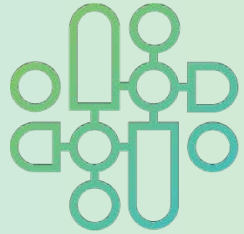


SC23

Denver, CO | i am hpc.

PERFORCE

From Bugs to Breakthroughs: Harnessing HPC Software Debuggers for Success

Bill Burns – Senior Director of Software Engineering and Product Manager, Perforce Software

Agenda

- Introduction to TotalView
- Simultaneously Debug CPU and GPU code
- Debug Hybrid MPI and OpenMP applications
- Advanced Debugger Features for Solving Tough Problems
- Questions and Answers

© Perforce Software, Inc.

Introduction to TotalView

How an HPC Debugger Is Different From a Regular Debugger?

HPC Application Characteristics

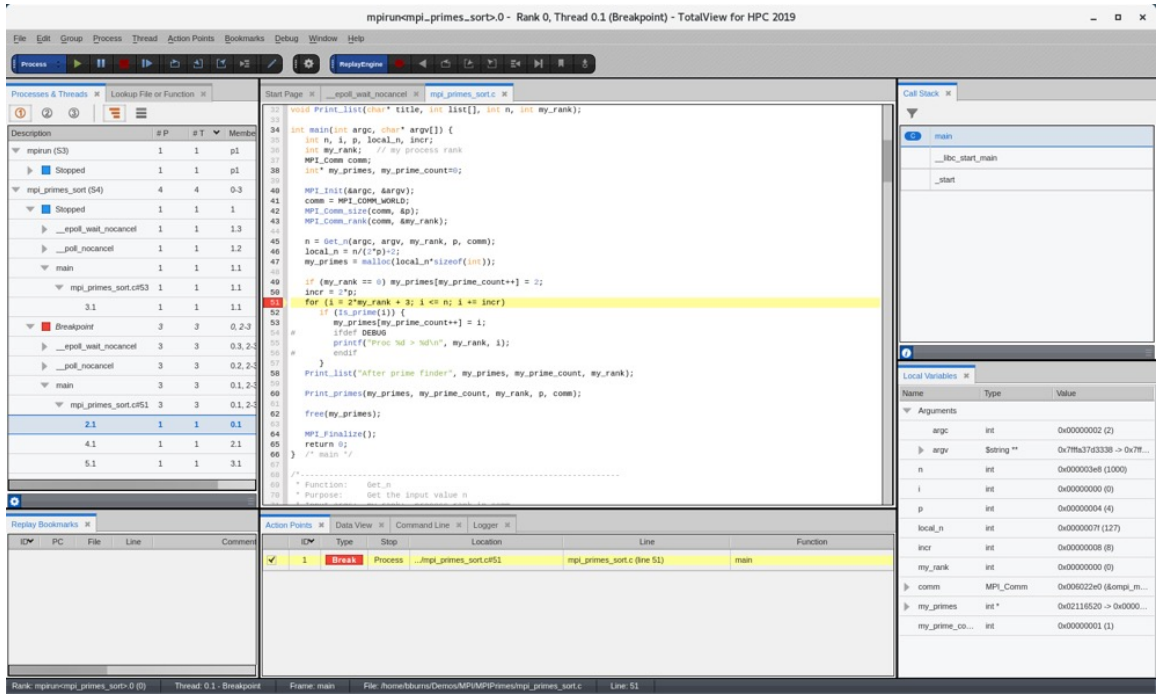
- **Parallel computation**
  - Multiple threads and processes
- **Scale**
  - Hundreds to thousands of threads and processes
  - Large applications with many lines of code
  - Large amounts of data
- **HPC enabling technologies**
  - MPI, OpenMP, and GPUs
- Complex applications built with different compute technologies and languages
- Remote access to supercomputing clusters

HPC Debugger

- “Parallel” is built into debugger operations
- Examine data in groups and down to individual threads
- “Understand” parallel technologies – aid the user in acquiring parallel processes into a debug session
- Provide seamless debugging across technologies
  - Host to GPU and back
  - Across languages
- Ability to handle massive applications
  - Millions of lines of code, gigabytes of debug information
- Ability to easily debug remotely
- Be good at all the ”normal” debugging operations

HPC Debugging with TotalView

- Comprehensive multi-process/thread dynamic analysis and debugging
- Debug hybrid MPI/OpenMP applications
- Advanced C, C++ and Fortran support
- NVIDIA CUDA GPU debugging support
- AMD / ROCm GPU debugging
- Integrated reverse debugging
- Mixed language C/C++ and Python debugging
- Memory debugging and leak detection
- Batch/unattended debugging



LANGUAGES

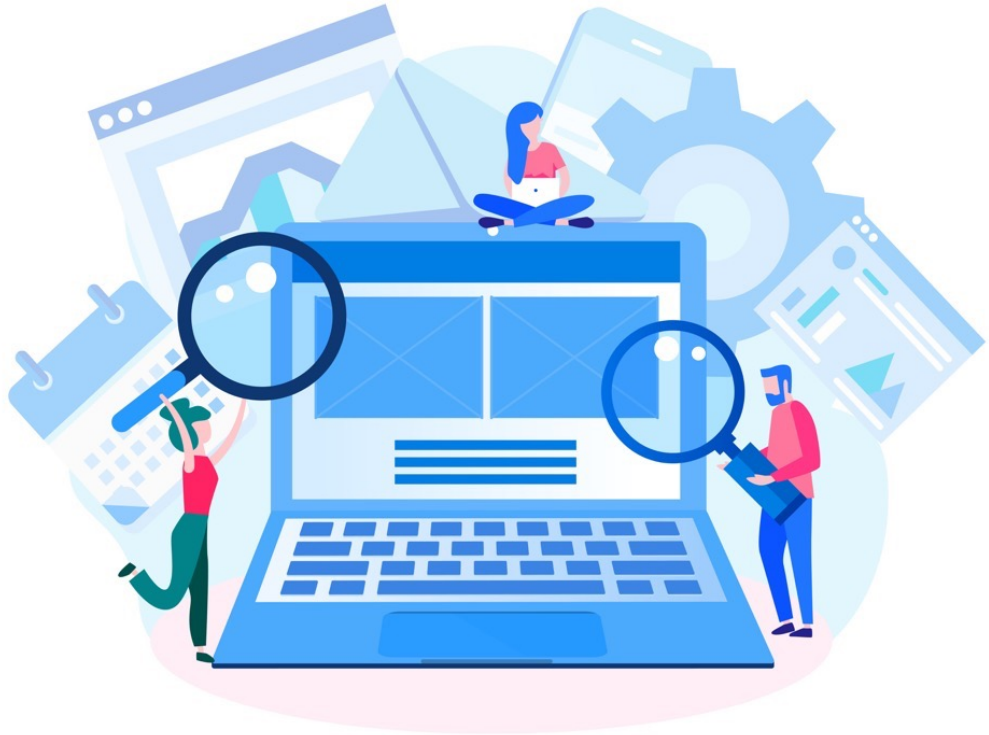
OPERATING SYSTEMS

APPLICATIONS

PLATFORMS

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



Simultaneously Debug CPU and GPU Code

GPU Debugging with TotalView

NVIDIA GPUs

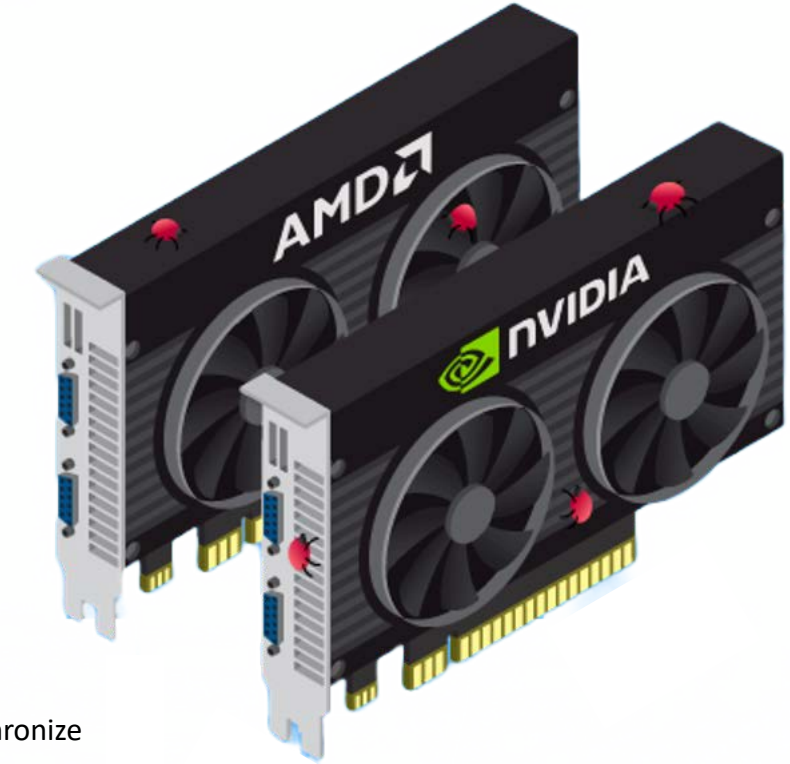
- NVIDIA Tesla, Fermi, Kepler, Pascal, Volta, Turing, Ampere, **Hopper coming soon**
- NVIDIA CUDA 9.2, 10, 11, and 12
  - With support for Unified Memory
- Debugging 64-bit CUDA programs
- Features and capabilities include
  - Support for dynamic parallelism
  - Support for MPI based clusters and multi-card configurations
  - Flexible Display and Navigation on the CUDA device
    - Physical (device, SM, Warp, Lane)
    - Logical (Grid, Block) tuples
  - GPU Status View reveals what is running where
  - Support for types and separate memory address spaces
  - Leverages CUDA memcheck

AMD GPUs

- AMD MI50, MI100, and MI200 series of GPUs
- ROCm 5.4 – 5.7
- Debug HIP (Heterogeneous Interface for Portability) and MPI
- Debugging Features:
  - Process launch, attach, and detach
  - Viewing scalar, vector, general, and special AMD GPU registers
  - Instruction disassembly
  - Breakpoint creation and deletion on AMD GPU code
  - Single-stepping and fast smart-stepping
  - Stack unwinding, including inlined functions
  - Navigation controls for changing the logical workgroup / work-item focus or physical agent, queue, dispatch, wave, and lane focus
  - Variable display with the ROCm 5.1+ compilers
  - Data watchpoints on global memory variables

HIP vs. CUDA

- HIP (Heterogeneous Interface for Portability) is AMD's answer to CUDA that provides a dedicated GPU programming environment
- HIP is very similar to CUDA in that it provides a C++ runtime API and kernel language for running code on GPUs
- HIP is different in that it can create portable applications for both AMD and NVIDIA GPUs
- **Key Features**
  - High performance with little impact over direct CUDA coding
  - Code in a single-source C++ programming language
  - Provides "hipify", a tool to automatically convert source from CUDA to HIP
  - Provides built in functions for accessing specific GPU hardware capabilities
- **Kernel Code**
  - HIP / CUDA look similar and have similar memory qualifiers
- **API differences**
  - Generally, equivalent HIP functions for CUDA, simply replace "cuda" with "hip"
    - cudaMalloc => hipMalloc; cudaMemcpy => hipMemcpy; cudaDeviceSynchronize => hipDeviceSynchronize
  - Launch of kernels differs
    - hipLaunchKernelGGL(kernel_name, gridSize, blockSize, shared_mem_size, stream, arg0, arg1, …)
      kernelName<<<gridSize, blockSize, shared_mem_size, stream>>>(arg0, arg1, …)



HIP HSA Model vs. CUDA

Terminology / Concepts

HIP		CUDA	
grid	A collection of work-groups	grid	A collection of blocks
work-group	An instance of execution on the kernel agent performed by the Compute Unit (CU)	block	Contains a group of threads run on a Streaming Multiprocessor (SM)
wavefront	Groups of work-items organized into up to 64 lanes. Size is defined by its number of lanes.	warp	Groups of threads organized into up to 32 lanes; finest grained execution entity
lane	An element of a wavefront. Finest grained execution entity.	lane	An element of a warp consisting of threads
thread / work-item		thread	

Source View Opened on CPU Host Code

Start Page | tx_cuda_matmul.cu

```
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++)
155     for (int col = 0; col < A.width; col++)
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);
171   MatMul(A, B, C);
172   print_Matrix(A, "A");
173   print_Matrix(B, "B");
174   print_Matrix(C, "C");
175   return 0;
176 }
177
178 /*
179  * Update log
180  *
181  * Feb 25 2015 NYP: Removed __forceinline__ , it is making cli too fast
182  * Feb  9 2011 JVD: Added __forceinline__ to the __device__ functions.
183  */
184
```

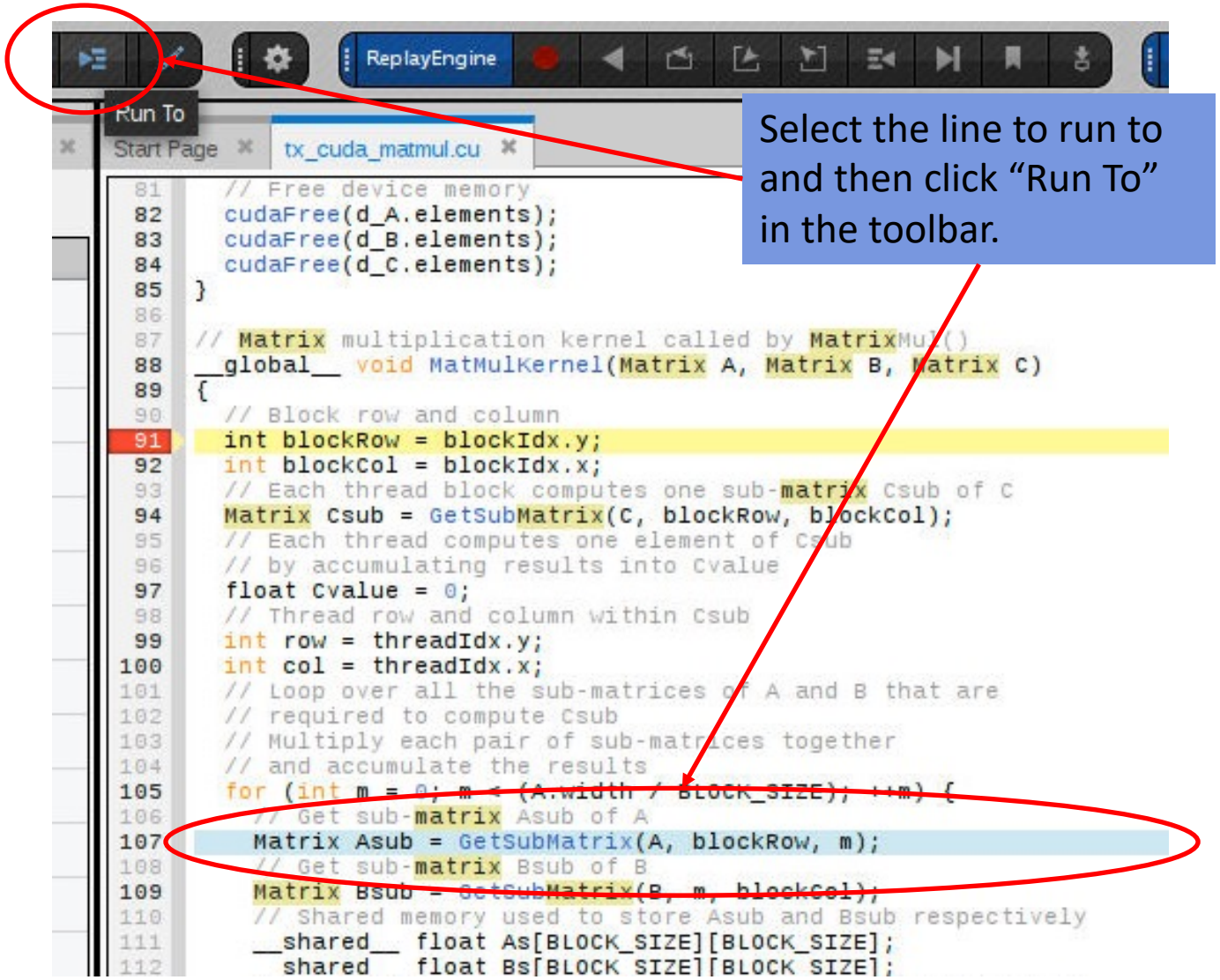
Set breakpoints on CPU host code

Source View Opened on GPU Kernel Code

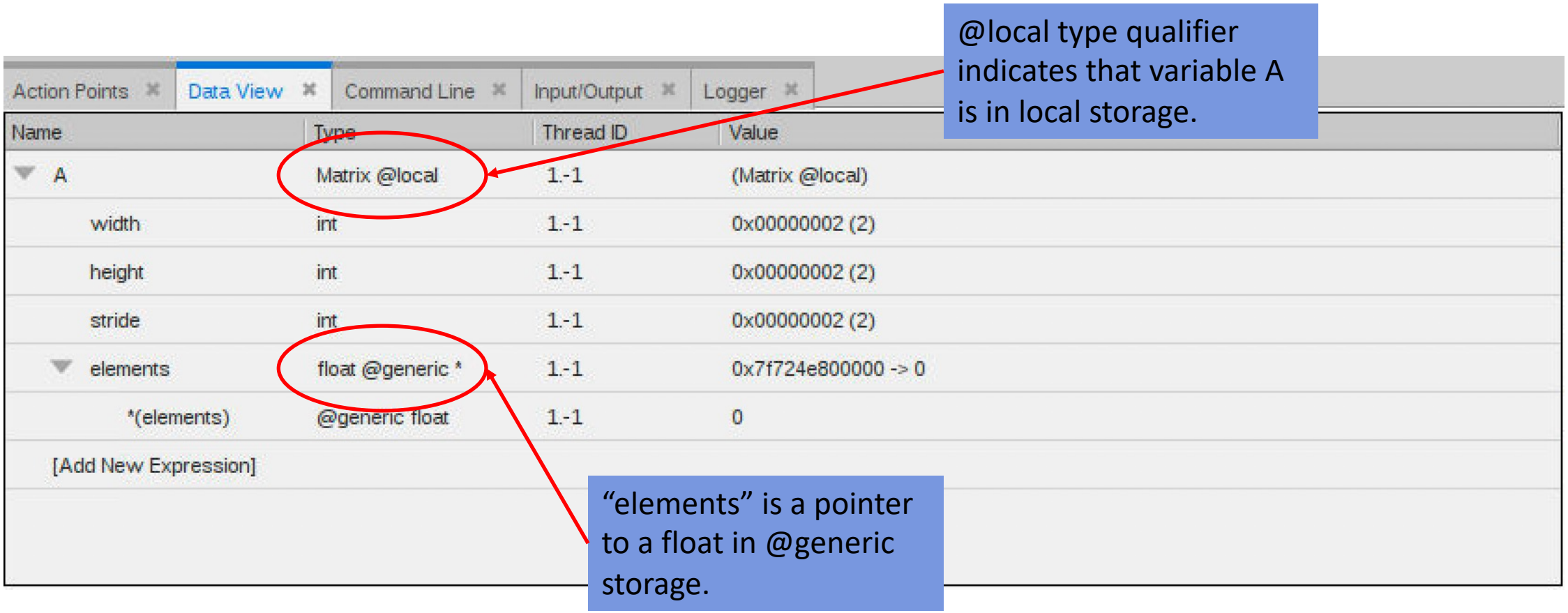
- Set breakpoints on kernel code before it is loaded
- Breakpoint first shown as a “hollow” marker
- Once kernel code is loaded onto GPU breakpoint is resolved and marker will be solid

Stepping GPU Code

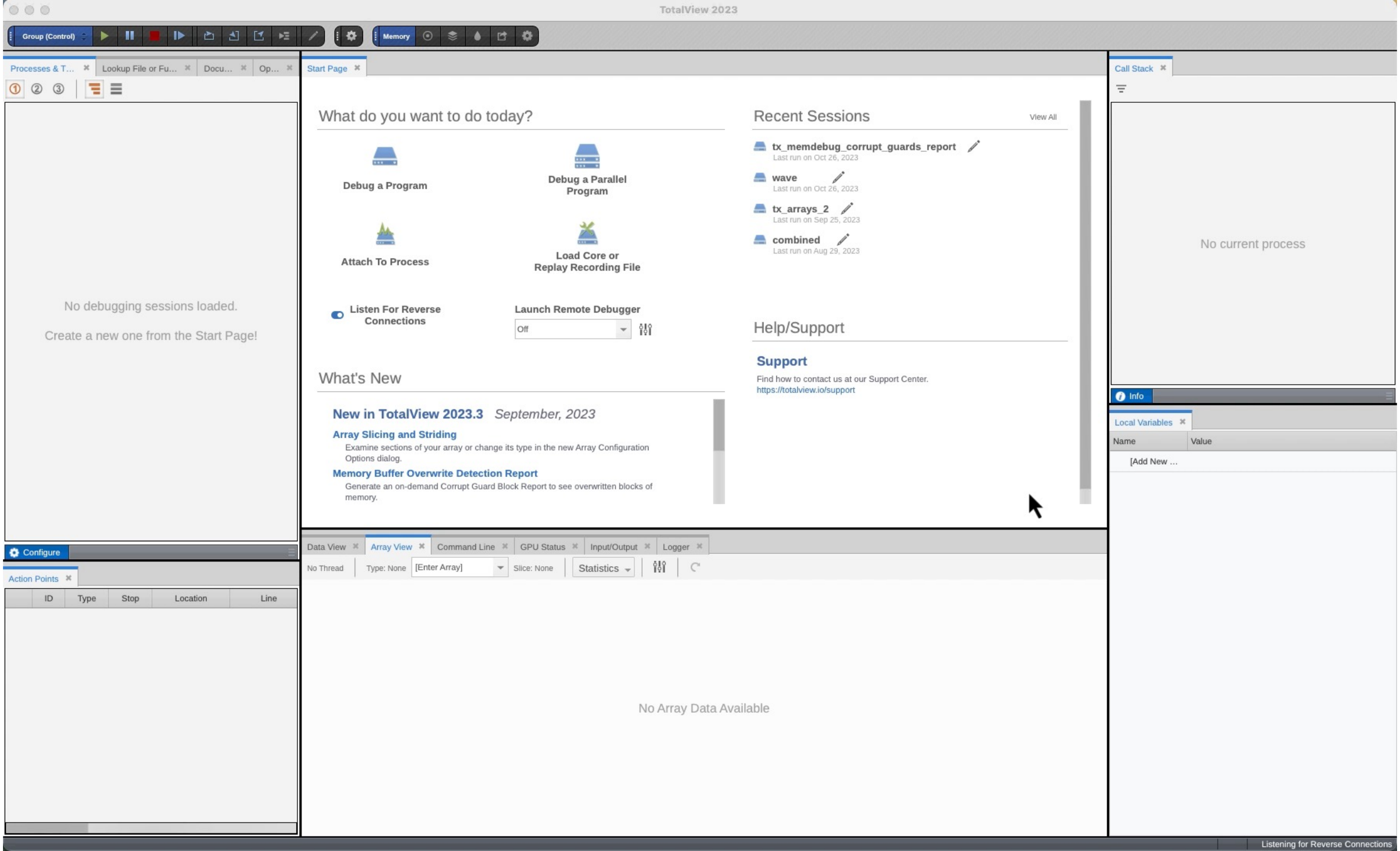
- Single-step operation advances all the GPU hardware threads in the same warp / wavefront
- To advance the execution of more than one warp, you may either:
  - Set a breakpoint and continue the process, or
  - Select a line number in the source pane and select “Run To”



Displaying CUDA Program Elements



- 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


TotalView 2023
Group (Control)
Memory
Processes & T…
Lookup File or Fu…
Docu…
Op…
No debugging sessions loaded.
Create a new one from the Start Page!
Configure
Action Points
ID
Type
Stop
Location
Line
Start Page
What do you want to do today?
Debug a Program
Debug a Parallel Program
Attach To Process
Load Core or Replay Recording File
Listen For Reverse Connections
Launch Remote Debugger
Off
What's New
New in TotalView 2023.3 September, 2023
Array Slicing and Striding
Examine sections of your array or change its type in the new Array Configuration Options dialog.
Memory Buffer Overwrite Detection Report
Generate an on-demand Corrupt Guard Block Report to see overwritten blocks of memory.
Recent Sessions
View All
tx_memdebug_corrupt_guards_report
Last run on Oct 26, 2023
wave
Last run on Oct 26, 2023
tx_arrays_2
Last run on Sep 25, 2023
combined
Last run on Aug 29, 2023
Help/Support
Support
Find how to contact us at our Support Center.
https://totalview.io/support
Data View
Array View
Command Line
GPU Status
Input/Output
Logger
No Thread
Type: None
[Enter Array]
Slice: None
Statistics
No Array Data Available
Call Stack
No current process
Info
Local Variables
Name
Value
[Add New …
Listening for Reverse Connections

Debug Hybrid MPI and OpenMP Applications

Parallel Programming Models – Hybrid Model

- A variety of parallel programming models exist to extract maximum performance out of compute resources
- Message passing models are used to maximize parallelism across compute nodes – MPI technology
- Thread models, a type of shared memory programming, is used to maximize parallelism across cores within a compute node – OpenMP technology
- A hybrid programming model combines the parallelism provided by the message passing model (MPI) with the thread model (OpenMP)
- Hybrid model also applicable to a CPU-GPU (Graphics Processing Unit) programming

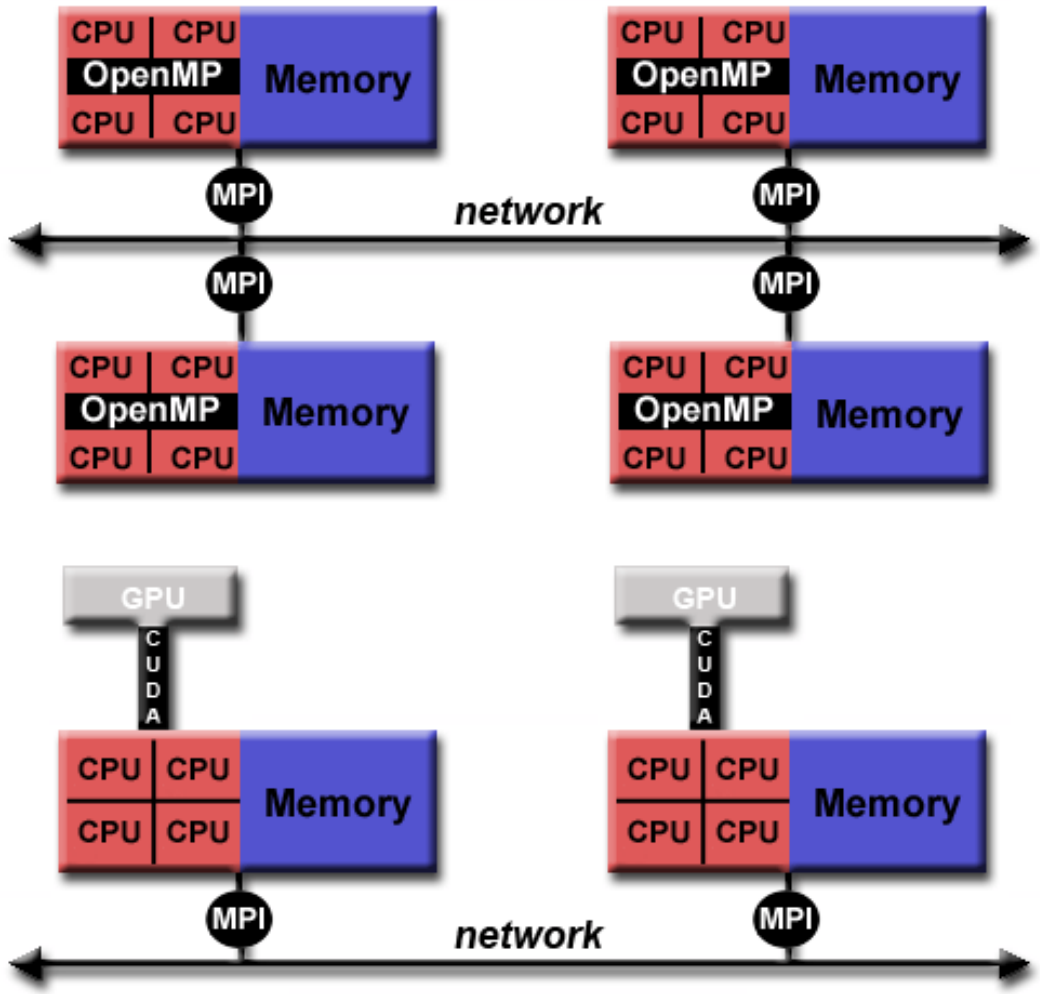
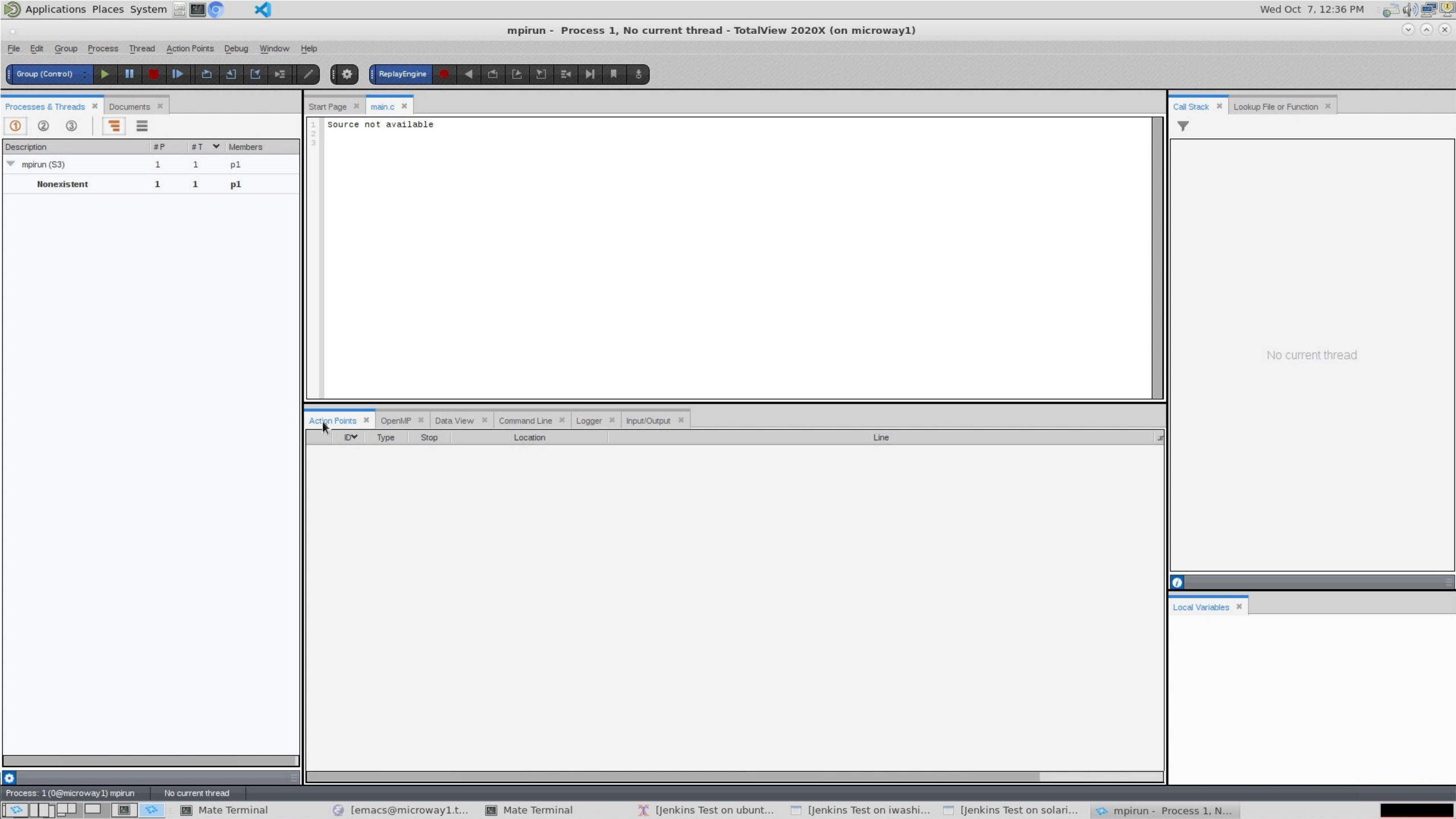
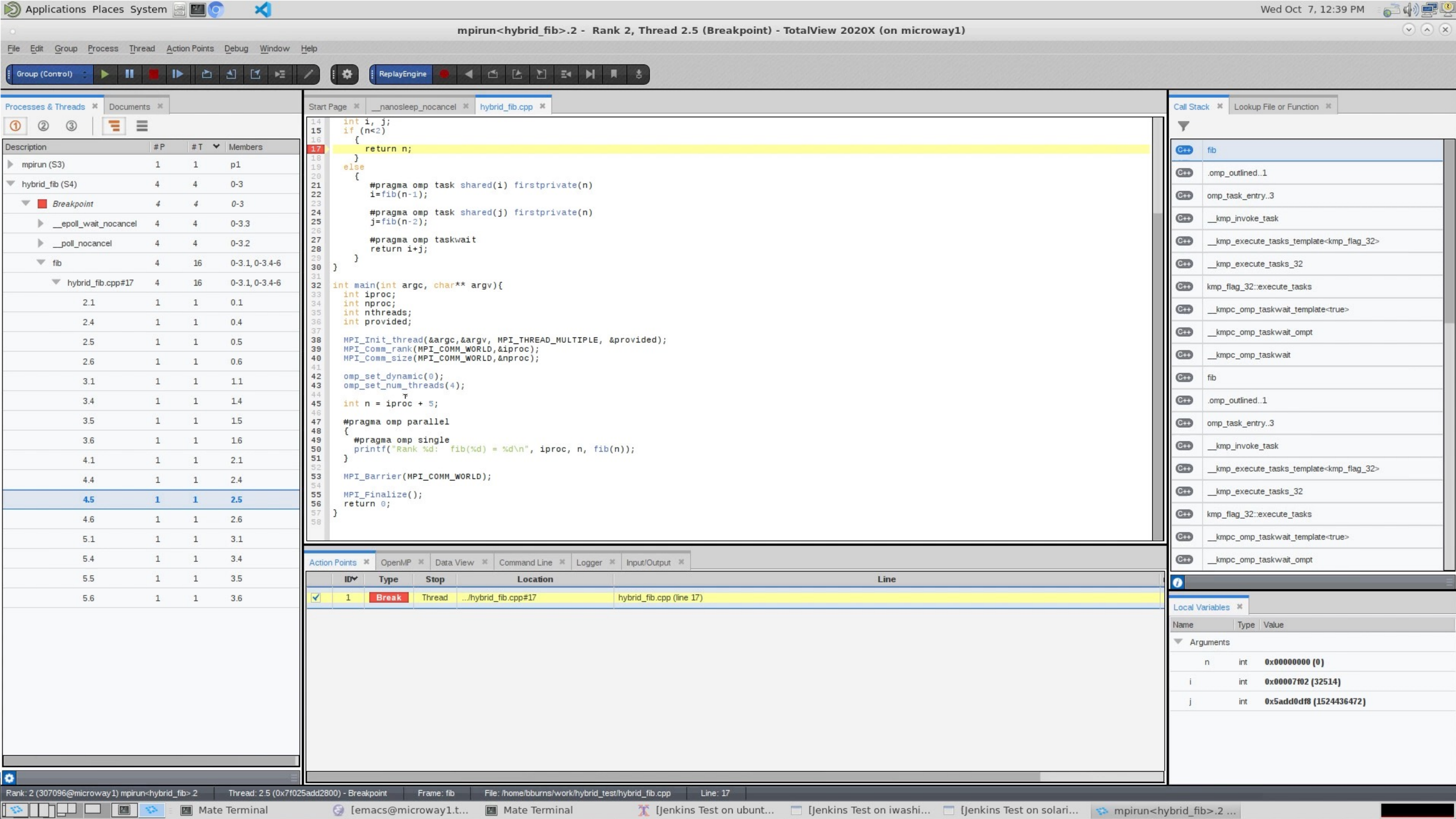


Image from U.S. Department of Energy by Lawrence Livermore National Laboratory


Applications Places System
Wed Oct 7, 12:36 PM
mpirun - Process 1, No current thread - TotalView 2020X (on microway1)
File Edit Group Process Thread Action Points Debug Window Help
Group (Control)
ReplayEngine
Processes & Threads
Documents
Description #P #T Members
mpirun (S3) 1 1 p1
Nonexistent 1 1 p1
Start Page
main.c
1 Source not available
2
3
Action Points
OpenMP
Data View
Command Line
Logger
Input/Output
ID Type Stop Location Line
Call Stack
Lookup File or Function
No current thread
Local Variables
Process: 1 (0@microway1) mpirun
No current thread
Mate Terminal
[emacs@microway1.t...
Mate Terminal
[Jenkins Test on ubunt...
[Jenkins Test on iwashi...
[Jenkins Test on solari...
mpirun - Process 1, N...

Debugging Hybrid Models – OpenMP Debugging

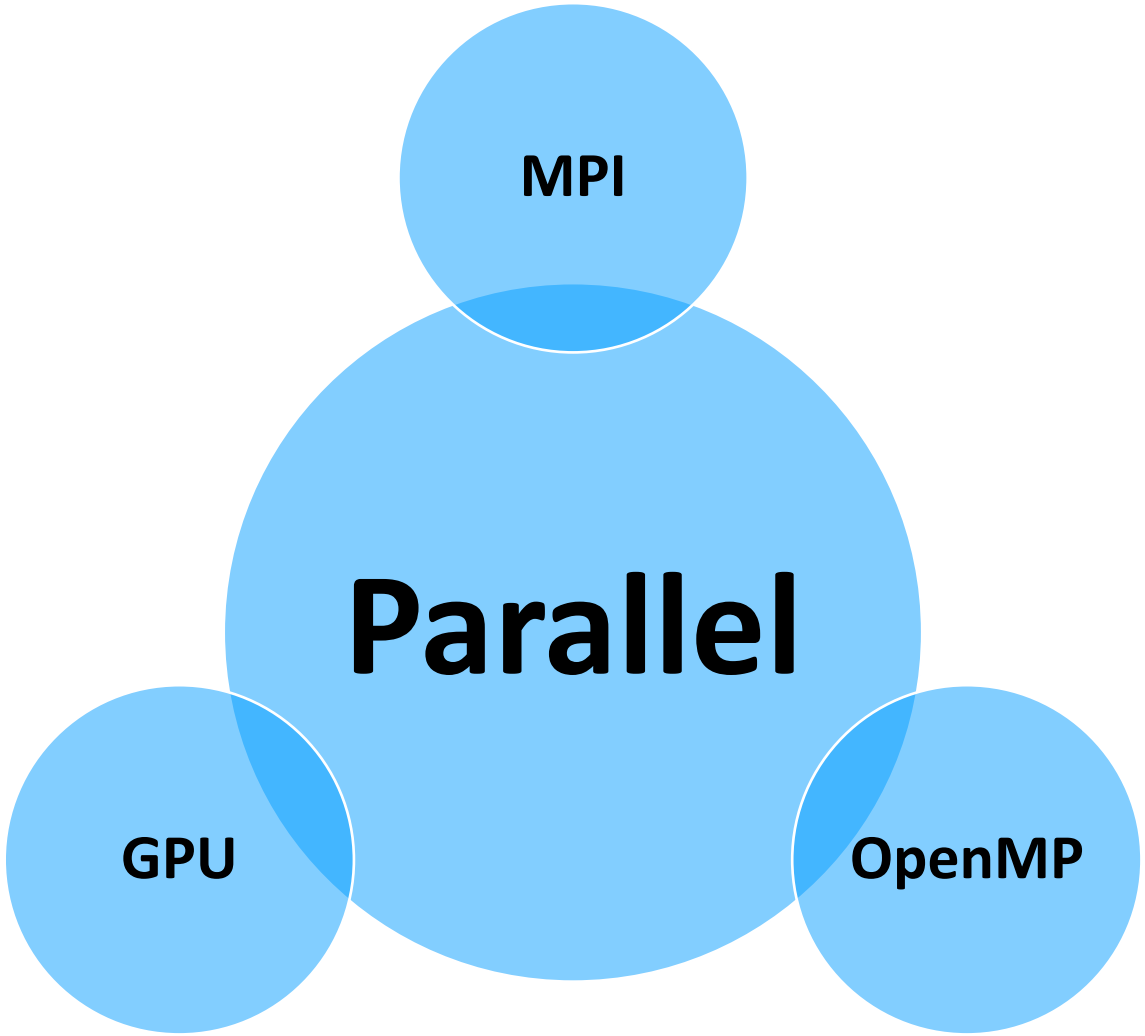
- OpenMP (Open Multi-Processing) is an API that supports multi-platform shared memory multiprocessing programming in C, C++ and Fortran
  - Compiler directives, library routines and environment variables influence run-time behavior
- Latest OpenMP Debugging API (OMPD v5.x) provides an innovative new interface allowing tools such as debuggers to extract execution state of an OpenMP runtime library
- Extract threads, parallel and task regions, internal control variables, thread relationships and runtime call-stack boundaries
- OMPD v5.x features integrated directly into TotalView's OMP debugging capabilities


Applications Places System
Wed Oct 7, 12:39 PM
mpirun<hybrid_fib>.2 - Rank 2, Thread 2.5 (Breakpoint) - TotalView 2020X (on microway1)
File Edit Group Process Thread Action Points Debug Window Help
Group (Control)
ReplayEngine
Processes & Threads
Documents
Description #P #T Members
mpirun (S3) 1 1 p1
hybrid_fib (S4) 4 4 0-3
Breakpoint 4 4 0-3
__epoll_wait_nocancel 4 4 0-3.3
__poll_nocancel 4 4 0-3.2
fib 4 16 0-3.1, 0-3.4-6
hybrid_fib.cpp#17 4 16 0-3.1, 0-3.4-6
2.1 1 1 0.1
2.4 1 1 0.4
2.5 1 1 0.5
2.6 1 1 0.6
3.1 1 1 1.1
3.4 1 1 1.4
3.5 1 1 1.5
3.6 1 1 1.6
4.1 1 1 2.1
4.4 1 1 2.4
4.5 1 1 2.5
4.6 1 1 2.6
5.1 1 1 3.1
5.4 1 1 3.4
5.5 1 1 3.5
5.6 1 1 3.6
Start Page
__nanosleep_nocancel
hybrid_fib.cpp
14 int i, j;
15 if (n<2)
16 {
17 return n;
18 }
19 else
20 {
21 #pragma omp task shared(i) firstprivate(n)
22 i=fib(n-1);
23
24 #pragma omp task shared(j) firstprivate(n)
25 j=fib(n-2);
26
27 #pragma omp taskwait
28 return i+j;
29 }
30 }
31
32 int main(int argc, char** argv){
33 int iproc;
34 int nproc;
35 int nthreads;
36 int provided;
37
38 MPI_Init_thread(&argc,&argv, MPI_THREAD_MULTIPLE, &provided);
39 MPI_Comm_rank(MPI_COMM_WORLD,&iproc);
40 MPI_Comm_size(MPI_COMM_WORLD,&nproc);
41
42 omp_set_dynamic(0);
43 omp_set_num_threads(4);
44
45 int n = iproc + 5;
46
47 #pragma omp parallel
48 {
49 #pragma omp single
50 printf("Rank %d: fib(%d) = %d\n", iproc, n, fib(n));
51 }
52
53 MPI_Barrier(MPI_COMM_WORLD);
54
55 MPI_Finalize();
56 return 0;
57 }
58
Call Stack
Lookup File or Function
C++ fib
C++ .omp_outlined..1
C++ omp_task_entry..3
C++ __kmp_invoke_task
C++ __kmp_execute_tasks_template<kmp_flag_32>
C++ __kmp_execute_tasks_32
C++ kmp_flag_32::execute_tasks
C++ __kmpc_omp_taskwait_template<true>
C++ __kmpc_omp_taskwait_ompt
C++ __kmpc_omp_taskwait
C++ fib
C++ .omp_outlined..1
C++ omp_task_entry..3
C++ __kmp_invoke_task
C++ __kmp_execute_tasks_template<kmp_flag_32>
C++ __kmp_execute_tasks_32
C++ kmp_flag_32::execute_tasks
C++ __kmpc_omp_taskwait_template<true>
C++ __kmpc_omp_taskwait_ompt
Action Points
OpenMP
Data View
Command Line
Logger
Input/Output
ID Type Stop Location Line
1 Break Thread .../hybrid_fib.cpp#17 hybrid_fib.cpp (line 17)
Local Variables
Name Type Value
Arguments
n int 0x00000000 (0)
i int 0x00007f02 (32514)
j int 0x5add0df8 (1524436472)
Rank: 2 (307096@microway1) mpirun<hybrid_fib> 2
Thread: 2.5 (0x7f025add2800) - Breakpoint
Frame: fib
File: /home/bburns/work/hybrid_test/hybrid_fib.cpp
Line: 17
Mate Terminal
[emacs@microway1.t...
Mate Terminal
[Jenkins Test on ubunt...
[Jenkins Test on iwashi...
[Jenkins Test on solari...
mpirun<hybrid_fib>.2 ...

Debugging Full Hybrid Models – MPI/OpenMP/GPU Offload

Full Hybrid GPU Debugging with TotalView

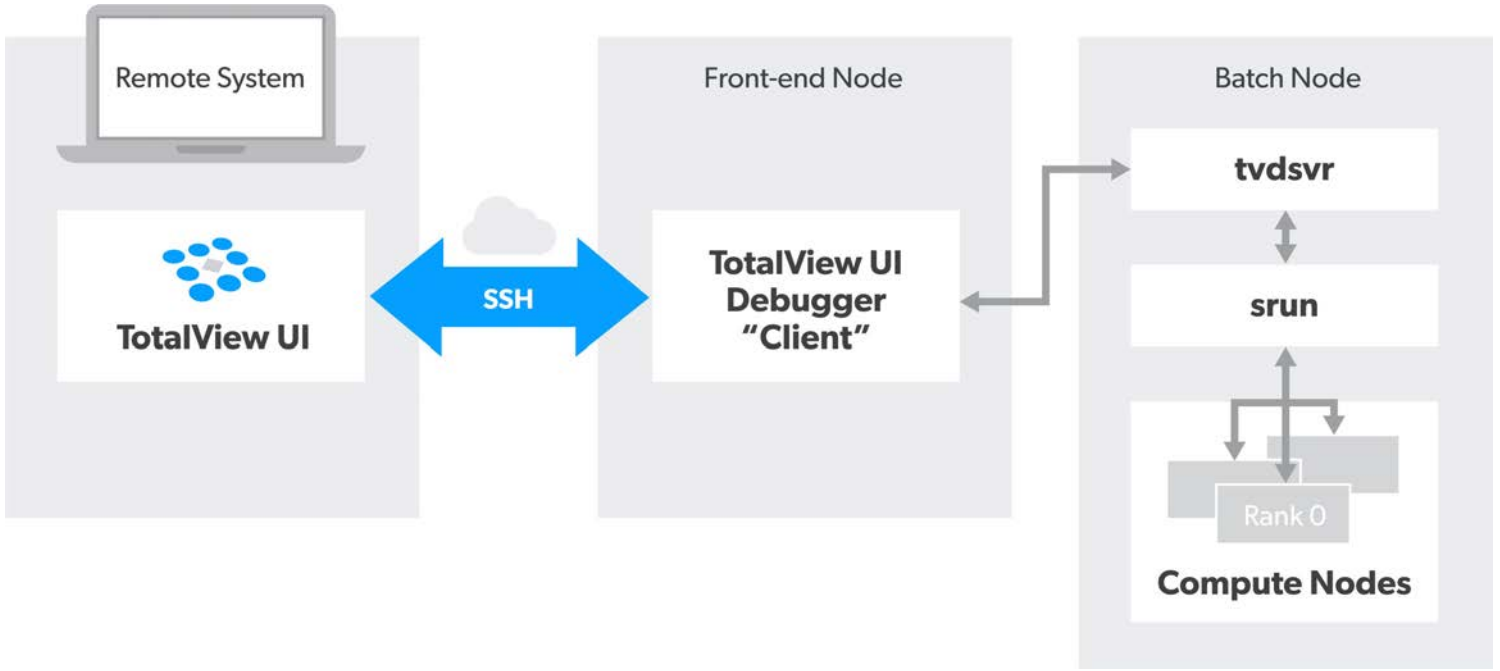
- Easily acquire MPI jobs
- Debug OpenMP tasks and regions
- Seamlessly debug GPU code as it is offloaded to a kernel



Advanced Debugger Features for Solving Tough Problems

Remote Debugging with TotalView

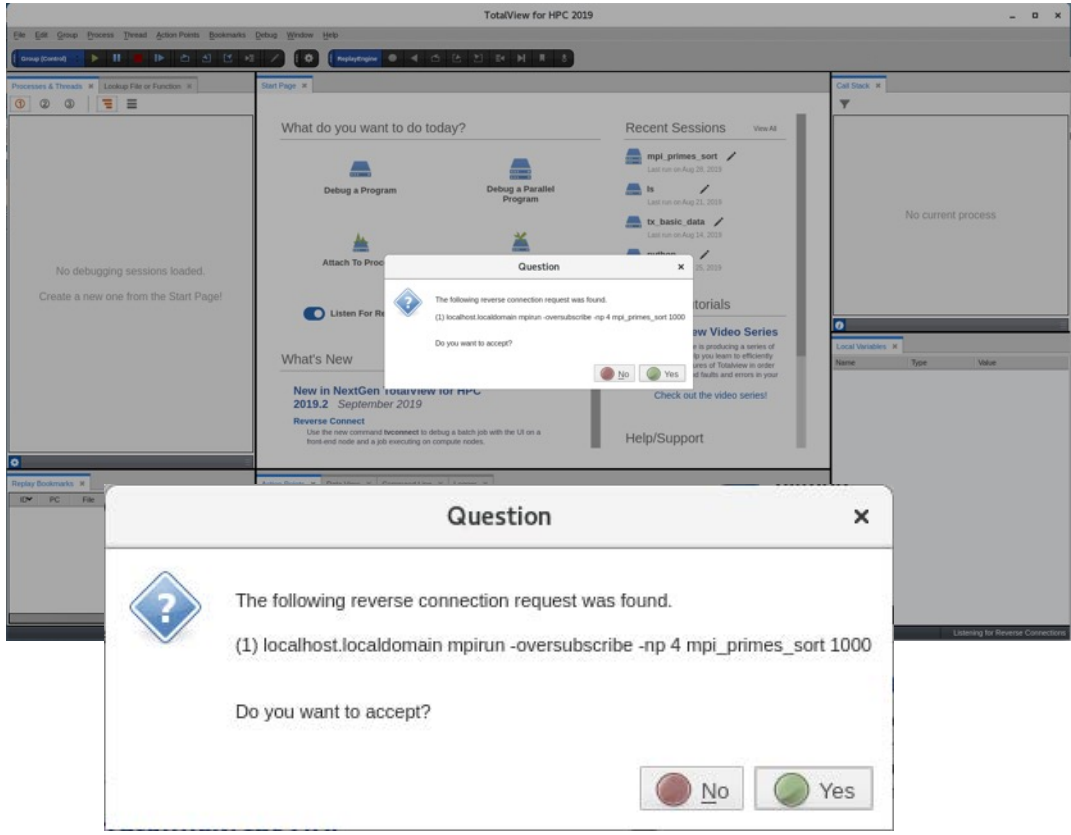
- Combine the convenience of establishing a remote connection to a cluster and the ability to run the TotalView GUI locally
- Front-end GUI architecture does not need to match back-end target architecture (macOS front-end -> Linux back-end)
- Secure communications
- Convenient saved sessions
- Once connected,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



Disconnect Backend job launch with Reverse Connect

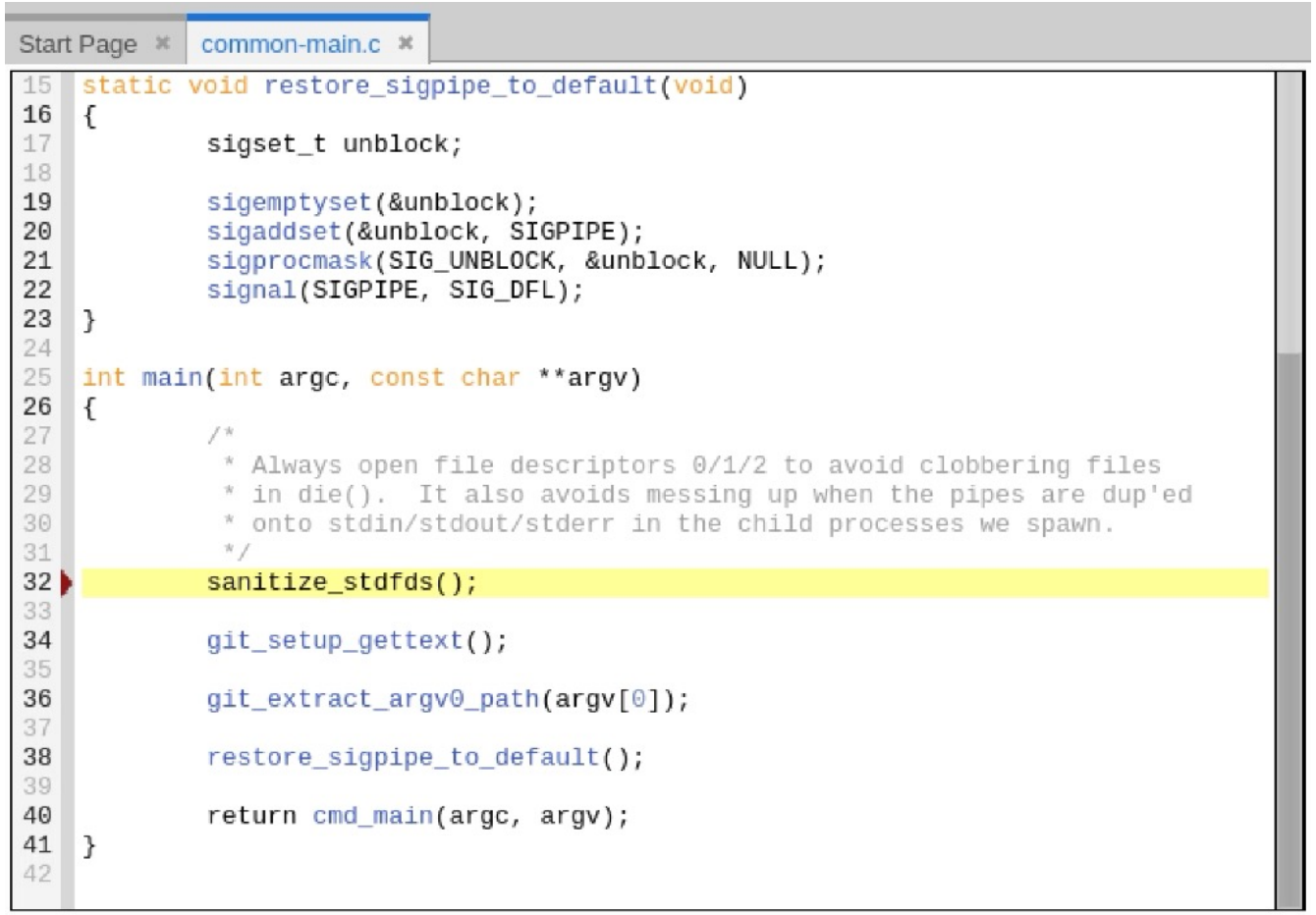
- Start a debugging session using TotalView Reverse Connect
- Reverse Connect enables the debugger to be submitted to a cluster and connected to the GUI once run
- Enables running TotalView UI on the front-end node and remotely debug jobs executing on the compute nodes
- Very easy to utilize, simply prefix job launch or application start with “tvconnect” command

```
#!/bin/bash
#SBATCH -J hybrid_fib
...
#SBATCH -n 2
#SBATCH -c 4
#SBATCH --mem-per-cpu=4000
export OMP_NUM_THREADS=4
tvconnect srun -n 2 --cpus-per-task=4 --mpi=pmix ./hybrid_fib
```



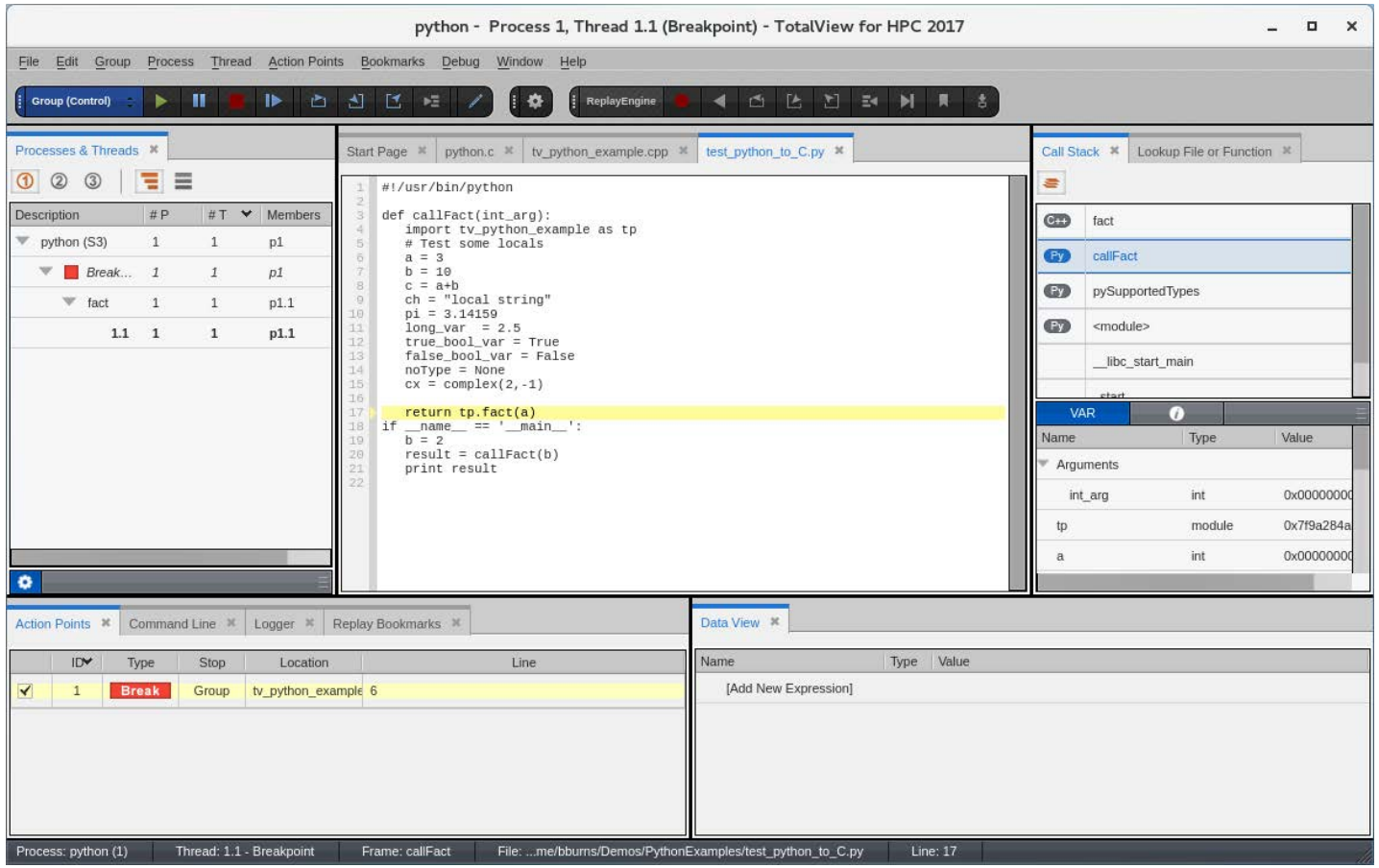
Reverse Debugging with TotalView

- Reverse debugging provides the ability for developers to go back in execution history
- Activated either before program starts running or at some point after execution begins
- Capturing and deterministically replay execution
- Enables stepping backwards and forward by function, line, or instruction
- Run backwards to breakpoints
- Run backwards and stop when a variable changes value
- Saving recording files for later analysis or collaboration
- Couple with **Undo's LiveRecorder** for maximum reverse debugging!



Mixed Language Python Debugging with TotalView

- Debugging one language is difficult enough
- Understanding the flow of execution across language barriers is hard
- Examining and comparing data in both languages is challenging
- What TotalView provides:
  - Easy python debugging session setup
  - Fully integrated Python and C/C++ call stack
  - ”Glue” layers between the languages removed
  - Easily examine and compare variables in Python and C++
  - Modest system requirements
  - Utilize reverse debugging and memory debugging

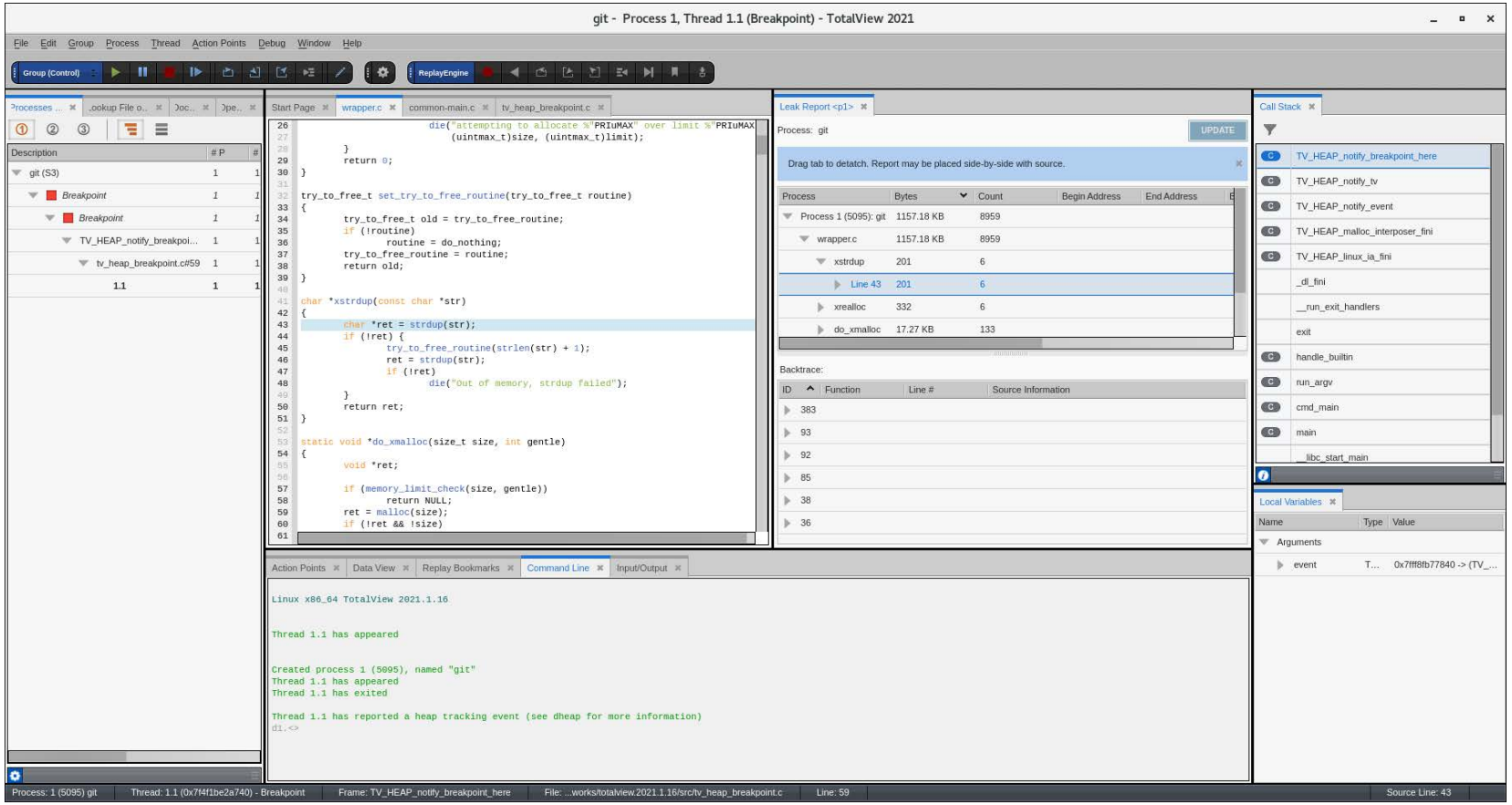


See it in action: https://totalview.io/video-tutorials/debugging-python-and-c-mixed-language-applications

Memory Debugging with TotalView

Memory Debugging Features

- Leak detection
- Dangling pointer detection
- Heap status
- Automatically detect allocation problems
- Memory Corruption Detection
- Memory Block Painting
- Memory Hoarding
- Memory Comparisons between processes
- Light weight memory block tracking

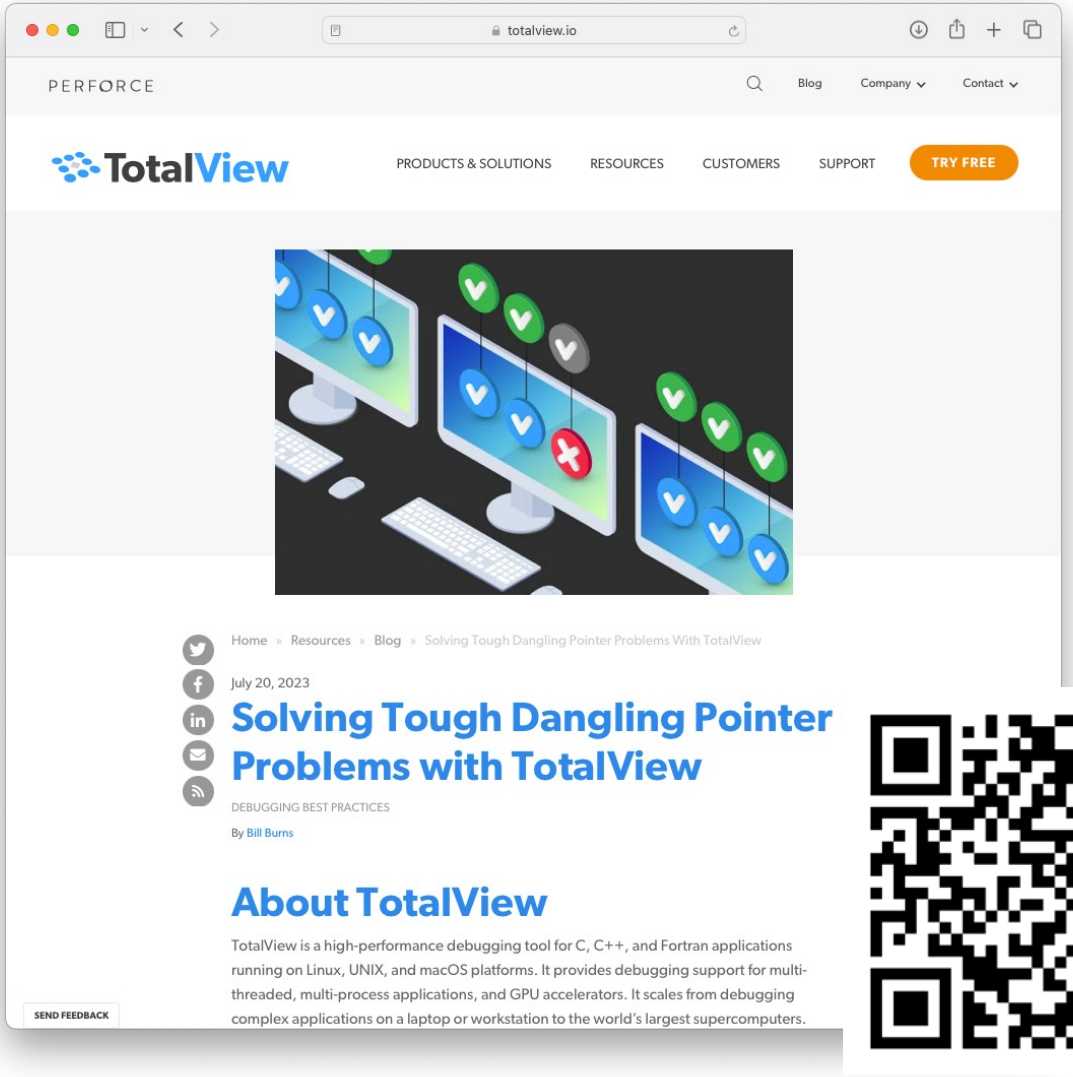


Boost Your Debugging Effectiveness

Solving Tough Dangling Pointer Problems with TotalView

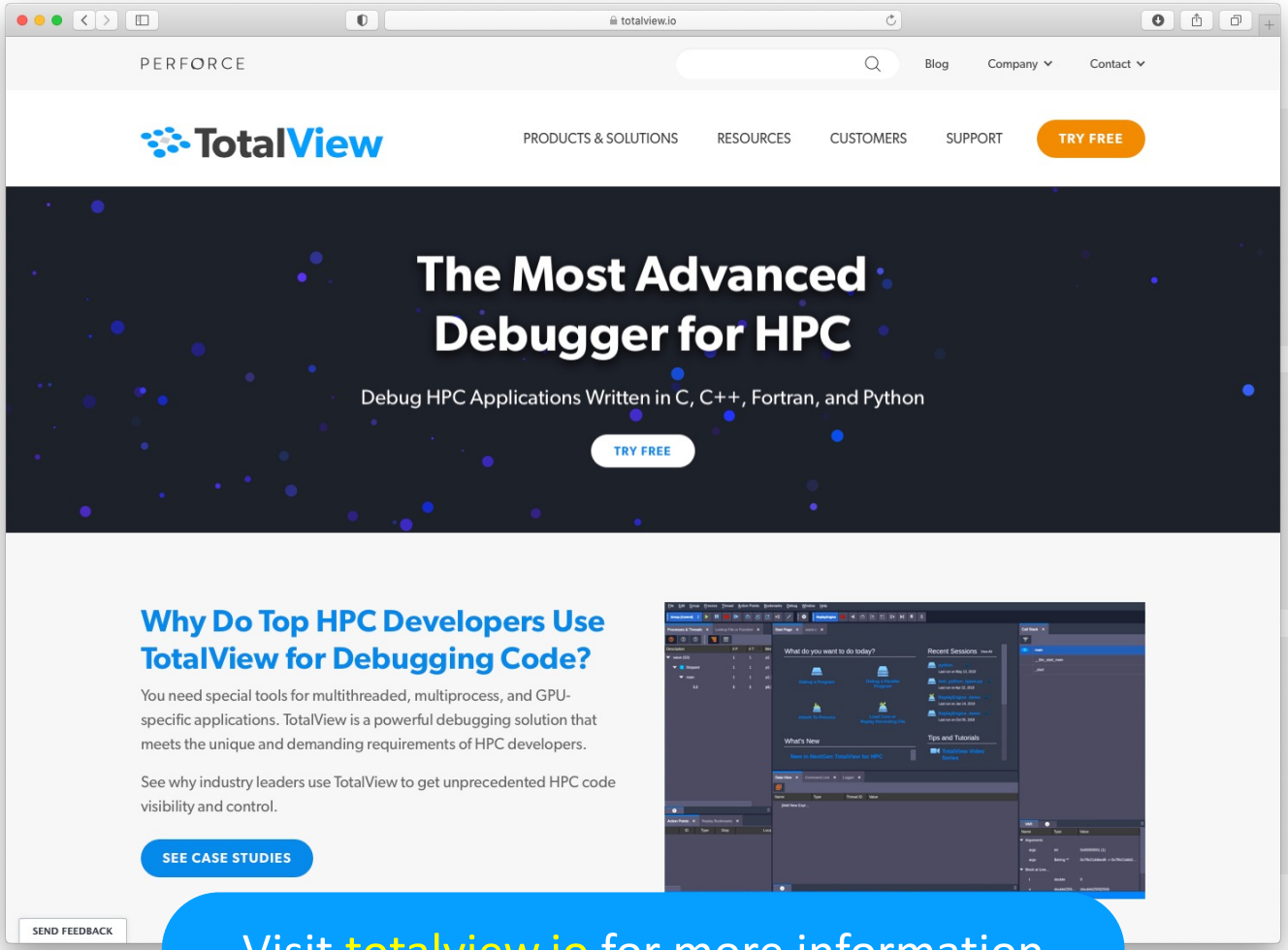
- Real world dangling pointer problem
- Combined TotalView Features
  - Memory Debugging
    - Painting memory blocks
    - Dangling pointer detection
  - Reverse Debugging
    - Easily go backwards and forwards through recorded history
  - Watchpoints
    - Stop execution when memory block contents changed

- Read about the full debugging session at:
  - https://totalview.io/blog/dangling-pointer



Q&A

Resources and Documentation



Visit totalview.io for more information

DOCUMENTATION
help.totalview.io

VIDEO TUTORIALS
totalview.io/support/video-tutorials

BLOG
totalview.io/blog

TotalView Debugging Feature References

Getting Started with TotalView

- https://totalview.io/video-tutorials/getting-started-totalview

How to Use Remote User Interface Debugging

- https://totalview.io/video-tutorials/how-use-remote-user-interface-debugging

Controlling Execution with Evaluation Points

- https://totalview.io/video-tutorials/controlling-execution-evaluation-points

Reverse Debugging

- https://totalview.io/video-tutorials/reverse-debugging

Debugging Python and C++ Mixed Language Applications

- https://totalview.io/video-tutorials/debugging-python-and-c-mixed-language-applications

Debugging the Toughest Challenges with NVIDIA and AMD GPUs

- https://totalview.io/resources/debugging-toughest-challenges-nvidia-and-amd-gpus

Blog: Solving Tough Dangling Pointer Problems with TotalView

- https://totalview.io/blog/dangling-pointer

PERFORCE

Thank You

Contact Info

Bill Burns

bburns@perforce.com

Visit us at Booth #268

SC23

Denver, CO | i am hpc.