrapper(int, int);</pre>
	8	
	9	volatile int spinner = 0;
	10	int main( int argc, char^ argv[] )
	12	int rank pproces
	12	MPT Toit(&argc_&argy):
	14	MPT_Comm_rank(MPT_COMM_WORLD&rank):
	15	MPI Comm size(MPI COMM WORLD, &nprocs):
	16	vecAdd wrapper(rank, nprocs);
	17	if $(argc > 1)$
	18	<pre>spinner = atoi (argv[1]);</pre>
	19	if (spinner > 0)
	20	{
	21	<pre>printf ("Sleeping %d seconds\n", spinner);</pre>
	22	fflush (⊕);
	23	while (spinner)
$\square$	24	<pre>sleep(1);</pre>
	25	} MDT_Finalize():
	20	MLT_LTHUTTSC();
	28	return 0:
	29	}
	30	

#### Navigate to a file or function you want to debug

	4	<pre>#include <stdlib.h></stdlib.h></pre>
P	5	
	6	//void vecAdd_wrapper(void);
P	7	<pre>void vecAdd_wrapper(int, int);</pre>
	8	
Р	9	<pre>volatile int spinner = 0;</pre>
	10	<pre>int main( int argc, char* argv[] )</pre>
м	11	{
	12	int rank, nprocs;
1	13	<pre>MPI_Init(&amp;argc, &amp;argv);</pre>
p	14	<pre>MPI_Comm_rank(MPI_COMM_WORLD, &amp;rank);</pre>
-	15	MPI_Comm_size(MPI_COMM_WORLD, &nprocs);
P	16	vecAdd_wrapper(rank. nprocs):
-	17	if (argc > 1 Navigate to File or Function
P	18	spinner = Add to Data View
	19	11 (spinner Add to New Data View
P	20	{ Add to New Data View
	21	printr ("Sleeping %d seconds\n", spinner);
P	22	ttiush (0);
	23	while (spinner)
	24	sieep(1);
=	20	/ MPT_Finalize():
	20	mrt_rinatize(),

# Find the CUDA kernel and select a line number to plant a breakpoint

- Line numbers indicate if there's code at that line
  - Pale line numbers indicate no code (yet)
  - Bold line numbers indicate code
- CUDA code is *dynamically* loaded at runtime, so TotalView does not have any debug information *until* the CUDA kernel is launched
- Select a line number in the CUDA kernel that will have CUDA code loaded
  - Hollow breakpoint markers indicate no code yet
  - Solid breakpoint markers indicate code
- Source line information for a source file is *unified* for both GPU and CPU code

	Chart	
	Start F	rage a _di_debug_state a vecAdd.cxx a vecAddwrapperCXX.cu a
	4	#include <assert.h></assert.h>
	5	
	6	// CUDA kernel. Each thread takes care of one element of c
	7	global void vecAdd(float *a, float *b, float *c, int n)
I	8	{
I	9	// Get our global thread ID
U	10	<pre>int id = blockIdx.x*blockDim.x+threadIdx.x;</pre>
	11	
1	12	// Make sure we do not go out of bounds
I	13	if $(id < n)$
I	14	c[id] = a[id] + b[id];
l	15	}
+	16	
	17	//void vecAdd_wrapper()
	18	void vecAdd_wrapper(int rank, int nprocs)
	19	
	20	char ^cvd = getenv ("CUDA_VISIBLE_DEVICES");
	21	printr ("Rank %d sees CODA_VISIBLE_DEVICES=%s\n",
	22	rank,
	23	char todo = geteny ("CUDA DEVICE OPDEP");
	24	printf ("Park %d sees CUDA DEVICE_ORDER");
	20	rank

### Click the "Go" button to run the application and launch the kernel



Stonned	at a brand main	A & h aut Bank () Thread () A / (() () () () () () () () () () () () ()	an aid001691)			
JCOPPCG		A(@.D.OUT - Rank 0, Thread 04 (<<<(0,0,0),(0,0,0)>>>) (Breakpoint) - Totatview 2024.1 (	5h HIGOOT081) ×			
Elle	Eait Group Process Inread Action Points Debug Window					
G	roup (Control) :					
Proc	cesses & Threads X Lookup File or Function X Documents X	Start Page × _dl_debug_state × vecAdd.cxx × vecAddWrapperCXX.cu ×	Call Stack ×			
0	2 3 = =	1 #include <stdio.h></stdio.h>	=			
Desc	cription # P # T • Members	3 #include <math.h></math.h>	G++ vecAdd			
•	srun (S3) 1 1 p1	4 #include <assert.n></assert.n>				
-	b.out (S4) 8 8 0-7	<pre>6 // CUDA kernel. Each thread takes care of one element of c 7global void vecAdd(float *a, float *b, float *c, int n)</pre>				
	Breakpoint 8 8 0-74	8 { 9 // Get our global thread ID				
	2.4 1 1 0.4	<pre>int id = blockIdx.x*blockDim.x+threadIdx.x;</pre>				
	2.4	12 // Make sure we do not go out of bounds				
	34 1 1 14	13 17 (1d < n) 14 c[id] = a[id] + b[id];				
	44 1 1 24	15 } 16				
	54 1 1 34	17 //void vecAdd_wrapper()				
	64 1 1 44	19 {				
	74 1 1 54	<pre>20 char *cvd = getenv ("CUDA_VISIBLE_DEVICES"); 21 printf ("Rank %d sees CUDA_VISIBLE_DEVICES=%s\n",</pre>				
	84 1 1 64	22 rank, 23 cvd 2 cvd : "(upset)"):	🕜 Info			
0	Configure	24 char *cdo = getenv ("CUDA_DEVICE_ORDER"); 25 printf ("Rank %d sees CUDA_DEVICE_ORDER");	Function vecAdd			
Selec	ct process or thread attributes to group by:	26 rank,	Sourcees/j/jdelsign/support-50919/vecAddWrapperCXX.cu			
	Stop Reason	27 cdo ? cdo : "(unset)"); 28	Line 10			
	Process ID	<pre>29 int deviceCount = 0; 30 cudaGetDeviceCount(&amp;deviceCount);</pre>	FP 0x fffdc0			
~	Thread ID	31				
	Process Held	Data View & Command Line & Longer & CPUI Status & Array View *	1			
	Thread Held	Control Group View Command Line C Logger - GPO Status - Anay View -	1			
	t ว ↓	Variables Status	Local Variables - X			
		V Processes	Name Type Value			
Actio	on Points 🗱 Replay Bookmarks 🕷	Process(\$dpid) for dpid in [r0 r7]:	Anuments			
	ID Type Stop Location L	Device(\$dev) for dev in {0.1.2}	* Arguments			
	1 Break Process/vecAddWrapperCXX.cu#10pperC	<ul> <li>SM(\$sm) for sm in [026]:</li> </ul>	a float @generic 0x7f1eb9600000 -> 0			
	1 Break Process/vecAddWrapperCXX.cu#10pperC	Warp(\$warp) for warp in [031]:	b float @generic 0x7f1eb9661c00 -> 0			
		Lane(\$lane) for lane in [0 31]:	▶ c float @generic 0x7f1eb96c3800 -> 0			
		Function = vecAdd	n @parameter int 0x000186a0 (100000)			
		State = breakpoint	<ul> <li>Block at Line 10</li> </ul>			
		✓ Warp(\$warp) for warp in [0 23]:	id @register int <bad (optimize<="" address:="" td=""></bad>			
		Lane(\$lane) for lane in [031]:				
		Function = vecAdd				

### Source view stopped in a CUDA kernel

- Line number information for the GPU code is *unified* with the CPU code
- The hollow breakpoint marker turns solid, indicating that there is now code at that line
- The PC arrow and highlighted source line indicates where the warp is stopped



### GPU thread focus and navigation controls

• "GPU (Logical)" control displays and allows focusing on a specific Block and Thread



• "GPU (Physical)" control displays and allows focusing on a specific Device, SM, Warp, and Lane



## CUDA stack backtrace and local variables

Call Stack

• Open the drawer for details

• Local Variables

Ca	all Stack 🗶		
=			
C	++ vecAdd		
0	Info		
	Function	vecAdd	
	Source	u1/j/jdelsign/sup	port-50919/vecAddWrapperCXX.cu
10	cal Variables	×	
Na	cal Variables me	Х	Value
Na Va	cal Variables me Arguments	× Туре	Value
Na Va	me Arguments	X Type float @generic	Value 0x15318b200000 -> 0
Na	Arguments a b b	X Type float @generic float @generic	Value 0x15318b200000 -> 0 0x15318b261c00 -> 0.382683
Na	Arguments a b c c	X Type float @generic float @generic float @generic	Value 0x15318b200000 -> 0 0x15318b261c00 -> 0.382683 0x15318b2c3800 -> 0
Na	Arguments a b c n	X Type Type float @generic float @generic float @generic int @parameter	Value 0x15318b200000 -> 0 0x15318b261c00 -> 0.382683 0x15318b2c3800 -> 0 0x000186a0 (100000)
Na	Arguments Arguments b c n Block at Lin	X Type Type float @generic float @generic float @generic int @parameter	Value 0x15318b200000 -> 0 0x15318b261c00 -> 0.382683 0x15318b2c3800 -> 0 0x000186a0 (100000)

#### **GPU** Status view

- The "GPU Status" view displays an aggregated overview of one or more of the GPUs in the whole job, in a single process, or on a single GPU
- The "GPU Status" view controls allow
  - Selecting the set of properties to display
  - Aggregation by the selected properties
  - Sorting by the selected properties
  - Creating compound filters to include/exclude properties that are equal, not equal, greater, etc.
- Allows you to get a "big picture" of the state of one or more of the GPUs in your job

	Control Group 👻 🚺	
	Variables	Status
	V Processes	
0	Process(r1)	
0	Device(0)	
-	Device(3)	
	SM(\$sm)	for sm in [0 26]:
	Warp(\$warp)	for warp in [0 31]:
	Lane(\$lane)	for lane in [0 31]:
		Function = vecAdd
		State = breakpoint

@TEMP@CUDA@.b.out - Rank 0, Thread 04 (<<<(0,0,0),(0,0,0)>>>) (Breakpoint) - TotalView 2024.1 (on nid001681) ×		
File Edit Group Process Thread Action Points Debug Window Help		
[ Group (Control) : ▶    ■  ▶ 같 ½ ½ / 🔅 İ ReplayEngine ◎ ◀ 앱 🕑 갈 좌 ▶ 🕸 🌡 Memory ◎ 📚 ♦ 🖄 🏟		
Processes & Threads       Lookup File or Function       Documents         Image: Constraint of the state of the	<pre>Start Page # _dl_debug_state # vecAdd.cxx # vecAddWrapperCXX.cu #      #include <stdio.h>     #include <stdib.h>     #include <stdib.h>     #include <stdib.h>     #include <stdib.h>     #include <aster.h>     // CUDA kernel. Each thread takes care of one element of c     _global void vecAdd(float *a, float *b, float *c, int n)     {         // Get our global thread ID         int id = blockIdx.x*blockDim.x+threadIdx.x;         // Ake sure we do not go out of bounds         if (id &lt; n)         c[id] = a[id] + b[id];     }     //void vecAdd_wrapper()     rovid vecAdd_wrapper()     rovid vecAdd_wrapper(int rank, int nprocs) </aster.h></stdib.h></stdib.h></stdib.h></stdib.h></stdio.h></pre>	Call Stack ×
6.4       1       1       4.4         7.4       1       1       5.4         8.4       1       1       6.4         Configure         Select process or thread attributes to group by:         Stop Reason	10       for "cvd = getenv ("CUDA_VISIBLE_DEVICES");         10       {         10       {         10       {         10       {         10       {         10       {         10       {         10       {         10       {         10       {         11       hit holds)         12       printf ("Rank %d sees CUDA_VISIBLE_DEVICES");         12       rank,         13       cdd ? cvd : "(unset)");         14       char *cdo = getenv ("CUDA_DEVICE_ORDER=%s\n",         15       printf ("Rank %d sees CUDA_DEVICE_ORDER=%s\n",         16       rank,         17       cdo ? cdo : "(unset)");         18       cdo ? cdo : "(unset)");         19       int deviceCount = 0;         10       cudaGetDeviceCount(&deviceCount);         13       Control Group *         110       C	Into         Function       vecAdd         Source      es/j/jdelsign/support-50919/vecAddWrapperCXX.cu         Line       10         FP       0x fffdc0
	Variables Status Variables	Local Variables ×
Action Points       X       Replay Bookmarks       X         ID       Type       Stop       Location       L         I       Break       Process      /vecAddWrapperCXX.cu#10      pperC         I       Break       Process      /vecAddWrapperCXX.cu#10      pperC	▼ Process(\$dpid)         for dpid in [r0r7]:           Device(\$dev)         for dev in {0 1 2}           ▼ Device(3)            ▼ SM(\$sm)         for sm in [026]:           ▼ Warp(\$warp)         for warp in [031]:           ▼ Lane(\$lane)         for lane in [031]:           ▼ SM(27)            ▼ Warp(\$warp)         for warp in [023]:           ▼ Lane(\$lane)         for lane in [031]:	Arguments     ox7f1eb9600000 -> 0       b     float @generic     0x7f1eb9661c00 -> 0       c     float @generic     0x7f1eb9661c00 -> 0       n     @parameter int     0x000186a0 (100000)       Block at Line 10     id     @register int <bad (optimize<="" address:="" td=""></bad>
	Function = vecAdd	ne V au Line: 10
Batch Debugging with TVScript

### tvscript

- A straightforward language for unattended and/or batch debugging with TotalView and/or MemoryScape
- Usable whenever jobs need to be submitted or batched
- Can be used for automation
- A more powerful version of printf, no recompilation necessary between runs
- Schedule automated debug runs with *cron* jobs
- Expand its capabilities using TCL

### tvscript

tvscript [options] [ filename ] [ -a program\_args]

#### options

TotalView and tvscript command-line options.

#### filename

The program being debugged.

-a program\_args

Program arguments.

## tvscript

• All of the following information is provided by default for each print

- Process id
- Thread id
- Rank
- Timestamp
- Event/Action description
- A single output file is written containing all of the information regardless of the number of processes/threads being debugged

### Supported tvscript events

Event Type	Event	Definition
General event	any_event	A generated event occurred.
/lemory debug- jing event	addr_not_at_start	Program attempted to free a block using an incorrect address.
	alloc_not_in_heap	The memory allocator returned a block not in the heap; the heap may be corrupt.
	alloc_null	An allocation either failed or returned NULL; this usu- ally means that the system is out of memory.
	alloc_returned_bad_alignment	The memory allocator returned a misaligned block; the heap may be corrupt.
	any_memory_event	A memory event occurred.
	bad_alignment_argument	Program supplied an invalid alignment argument to the heap manager.
	double_alloc	The memory allocator returned a block currently being used; the heap may be corrupt.
	double_dealloc	Program attempted to free an already freed block.
	free_not_allocated	Program attempted to free an address that is not in the heap.
	guard_corruption	Program overwrote the guard areas around a block.

# Supported tvscript events

Event Type	Event	Definition
	hoard_low_memory_threshold	Hoard low memory threshold crossed.
	realloc_not_allocated	Program attempted to reallocate an address that is not in the heap.
	rz_overrun	Program attempted to access memory beyond the end of an allocated block.
	rz_underrun	Program attempted to access memory before the start of an allocated block.
	rz_use_after_free	Program attempted to access a block of memory after it has been deallocated.
	rz_use_after_free_overrun	Program attempted to access memory beyond the end of a deallocated block.
	rz_use_after_free_underrun	Program attempted to access memory before the start of a deallocated block.
	termination_notification	The target is terminating.
Source code debugging event	actionpoint	A thread hit an action point.
	error	An error occurred.
Reverse debugging	stopped_at_end	The program is stopped at the end of execution and is about to exit.

# Supported tvscript actions

Action Type	Action	Definition
Memory debug- ging actions	check_guard_blocks	Checks all guard blocks and write violations into the log file.
	list_allocations	Writes a list of all memory allocations into the log file.
	list_leaks	Writes a list of all memory leaks into the log file.
	save_html_heap_status_source_view	Generates and saves an HTML version of the Heap Status Source View Report.
	save_memory_debugging_file	Generates and saves a memory debugging file.
	save_text_heap_status_source_view	Generates and saves a text version of the Heap Status Source View Report.
Source code debugging actions	display_backtrace [-levellevel-num] [ num_levels ]	Writes the current stack backtrace into the log file.
	[ options]	<b>-level</b> <i>level-num</i> sets the level at which infor- mation starts being logged.
		<i>num_levels</i> restricts output to this number of levels in the call stack.
		If you do not set a level, <b>tvscript</b> displays all levels in the call stack.
		<i>options</i> is one or more of the following: -[no]show_arguments
		-[no]show_tp -[no]show_fp_registers
		-[no]show_image
		-[no]show_locals
		-[no]show_pc
		-[no]show_registers

# Supported tvscript actions

Action Type	Action	Definition
	<pre>print [ -slice {slice_exp}] {variable   exp}</pre>	Writes the value of a variable or an expression into the log file. If the variable is an array, the <b>-</b> <b>slice</b> option limits the amount of data defined by slice_exp. A slice expression is a way to define the slice, such as <b>var[100:130]</b> in C and C++. (This displays all values from <b>var[100]</b> to <b>var[130]</b> .) To display every fourth value, add an additional argument; for exam- ple, <b>var[100:130:4]</b> . For additional information, see <i>"Examining Arrays"</i> in the <i>TotalView for HPC User Guide</i> .
Reverse debug- ging actions	enable_reverse_debugging	Turns on ReplayEngine reverse debugging and begins recording the execution of the program.
	save_replay_recording_file	Saves a ReplayEngine recording file. The file- name is of the form <processname>- <pid>_<date>.<index>.recording.</index></date></pid></processname>

## tvscript examples

#### Simple example

#### MPI example

```
tvscript -mpi "Open MPI" -tasks 4 \
-create_actionpoint \
"hello.c#14=>display_backtrace" \
~/tests/MPI_hello
```

### tvscript examples

#### Memory Debugging example

```
tvscript -maxruntime "00:00:30" \
```

-event\_action "any\_event=save\_memory\_debugging\_file" \
-guard\_blocks -hoard\_freed\_memory -detect\_leaks \

```
~/work/filterapp -a 20
```

ReplayEngine example tvscript \

-create\_actionpoint "main=>enable\_reverse\_debugging" \
-event\_action "stopped\_at\_end=>save\_replay\_recording\_file" \
filterapp



• TVScript demo (tvscript -script\_file file tvscript\_example.tvd ex2)

Common TotalView Usage Hints

### Common TotalView Usage Hints

- TotalView can't find the program source
  - Did you compile with -g?
  - How to adjust the TotalView search paths? Preferences -> Search Path
- Python Debugging
  - Making sure proper system debug packages are installed for Python
- X11 forwarding performance
  - If users are forwarding X11 displays through ssh TotalView UI performance can be bad
- Understanding different ways to stop program execution with TotalView Action Points
  - Using a watchpoint on a local variable
- Focus
  - Diving on a variable that is no longer in scope. Check the Local Variables window for in scope variables
  - TotalView doesn't change focus to the thread hitting a breakpoint. Set Action Point Preferences to "Automatically focus on threads/processes at breakpoint"

### Common TotalView Usage Hints (cont.)

- MPI Debugging
  - Differences in launching MPI job from within the TotalView UI vs the command line.
  - TotalView runs an MPI program without stopping. Set the Parallel Preferences to "Ask What To Do" in After Attach Behavior
  - Using wrong attributes in processes and threads view
- Reverse Debugging
  - Running out of memory by not setting the maximum memory allocated to ReplayEngine
  - Defer turning on reverse debugging until later in program execution to avoid slow initialization phases
  - Adjust reverse debugging circular buffer size to reduce resources
- Memory Debugging
  - Starting with All memory debugging options enabled rather than Low
  - Not setting a size restriction for Red Zones
  - Issues with getting memory debugging turned on in an MPI job. May have to set LD\_PRELOAD environment variable or worst case, prelink HIA

## Common TotalView Usage Hints (cont.)

- Differences in functionality between new UI and classic UI
  - How to switch between them. Preferences -> Display or totalview –newUI and totalview -oldUI
  - Where the gaps still are in functionality
- Reverse Connect with tvconnect
  - When I use Reverse Connect I get the following obscure message: myProgram is an invalid or incompatible executable file format for the target platform
  - The message indicates an incompatible file format but most often this occurs if the program provided to tvconnect for TotalView to debug cannot be found. The easiest way to resolve problem is to provide the full path to the target application, e.g., tvconnect /home/usr/myProgram
- How do I get help?
  - How to submit a support ticket? techsupport@roguewave.com
  - Where is TV documentation (locally and on the internet). <u>https://help.totalview.io/</u>
  - Are there videos I can watch to learn how to use TotalView? <u>https://totalview.io/support/video-tutorials</u>

## TotalView Resources and Documentation

# TotalView Resources and Documentation

- TotalView website: <u>https://totalview.io</u>
- TotalView documentation:
  - <u>https://help.totalview.io</u>
  - User Guides: Debugging, Memory Debugging and Reverse Debugging
  - Reference Guides: Using the CLI, Transformations, Running TotalView
- Blog:

https://totalview.io/blog

• Video Tutorials:

#### https://totalview.io/support/video-tutorials





## Contact us

• Bill Burns (Senior Director of Software Engineering and Product Manager)

bburns@perforce.com

• John DelSignore (TotalView Chief Architect)

jdelsignore@perforce.com

• Scot Halverson (NVIDIA Solutions Architect)

shalverson@nvidia.com

• Peter Thompson (Senior Support Engineer)

pthompson@perforce.com

• Bruce Ryan (Senior Account Executive)

bryan@perforce.com

• Ken Hill (Senior Sales Engineer)

khill@perforce.com