



Advanced Parallel Debugging with TotalView

Nikolay Piskun

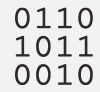
Software Architect, TotalView products, Perforce Software

August 2022

Agenda



HPC Debugging and Dynamic Analysis with TotalView



Advanced Debugging Technologies for HPC



Combining HPC Debugging Technologies

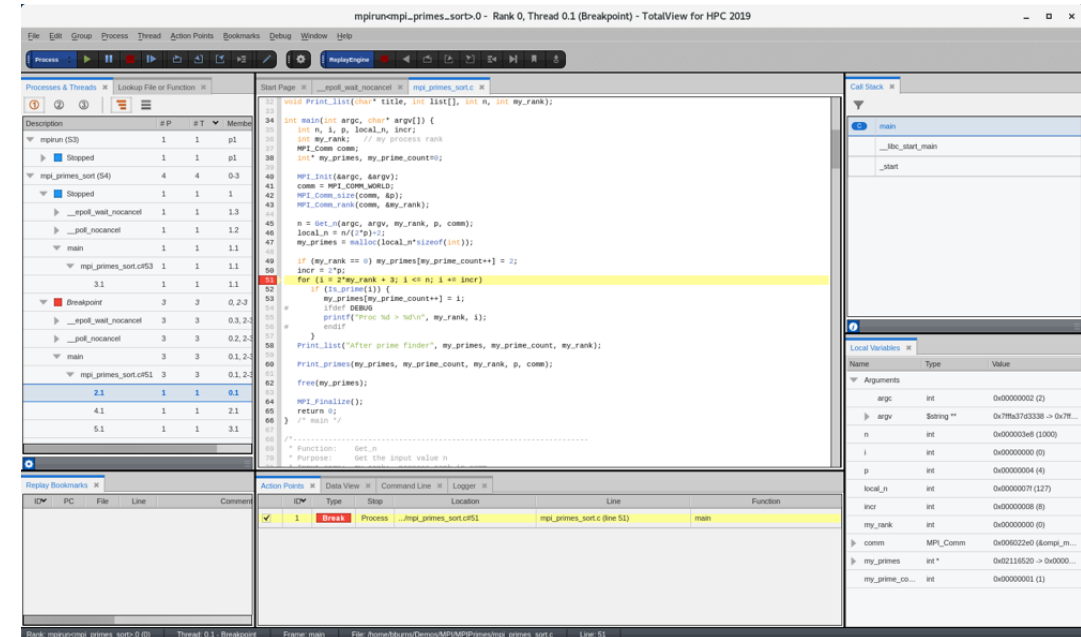


Q&A

HPC Debugging and Dynamic Analysis With TotalView

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



Supported Technologies...

LANGUAGES



OPERATING SYSTEMS



APPLICATIONS

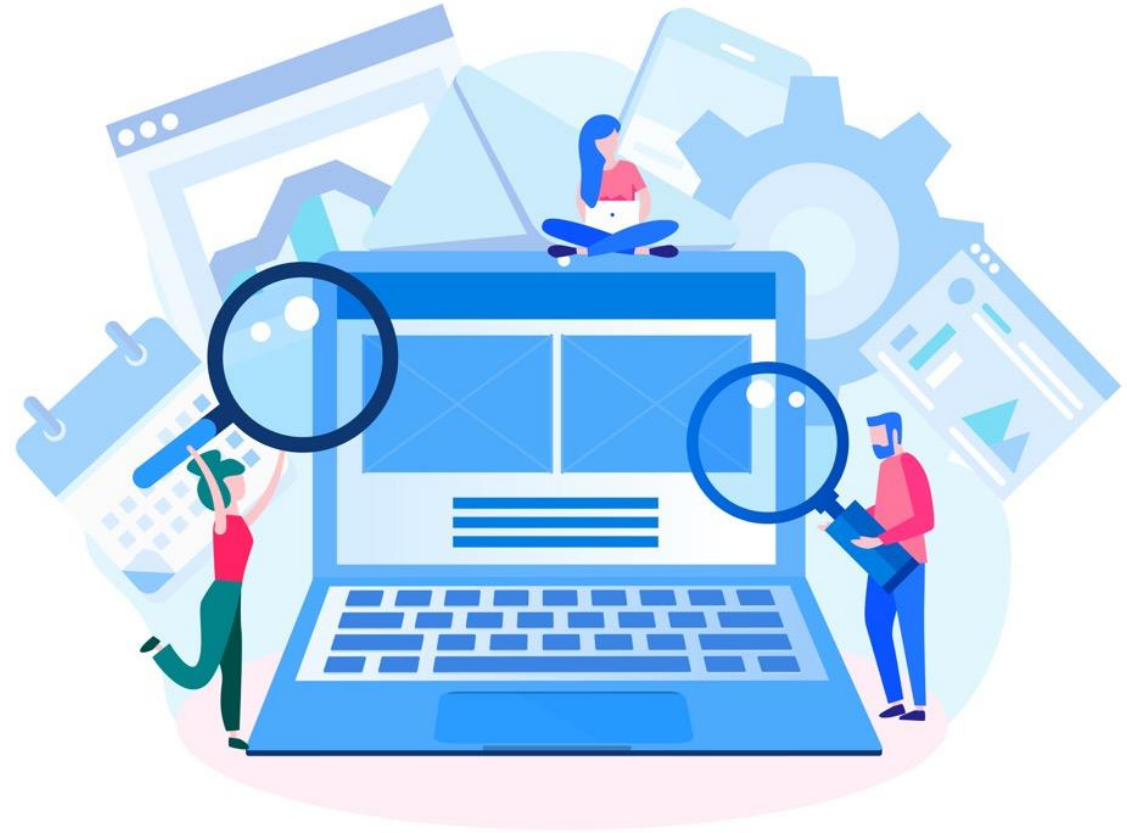


PLATFORMS



Debuggers – More Than Just a Tool to Find Bugs

- Understand complex code
- Improve developer efficiency
- Collaborate with team members
- Improve code quality
- Shorten development time



UI Navigation and Process Control

TotalView's Default Views

1. Processes & Threads

Control View

- Lookup File or Function
- Documents

2. Source View

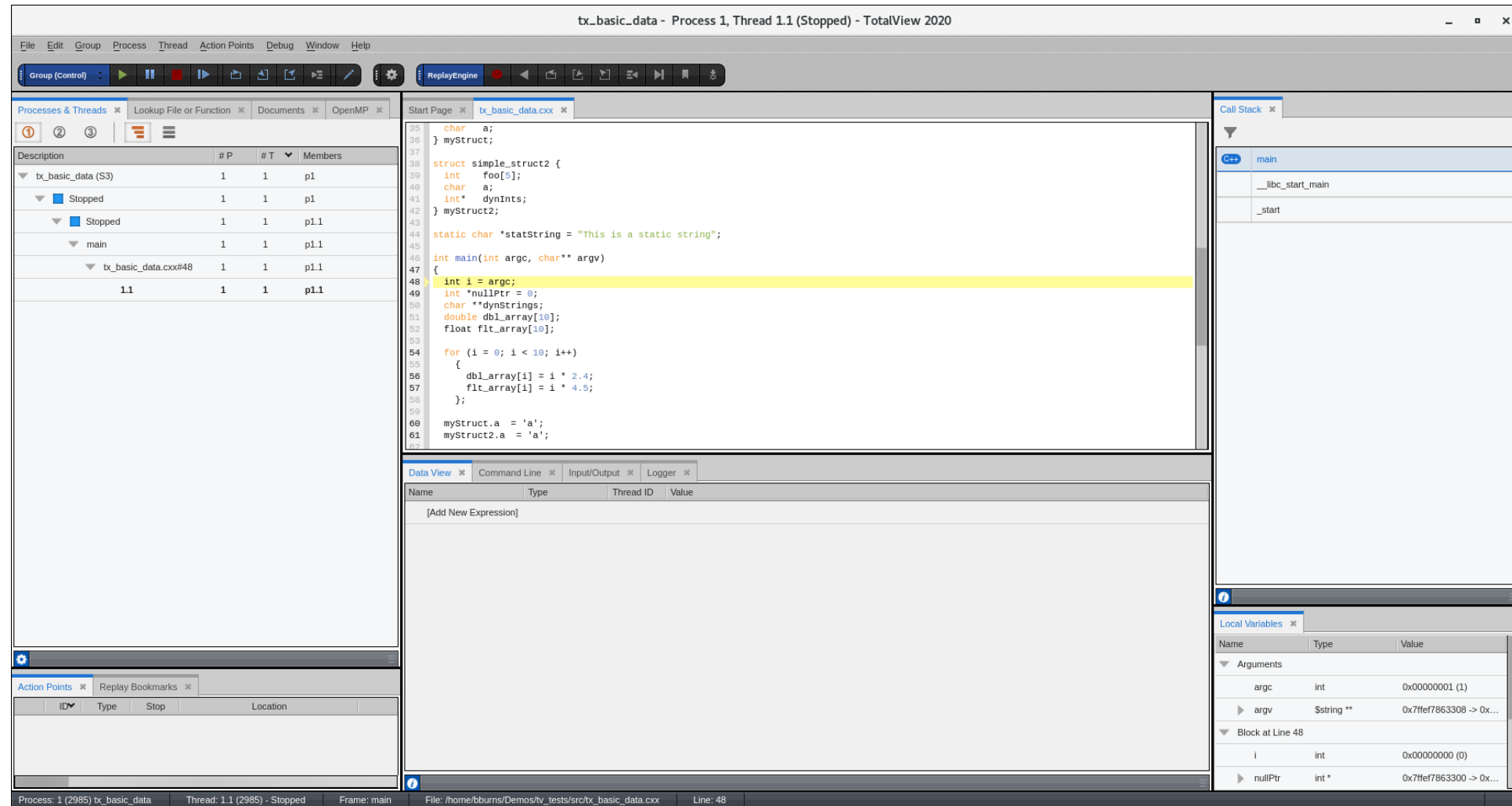
3. Call Stack View

4. Local Variables View

5. Data View, Command Line, Input/Output

6. Action Points, Replay Bookmarks

7. Array Tool



Process and Threads View

Processes & Threads			
Description	# P	# T	Members
tx_fork_loop (S3)	4	4	p1-4
Breakpoint	4	4	p1-4
Breakpoint	4	4	p2.1, p4.1, p3.2,...
snore	4	4	p2.1, p4.1, p3.2,...
tx_fork_loop.cxx#682	4	4	p2.1, p4.1, p3.2,...
1.3	1	1	p1.3
2.1	1	1	p2.1
3.2	1	1	p3.2
4.1	1	1	p4.1
Stopped	4	8	p1.1, p3.1, p1-2,...
__select_nocancel	2	3	p1-2.2, p2.3
<unknown line>	2	3	p1-2.2, p2.3
1.2	1	1	p1.2
2.2	1	1	p2.2
2.3	1	1	p2.3
snore	3	5	p1.1, p3.1, p4.2,...
tx_fork_loop.cxx#682	3	5	p1.1, p3.1, p4.2,...
1.1	1	1	p1.1
3.1	1	1	p3.1
3.3	1	1	p3.3

Processes & Threads			
Description	# P	# T	Members
tx_fork_loop (S3)	4	4	p1-4
Breakpoint	4	4	p1-4
Breakpoint	4	4	p2.1, p4.1, p3.2,...

Select process or thread attributes to group by:

☐ Control Group

☒ Share Group

☐ Hostname

☒ Process State

☒ Thread State

☒ Function

☒ Source Line

☐ PC

☐ Action Point ID

☐ Stop Reason

☐ Process ID

☒ Thread ID

☐ Process Held

☐ Thread Held

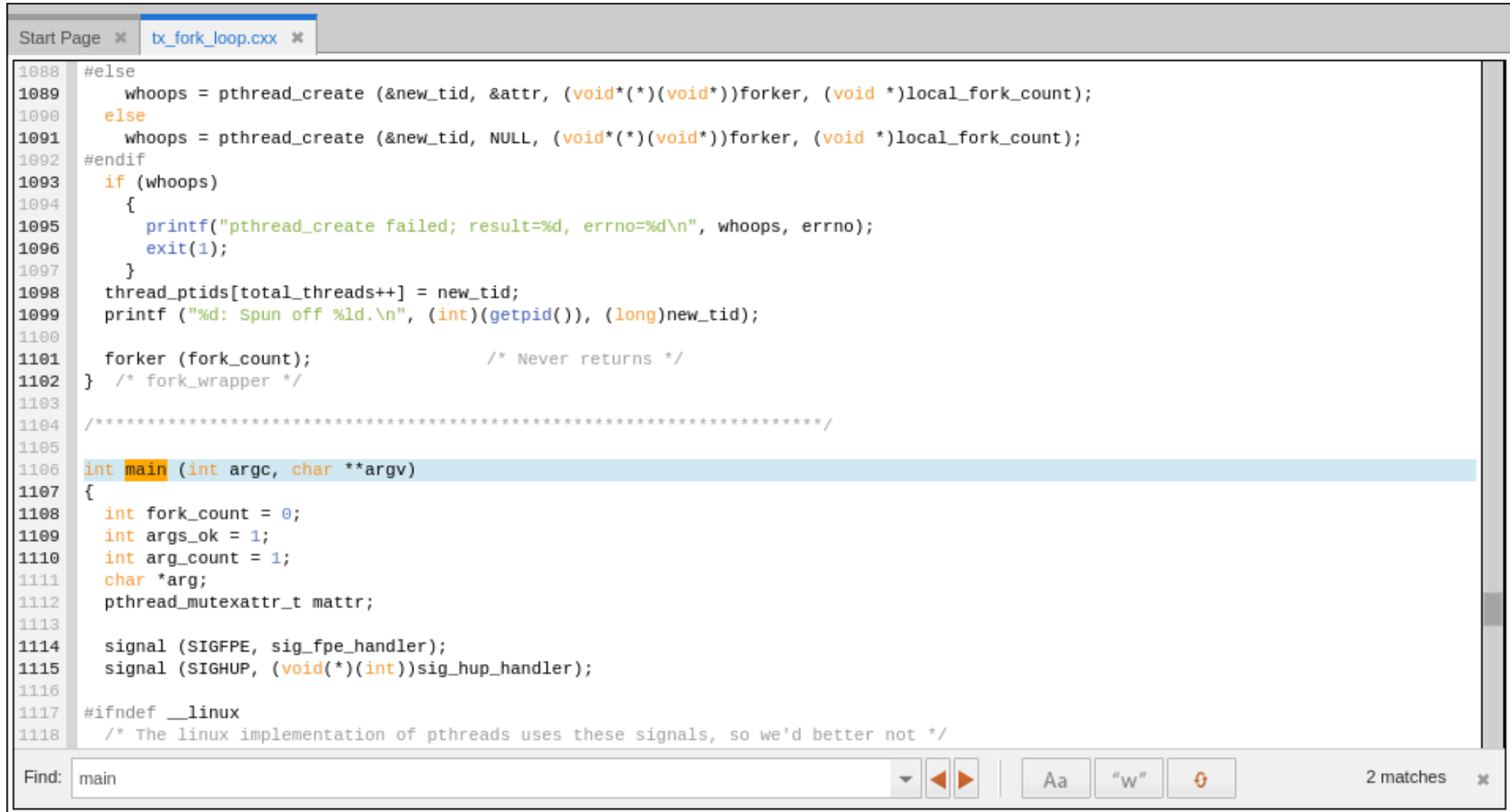
☐ Replay Mode

↑

↺

↓

Source View



```
1088 #else
1089     whoops = pthread_create (&new_tid, &attr, (void*)(*)(void*))forker, (void *)local_fork_count);
1090     else
1091     whoops = pthread_create (&new_tid, NULL, (void*)(*)(void*))forker, (void *)local_fork_count);
1092 #endif
1093     if (whoops)
1094     {
1095         printf("pthread_create failed; result=%d, errno=%d\n", whoops, errno);
1096         exit(1);
1097     }
1098     thread_ptids[total_threads++] = new_tid;
1099     printf ("%d: Spun off %ld.\n", (int)(getpid()), (long)new_tid);
1100
1101     forker (fork_count);                /* Never returns */
1102 } /* fork_wrapper */
1103
1104 /*****
1105
1106 int main (int argc, char **argv)
1107 {
1108     int fork_count = 0;
1109     int args_ok = 1;
1110     int arg_count = 1;
1111     char *arg;
1112     pthread_mutexattr_t mattr;
1113
1114     signal (SIGFPE, sig_fpe_handler);
1115     signal (SIGHUP, (void*)(int))sig_hup_handler);
1116
1117 #ifndef __linux
1118     /* The linux implementation of pthreads uses these signals, so we'd better not */
```

Find: main 2 matches

Call Stack View and Local Variables View

Call Stack	
▼	
C++	funcB
C++	funcA
C++	funcB
C++	funcA
C++	funcB
C++	funcA
C++	funcB
C++	funcA
C++	funcB
C++	funcA

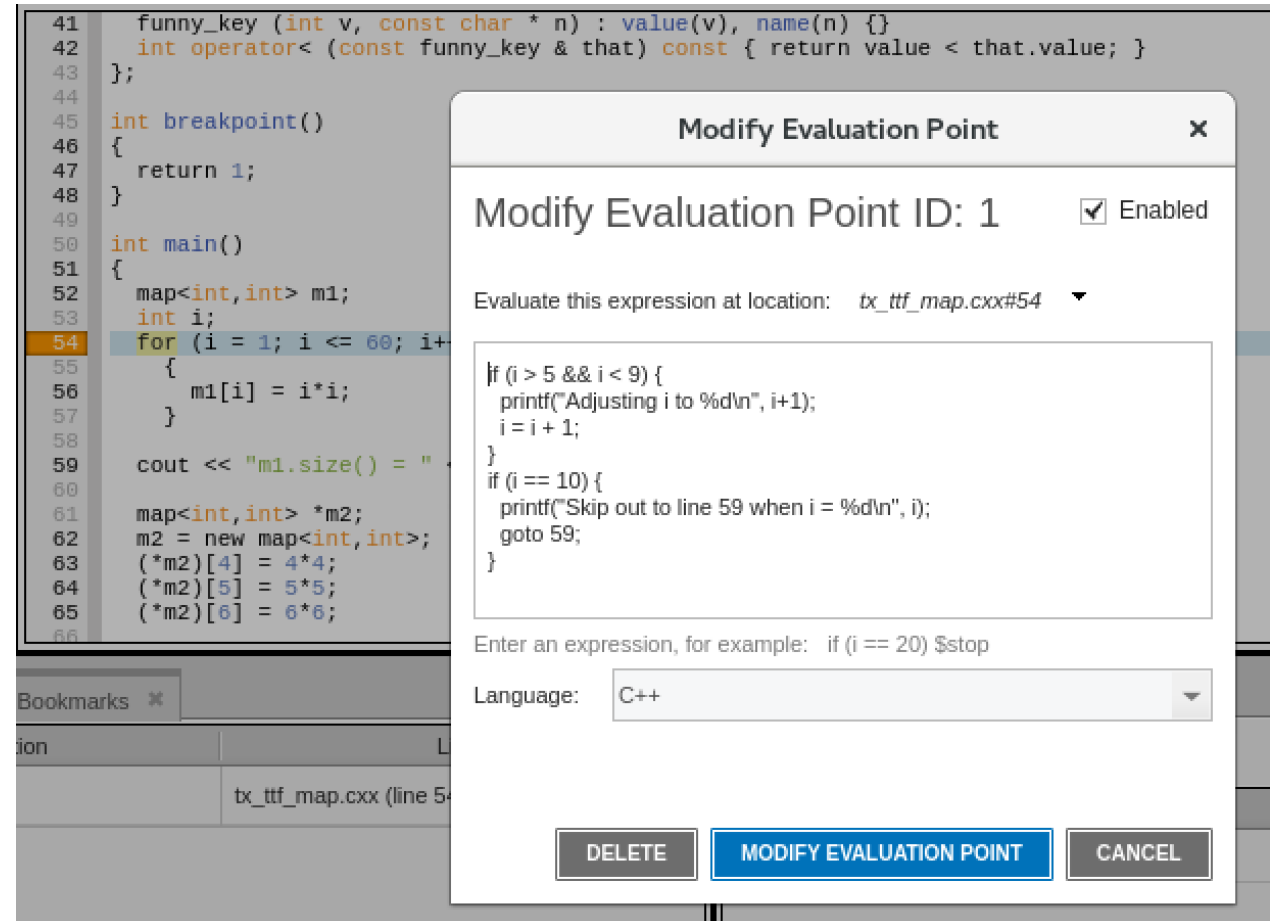
Local Variables		
Lookup File or Function		
Name	Type	Value
▼ Arguments		
b	int	0x00000012 (18)
▼ Block at Line 47		
c	int	0x00000014 (20)
i	int	0x00000000 (0)
▼ v	int[20]	(int[20])
[0]	int	0x00000000 (0)
[1]	int	0x00000000 (0)
[2]	int	0x00000000 (0)
[3]	int	0x00000000 (0)
[4]	int	0x00000000 (0)
[5]	int	0x00000000 (0)
[6]	int	0x00000000 (0)
[7]	int	0x00000000 (0)
[8]	int	0x00000000 (0)

Action Points View

OpenMP * Action Points * Data View * Replay Bookmarks * Command Line * Input/Output *						
	ID▼	Type	Stop	Location	Line	Function
<input checked="" type="checkbox"/>	1	Break	Process	.../ReplayEngine_demo.cxx#27	ReplayEngine_demo.cxx (line 27)	main
<input checked="" type="checkbox"/>	2	Watch	Group	4 bytes @ 0x601058 (arraylength)		

Patch Code With Evaluation Points

- Evaluation points allow a segment of code to be run at a line number
- Patch code on the fly
- Use special directives such as \$stopthread and \$stopprocess to control threads and processes

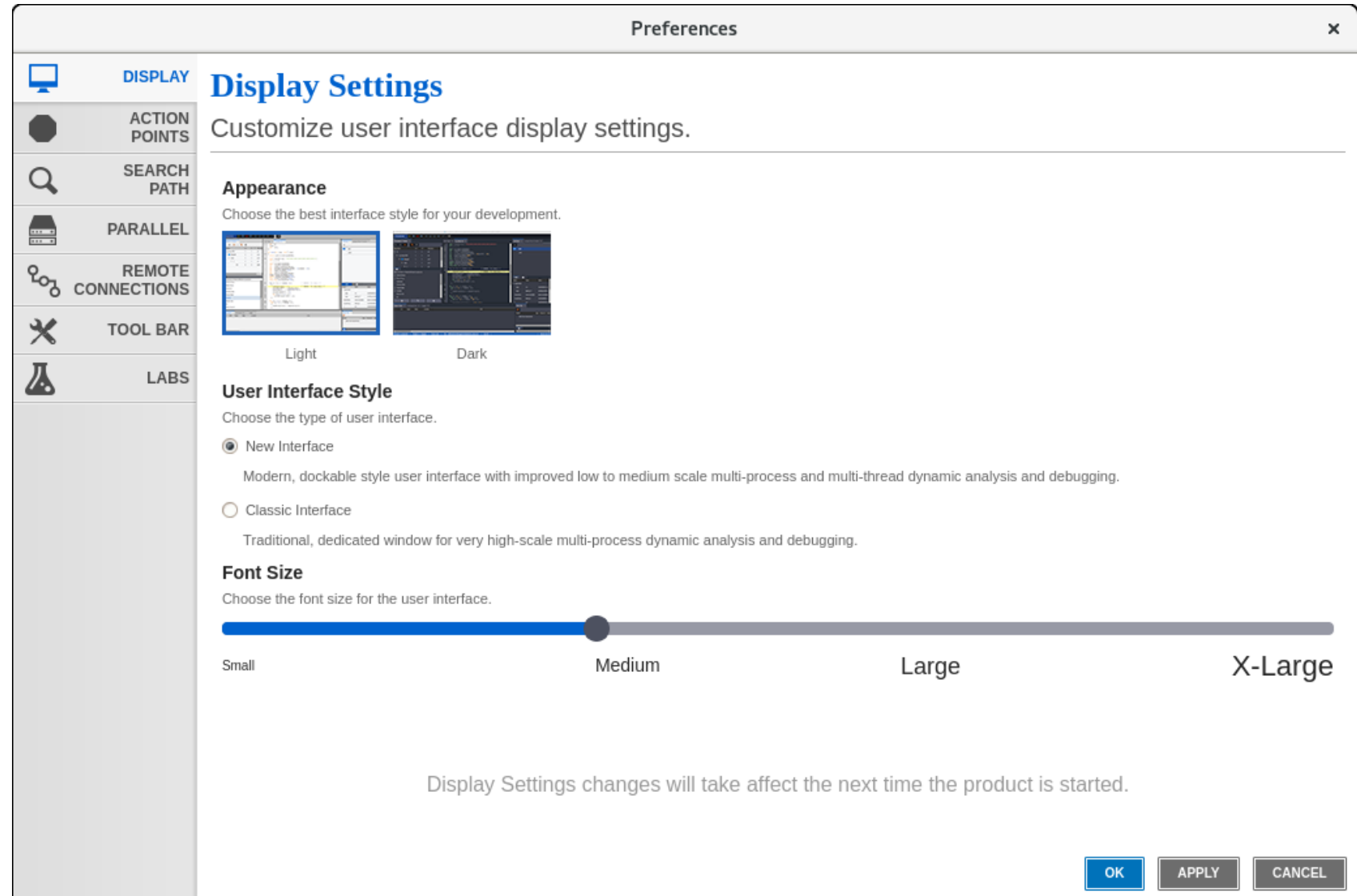


Preferences

File > Preferences Menu

Or

“Gear” Toolbar Item



A person wearing a cap and a dark shirt is seen from the side, working on a laptop in a server room. The room is filled with tall server racks containing various electronic components. The image has a blue tint and a semi-transparent blue box on the left side containing the title text.

Advanced Debugging Technologies for HPC

MPI/OpenMP/GPU Hybrid Debugging

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.

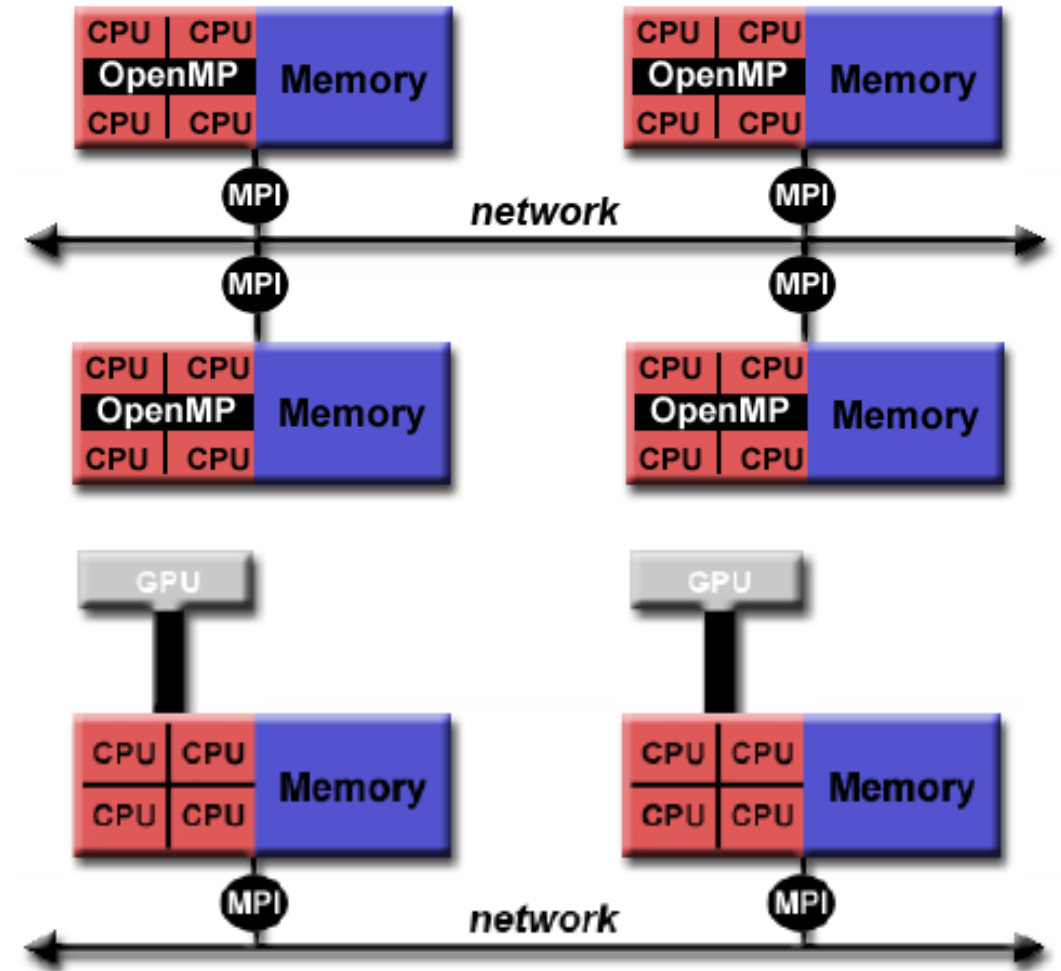
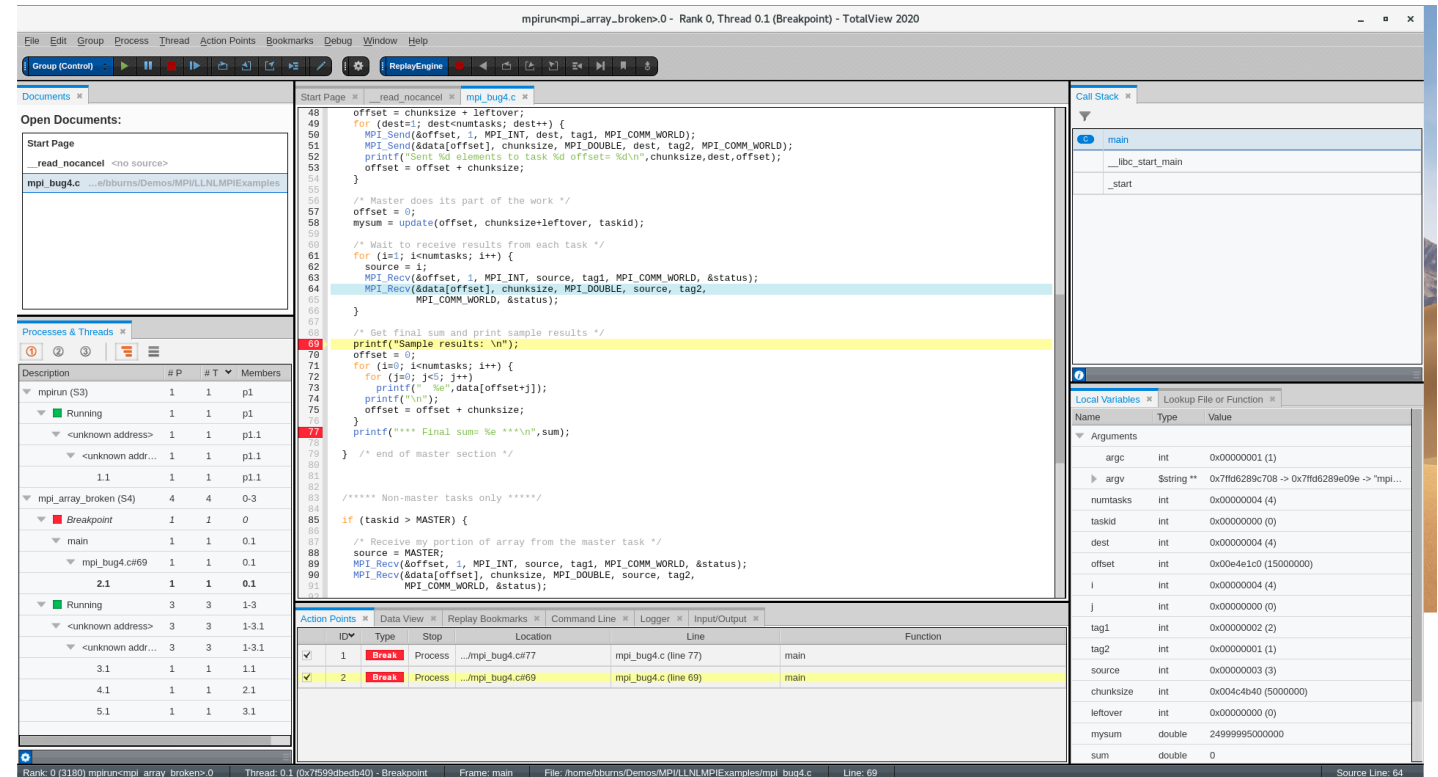


Image from [U.S. Department of Energy by Lawrence Livermore National Laboratory](https://www.llnwd.com/technology/parallel-programming-models)

Debugging Hybrid Models – MPI/OpenMP/GPU

Hybrid Debugging with TotalView

- MPI Debugging
- OpenMP Debugging
- GPU Debugging
- Hybrid debugging
 - Mixing MPI and OpenMP

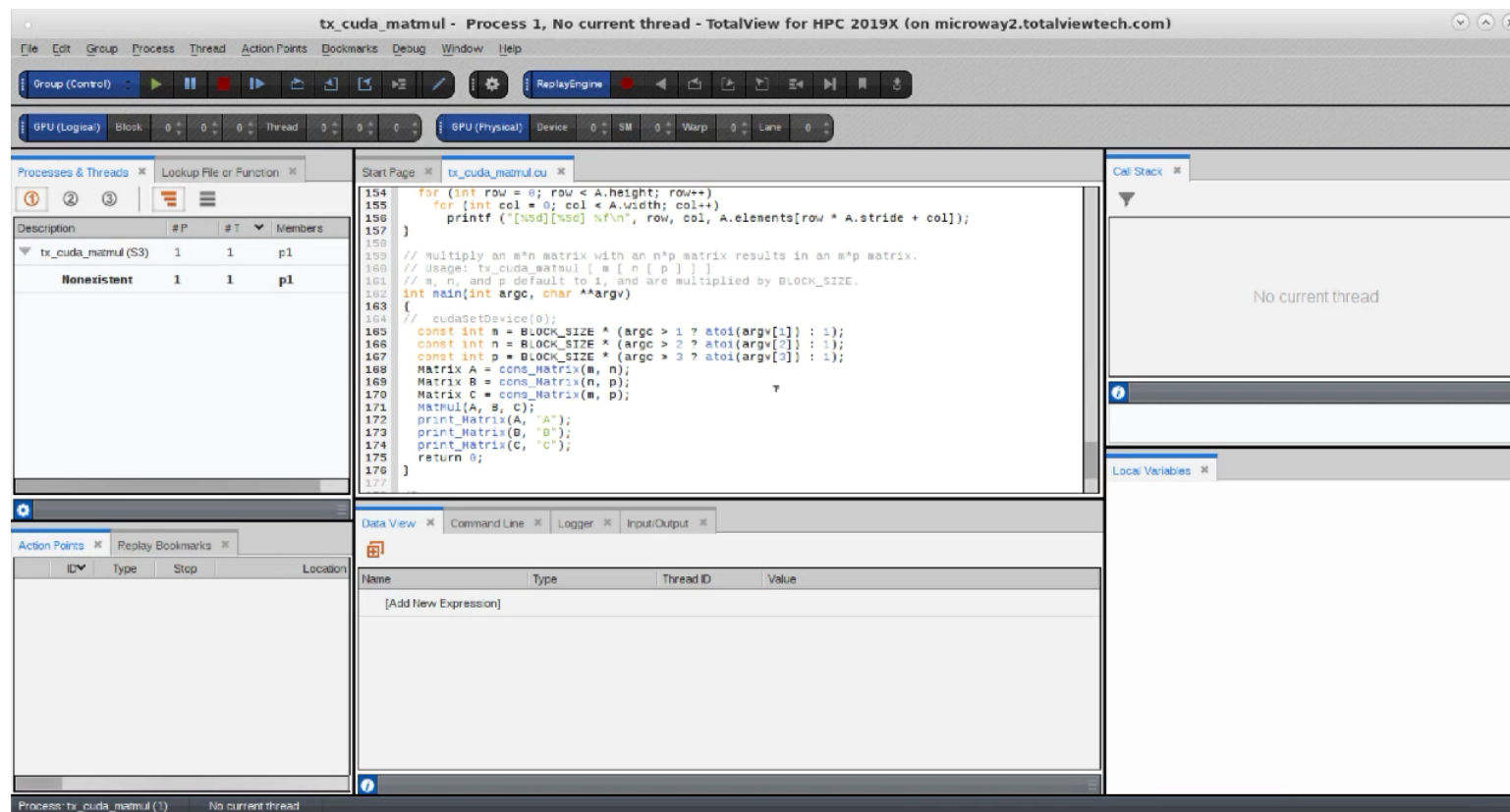


See it in action: <https://totalview.io/webinars/debugging-hybrid-mpi-openmp-applications-remotely>

CUDA Debugging

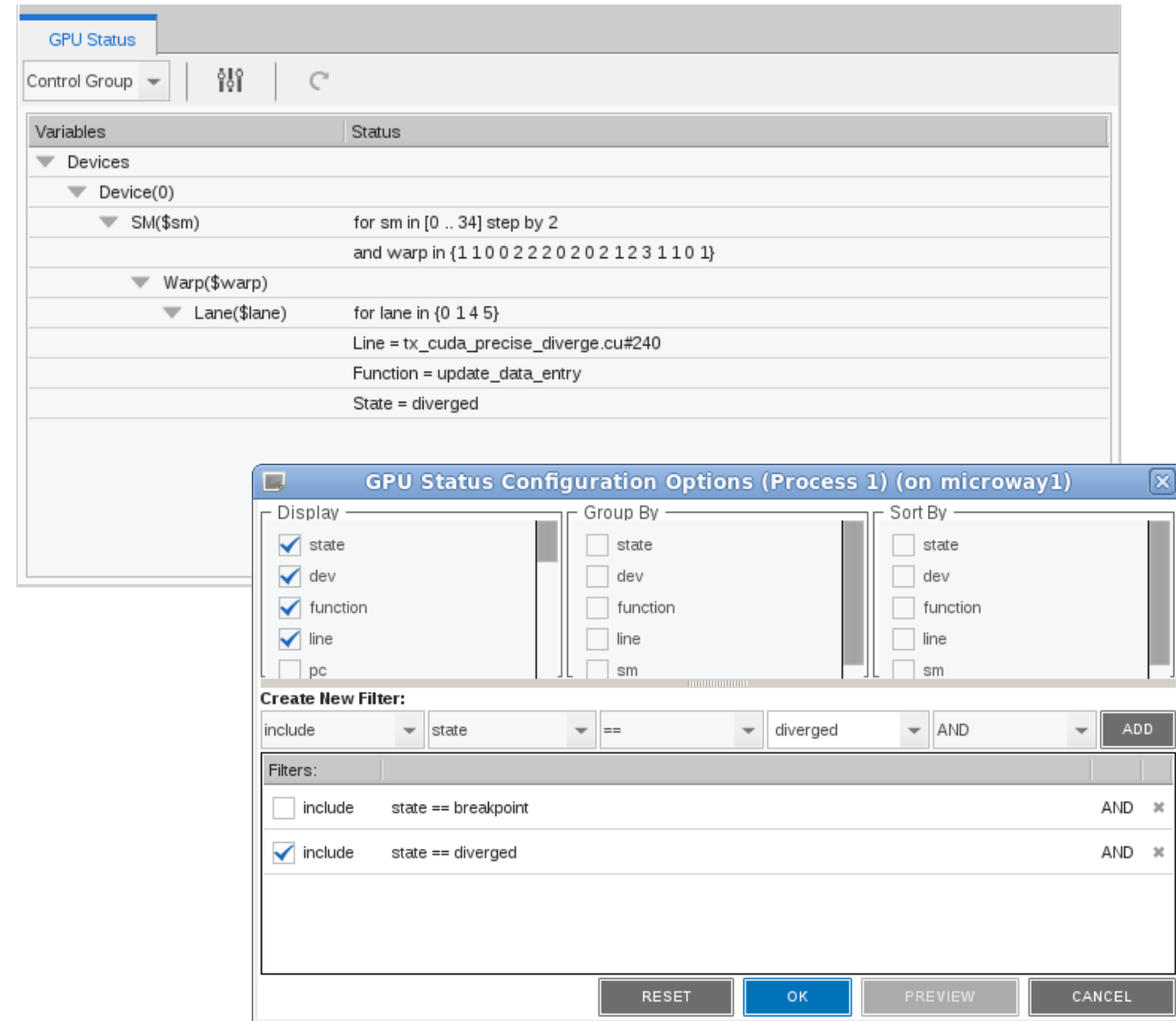
TotalView for the NVIDIA® GPU Accelerator

- NVIDIA Tesla, Fermi, Kepler, Pascal, Volta, Turing, Ampere
- NVIDIA CUDA 9.2, 10 and 11
- With support for Unified Memory
- Debugging 64-bit CUDA programs
- 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 shows how code runs on GPUs
- Support for types and separate memory address spaces
- Leverages CUDA memcheck



Advanced GPU Debugging With the GPU Status View

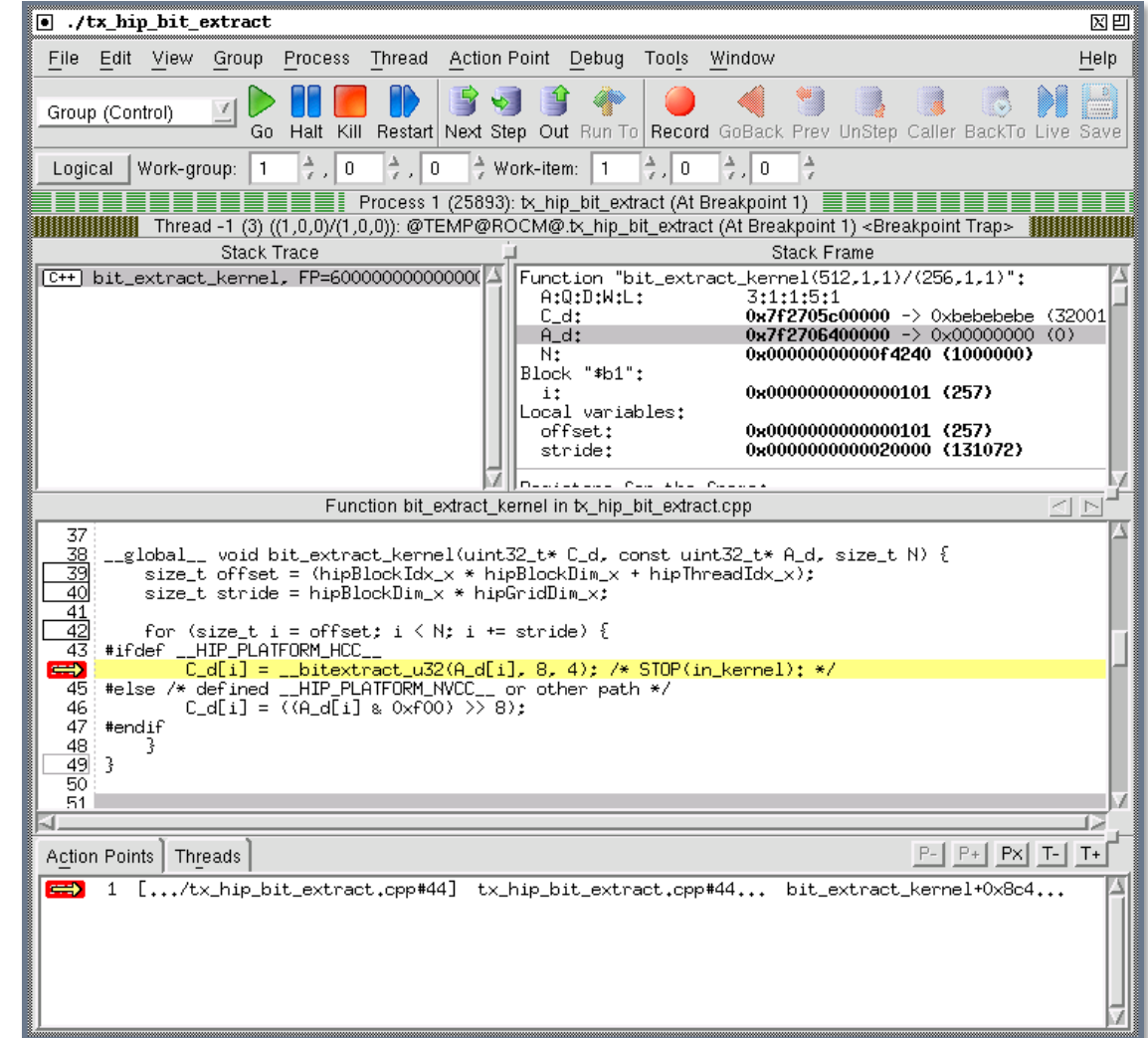
- Easily understand how your code is running across one or more GPUs.
- Use a simple attribute aggregation interface and filters to define an informative GPU Status display.
- Built to support one or more GPUs within a node and across a cluster.



AMD / ROCm GPU Debugging

TotalView ROCm GPU Support

- Process launch, attach, detach, etc.
- GPU ELF code-object load events
- Both deferred and non-deferred loading
- Registers (scalar, vector, general, special)
- Instruction disassembly
- Breakpoint create/delete, events
- Single-stepping and fast smart-stepping
- Stack unwinding (including inlined functions)
- GPU navigation controls
- Variable display (with AFAR compilers only)
- Compile as follows
 - ROCm 4.5/5.x: `"-O0 -ggdb"`
 - afar001-264: `"-O0 -mllvm -amdgpu-spill-cfi-saved-regs \-gheterogeneous-dwarf"`
 - afar001-273: `"-O0 -g"`

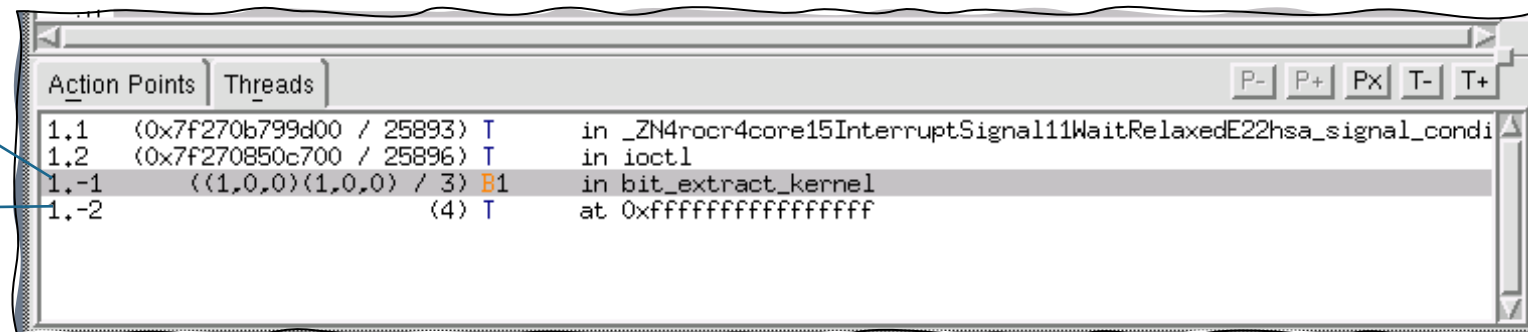


ROCm GPU Agents Are Represented As TotalView Threads

- TotalView uses a “one TotalView thread per GPU agent (device)” model (like CUDA)
- All waves on an agent within a process are grouped within a single “super thread”
- Each super thread has a GPU focus thread (a lane, within a wave, on the agent) controlled by the user

Super thread 1.-1
for Agent 3 (active)

Super thread 1.-2
for Agent 4 (idle)



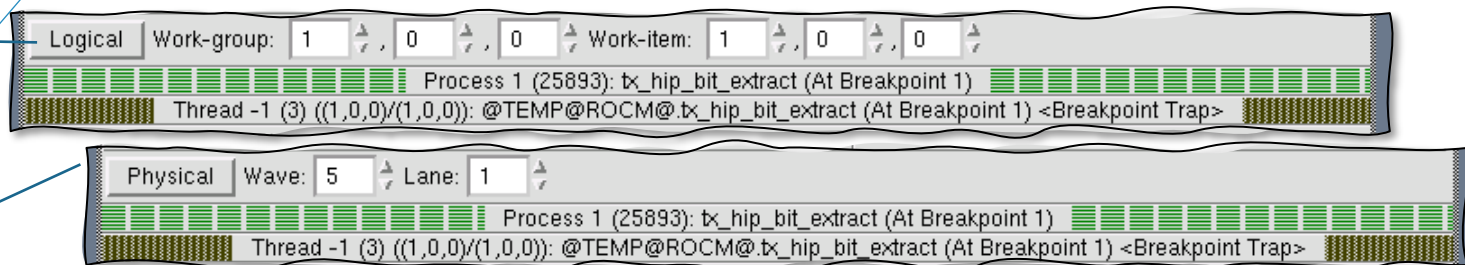
ROCm GPU Focus Control

New UI



Logical focus by
work-group / work-item

Classic UI

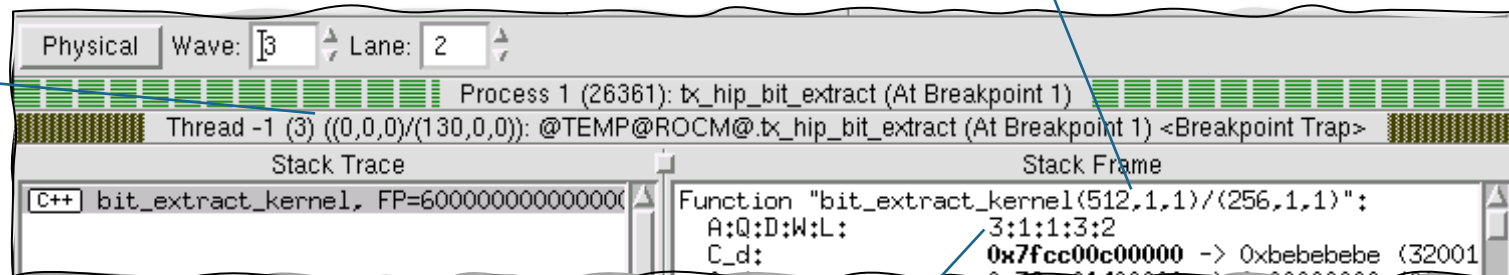


Physical focus by
for Agent 3 (active)

Logical and Physical Focus, and Grid Dimensions

Grid dimensions

Logical focus displayed in
thread status



Physical focus
Agent:Queue:Dispatch:Workgroup:Lane

Variable Display With the AFAR Compilers

- AFAR compilers can generate DWARF for variables
- There are limitations (ask AMD)
- Support planned for ROCm 5.1 (AFAIK)
- Built-in variables (block/thread idx/dim) can be displayed and used in expressions

Parameter and local variables

Data display

The screenshot shows the TotalView debugger interface. The main window displays the source code of the `bit_extract_kernel` function in `tx_hip_bit_extract.cpp`. The function signature is `__global__ void bit_extract_kernel(uint32_t* C_d, const uint32_t* A_d, size_t N)`. The code includes a loop that iterates over the array `A_d` and extracts bits from it. The `Stack Trace` and `Stack Frame` panels show the current state of the program, including the function `bit_extract_kernel(512,1,1)/(256,1,1)` and its parameters `A_d` and `N`. The `Registers` panel shows the values of the registers. The `Action Points` and `Threads` panels show the current state of the program. The `Expression` panel shows the expression `((const uint32_t[1000000]) A_d)` and its address `0x7fcc01400000`. The `Data display` panel shows the values of the array `A_d` at the current address.

Field	Value
[0]	0x00000000 (0)
[1]	0x00000001 (1)
[2]	0x00000002 (2)
[3]	0x00000003 (3)
[4]	0x00000004 (4)
[5]	0x00000005 (5)
[6]	0x00000006 (6)
[7]	0x00000007 (7)
[8]	0x00000008 (8)

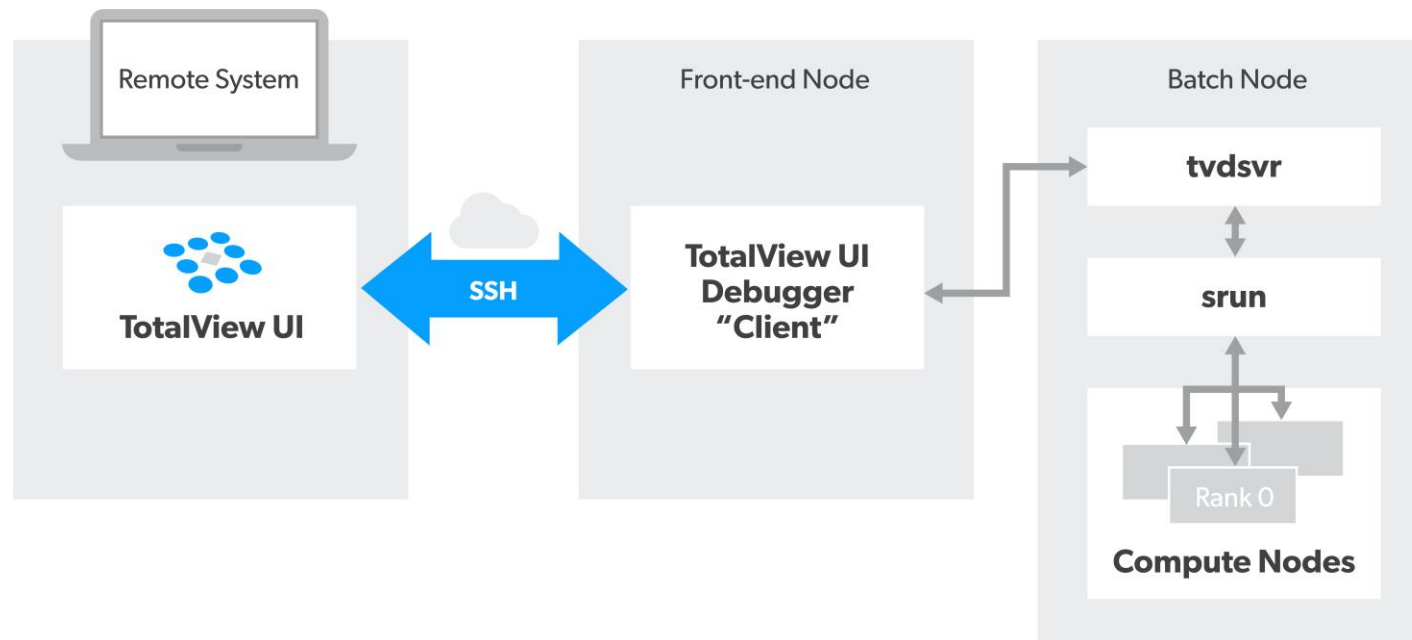
Debugging AMD GPUs with TotalView for AMD GPUs

- TotalView does not “officially” support AMD GPUs yet, but...
- “Unofficial” support is included in production versions of TotalView
- Official AMD GPU support coming later this year
- Enabled it using the “-rocm” flag, for example:
 - `totalview -rocm a.out`
- Latest TotalView 2022.2 version supports ROCm 5.1

Remote Debugging

TotalView Remote UI

- 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



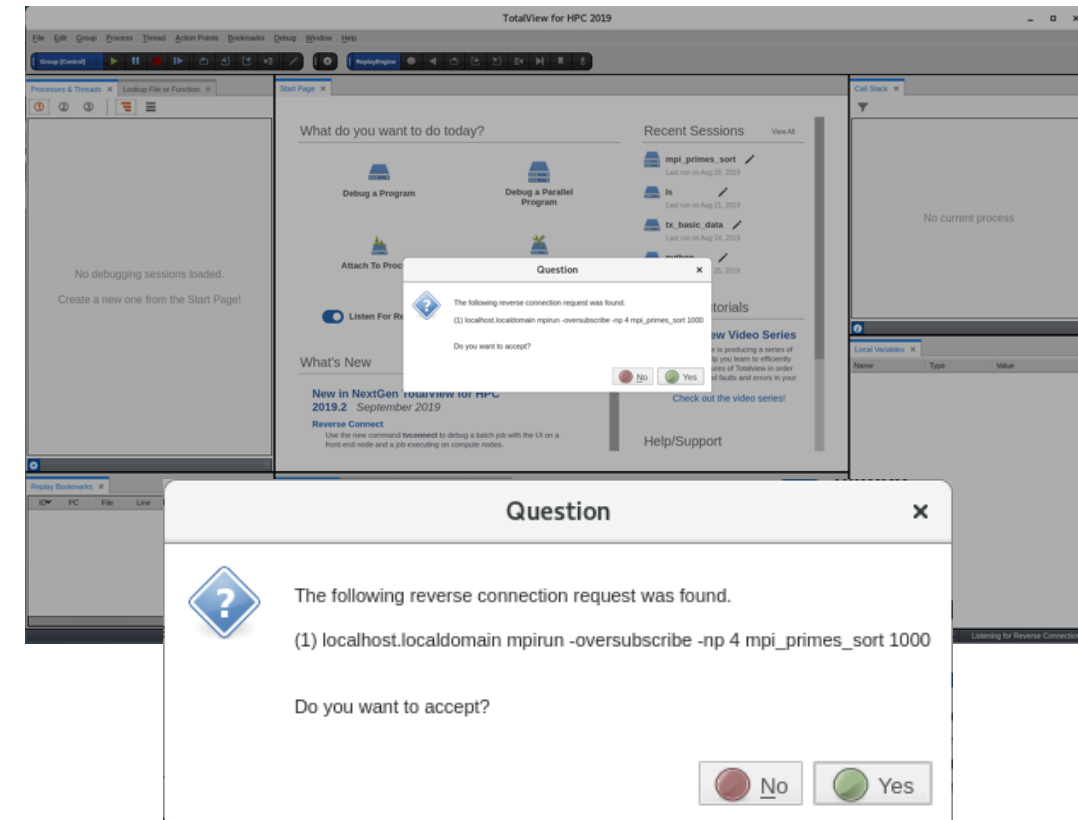
See it in action: <https://totalview.io/video-tutorials/how-use-remote-user-interface-debugging>

Reverse Debugging Connections

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
```



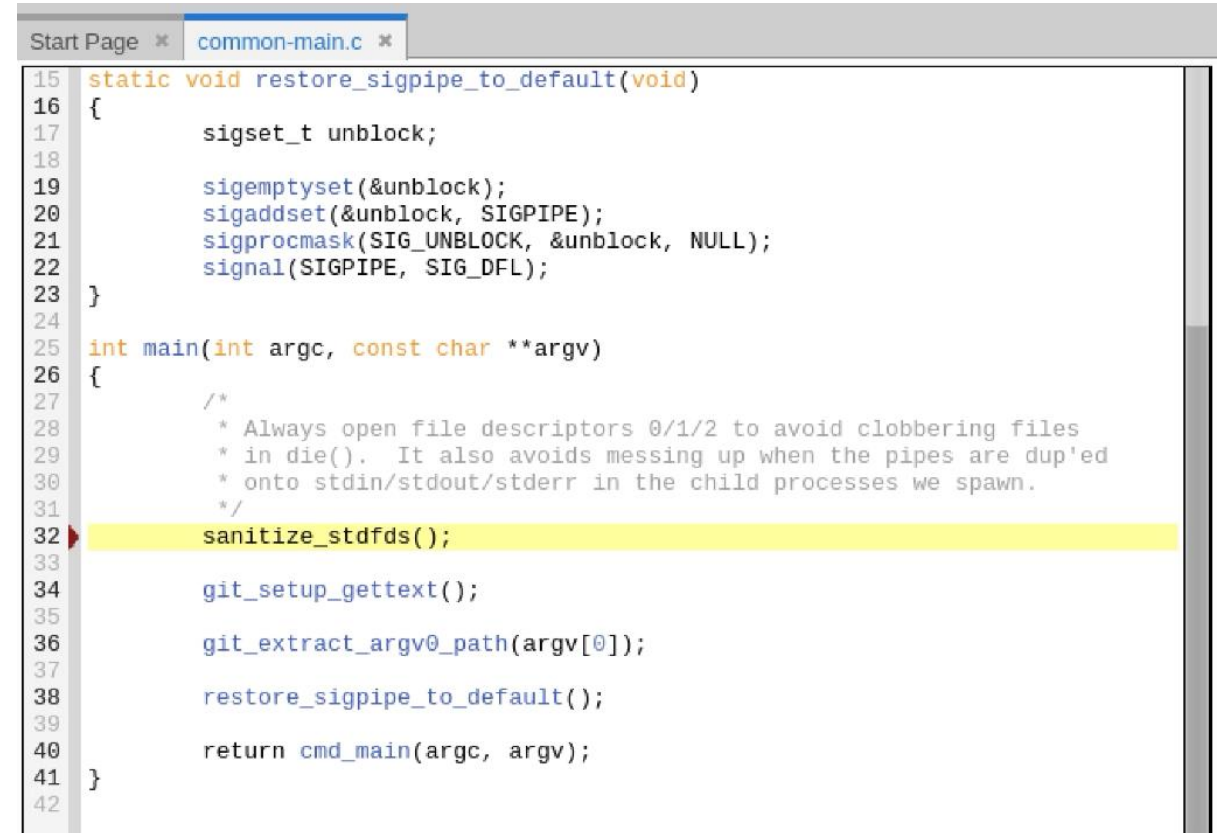
ANL Connect Demo

- TotalView Reverse Connect Demo
- TotalView Remote Debugging Demo

Reverse Debugging

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.

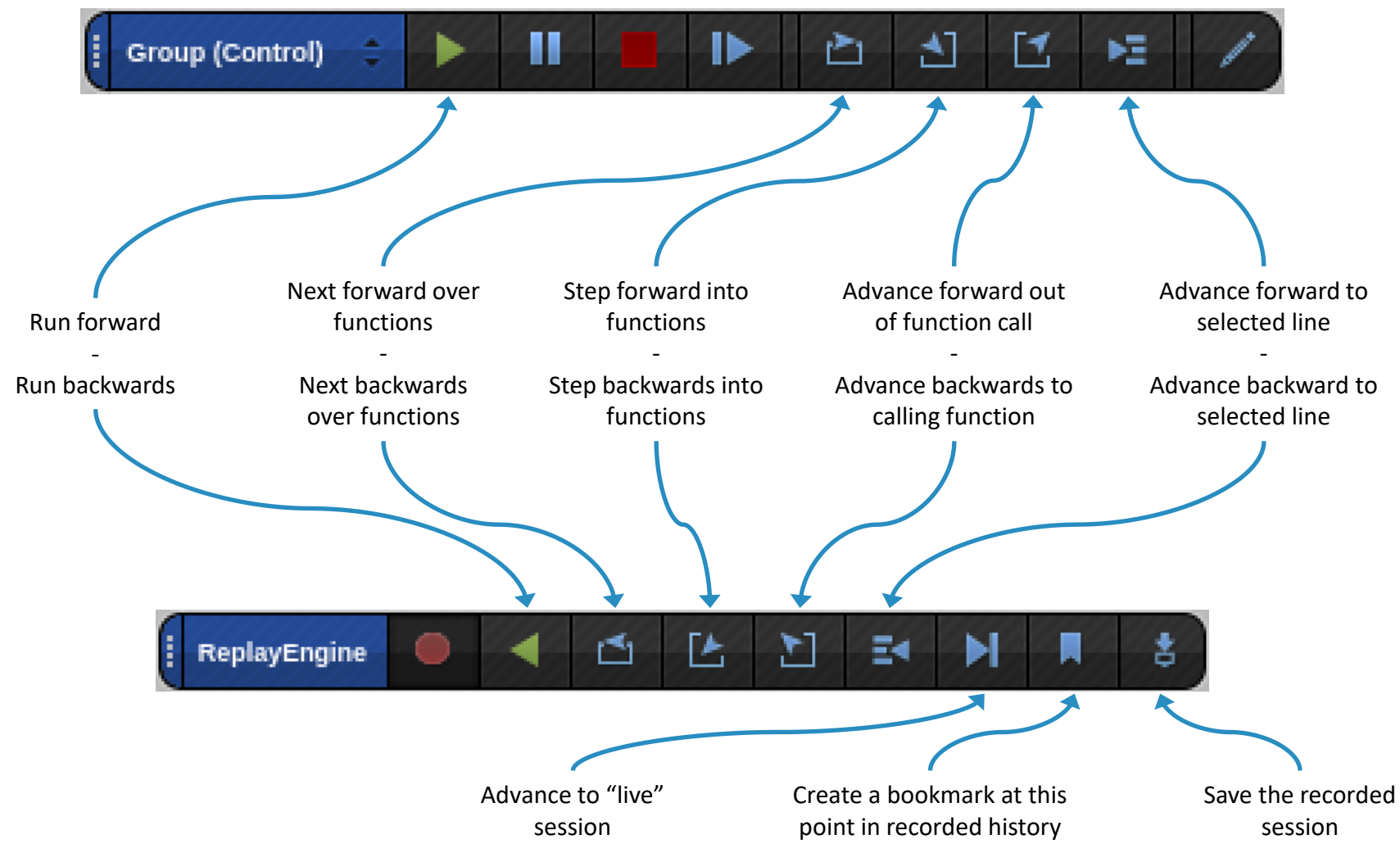


```
15 static void restore_sigpipe_to_default(void)
16 {
17     sigset_t unblock;
18
19     sigemptyset(&unblock);
20     sigaddset(&unblock, SIGPIPE);
21     sigprocmask(SIG_UNBLOCK, &unblock, NULL);
22     signal(SIGPIPE, SIG_DFL);
23 }
24
25 int main(int argc, const char **argv)
26 {
27     /*
28      * Always open file descriptors 0/1/2 to avoid clobbering files
29      * in die(). It also avoids messing up when the pipes are dup'ed
30      * onto stdin/stdout/stderr in the child processes we spawn.
31      */
32     sanitize_std fds();
33
34     git_setup_gettext();
35
36     git_extract_argv0_path(argv[0]);
37
38     restore_sigpipe_to_default();
39
40     return cmd_main(argc, argv);
41 }
42
```



See it in action: <https://totalview.io/video-tutorials/reverse-debugging>

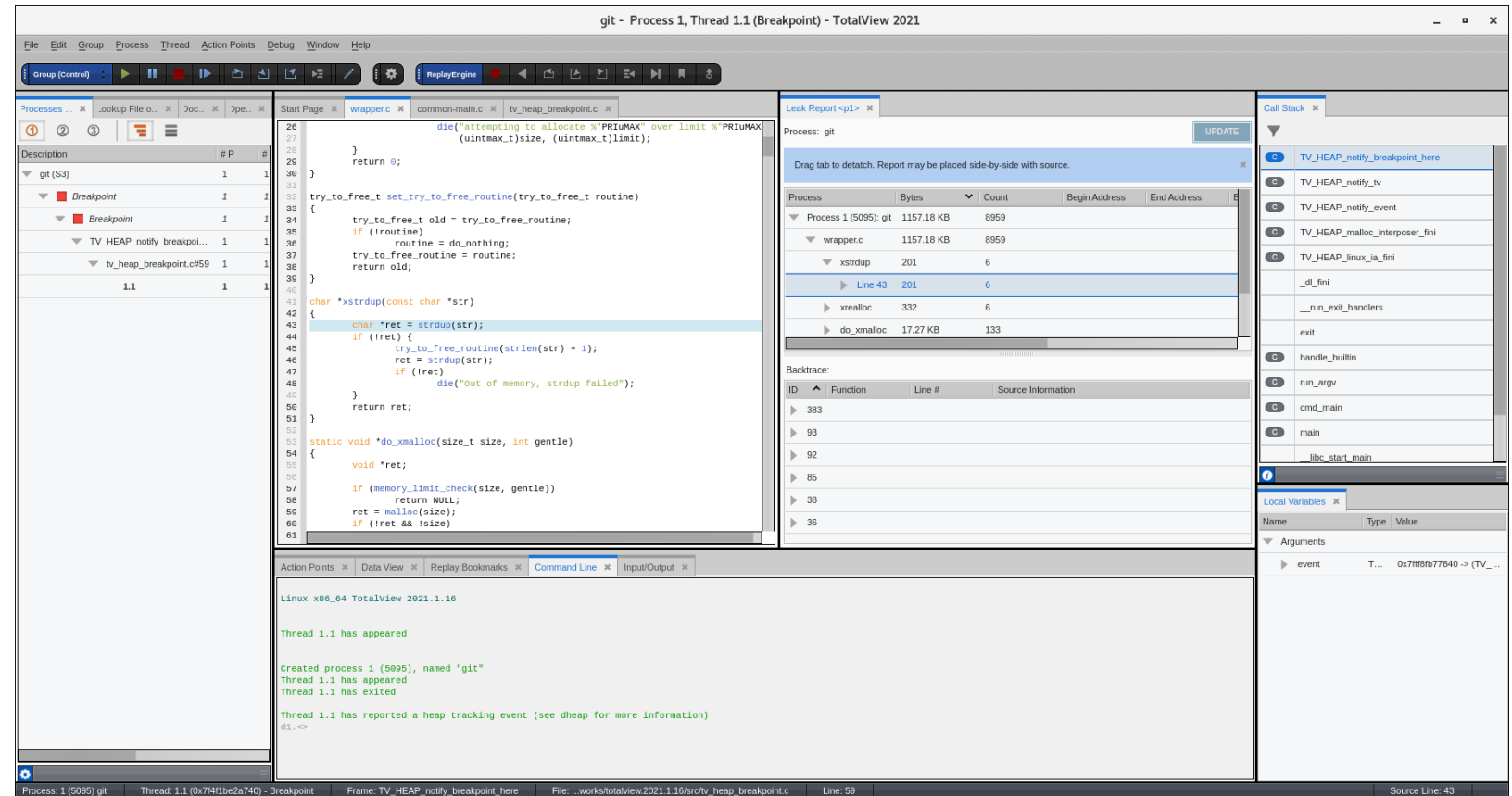
Reverse Debugging Controls



Memory Debugging

TotalView HPC Memory Debugging

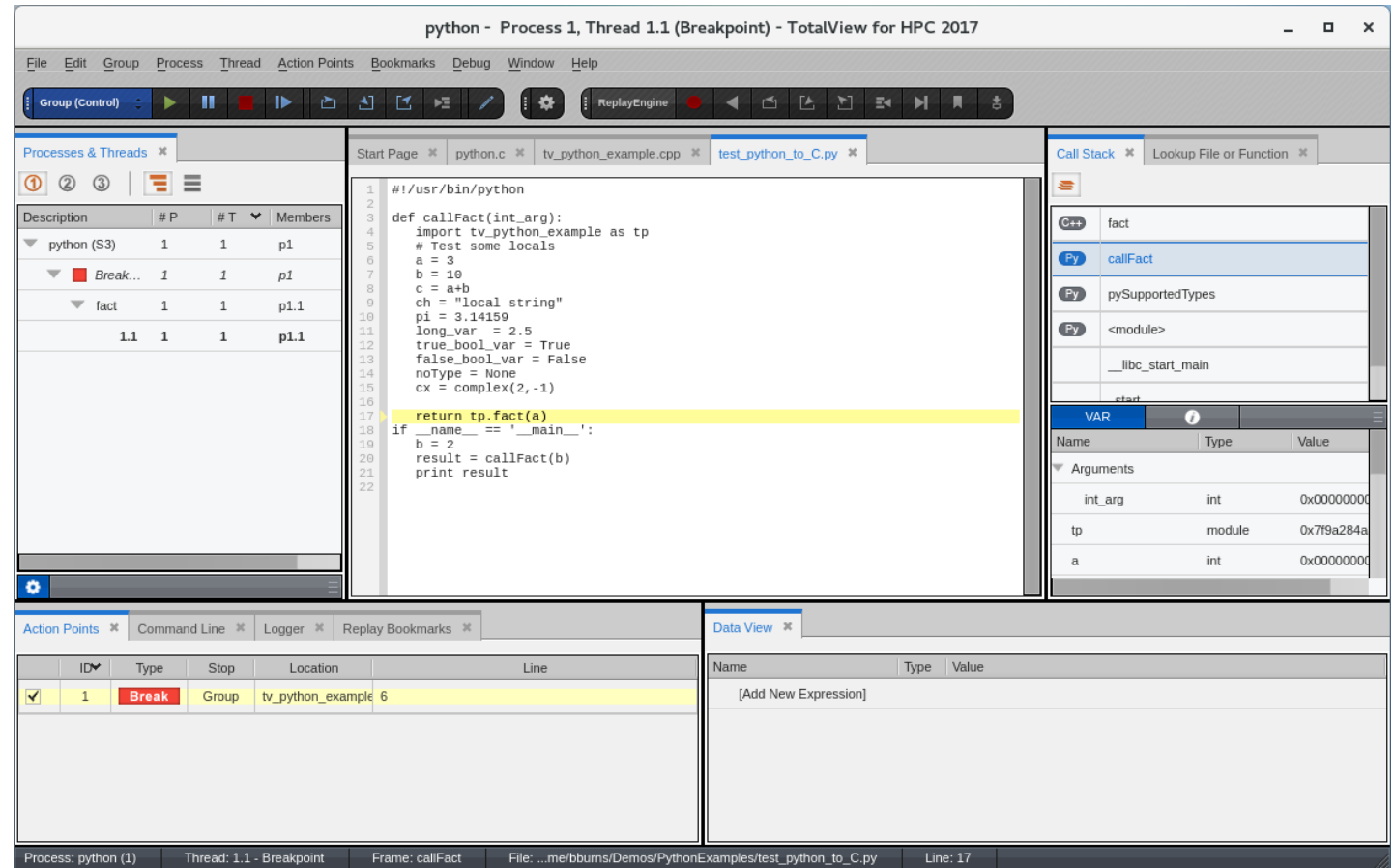
- Easily find memory leaks and other memory errors
- Understand heap usage
- Detect malloc/free new/delete API misuse
- Detect buffer overruns
- Understand where memory is being used
- Remote and MPI debugging



Python Debugging

Mixed Language Python Debugging

- 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.



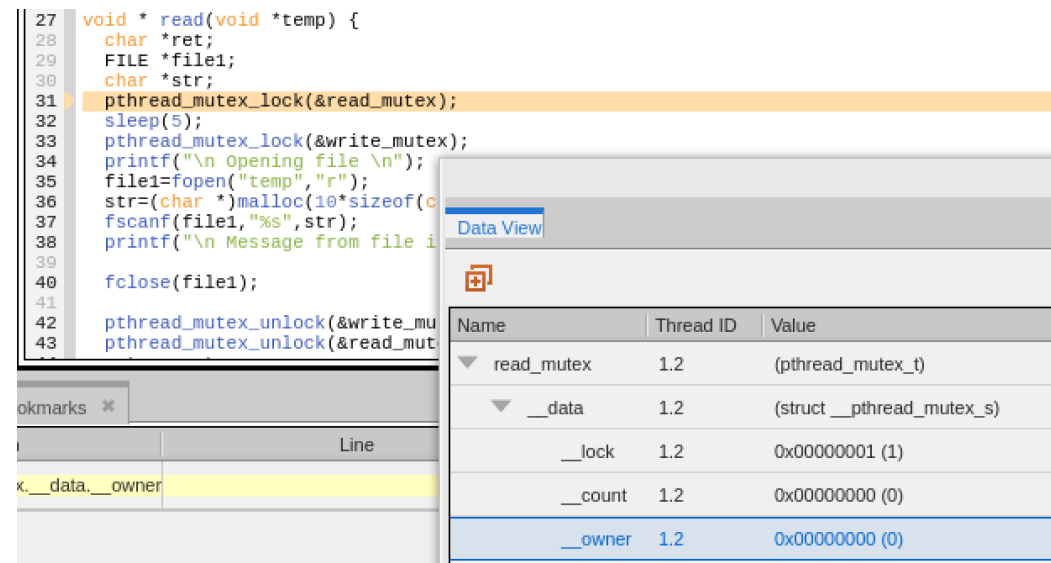
See it in action: <https://totalview.io/video-tutorials/debugging-python-and-c-mixed-language-applications>

A person wearing a cap and a dark shirt is seen from the side, working on a laptop in a server room. The background is filled with rows of server racks. The entire image has a blue tint, and a semi-transparent blue box is overlaid on the left side containing the title text.

Find Tough Bugs by Combing Debugging Technologies

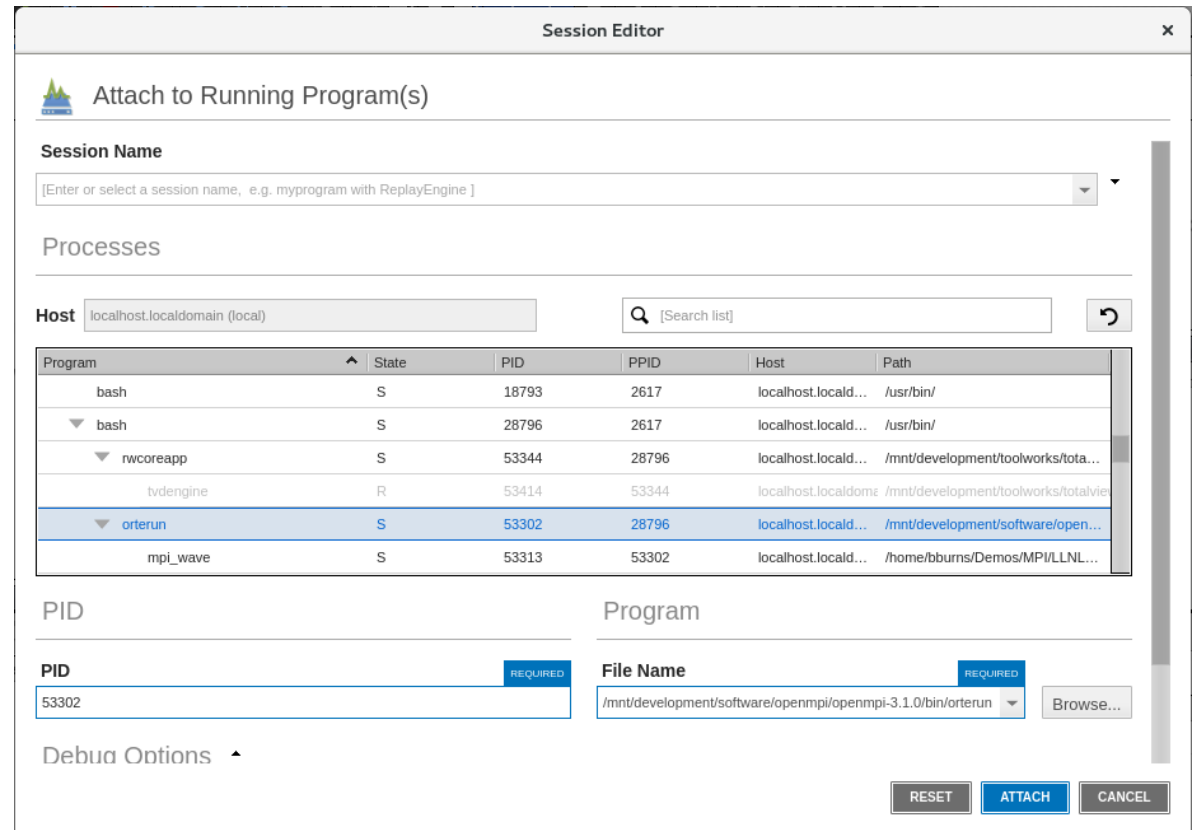
Combine Multiple Debugging Technologies

- Find where a mutex lock was acquired
 - Combine reverse debugging and watchpoints
 - Run backwards until pthread_mutex_t __owner changes
- Mix source code debugging, reverse debugging and memory debugging
 - Find memory allocations and leaks during your debugging session
- Use TotalView's Remote UI for efficient debugging using all TotalView's features from your laptop



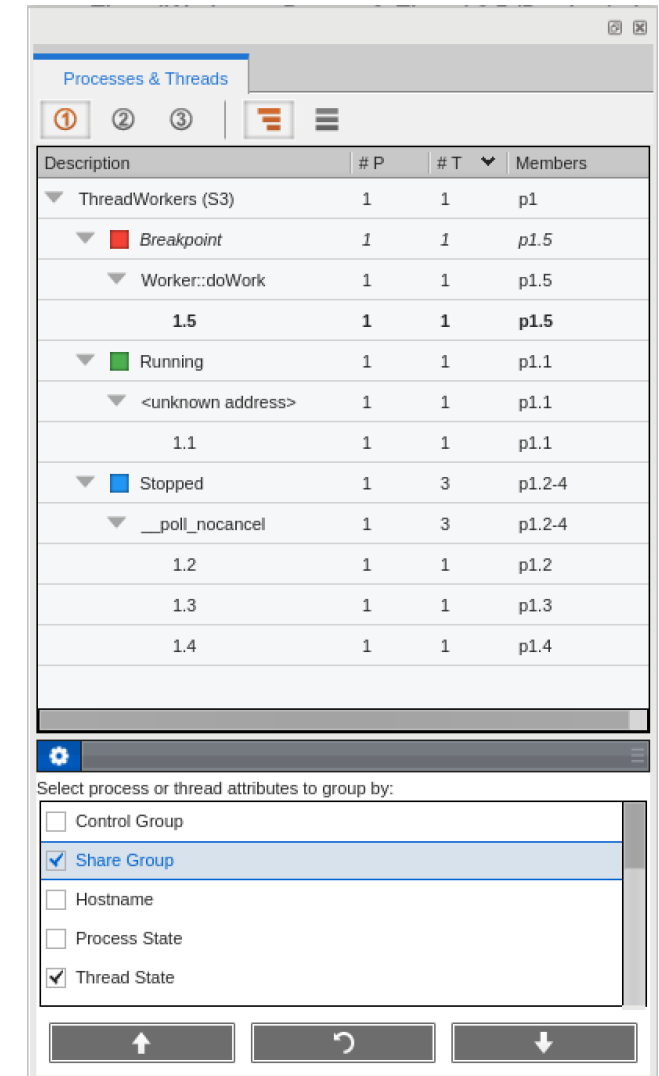
Attach and Detach from a Parallel Job

- Peek at the state of your parallel job
- Use TotalView's attach and detach capabilities to examine the job and then let it continue to run
- Attaching to starter process enables TotalView to discover and attach to all (or a subset) of the ranks



Process/Thread Aggregation

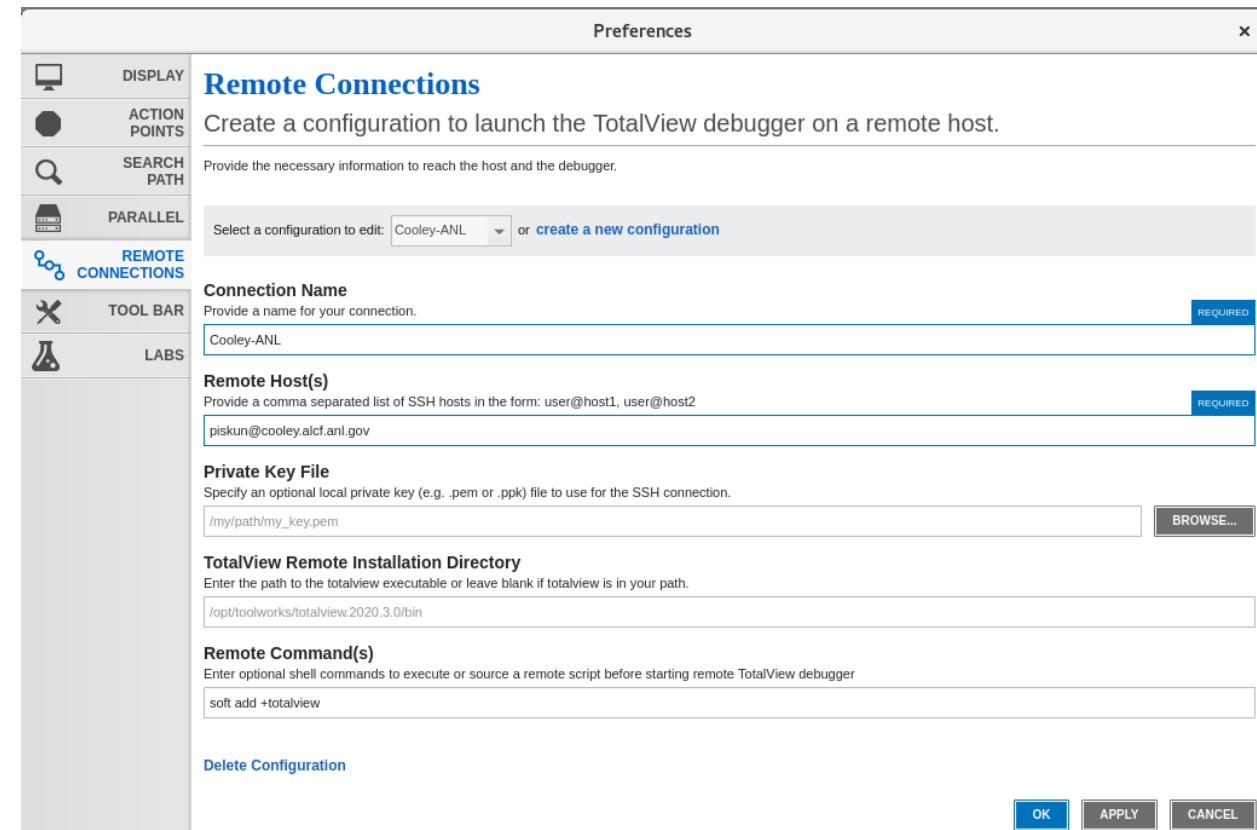
- Aggregate process and thread state to quickly understand the state of the job
- Find outliers quickly
- Views allow different configuration to be easily switched



Using TotalView for Parallel Debugging on ANL

TotalView remote debugging on Linux and Mac OS

- Download and install TotalView on your linux or mac. (ignore license)
 - `/grand/ATRESC2022/EXAMPLES/track-6-tools/TotalVlew/`
 - `www.totalview.io/downloads`
- Copy `/grand/ATRESC2022/EXAMPLES/track-6-tools/TotalView/2022.labs.tar.gz` to your area and untar it
- Run `make` to build examples.
- Connect to remote front node from the **terminal**
- Run labs remotely



The screenshot shows the 'Preferences' dialog box with the 'Remote Connections' tab selected. The left sidebar contains icons for DISPLAY, ACTION POINTS, SEARCH PATH, PARALLEL, REMOTE CONNECTIONS (highlighted), TOOL BAR, and LABS. The main area is titled 'Remote Connections' and contains the following fields:

- Connection Name:** A text field containing 'Cooley-ANL'. A 'REQUIRED' label is on the right.
- Remote Host(s):** A text field containing 'piskun@cooley.alcf.anl.gov'. A 'REQUIRED' label is on the right.
- Private Key File:** A text field containing '/my/path/my_key.pem' and a 'BROWSE...' button.
- TotalView Remote Installation Directory:** A text field containing '/opt/toolworks/totalview.2020.3.0/bin'.
- Remote Command(s):** A text field containing 'soft add +totalview'.

At the bottom right are 'OK', 'APPLY', and 'CANCEL' buttons. A 'Delete Configuration' link is at the bottom left of the main area.

TotalView is available on Theta, ThetaGPU and Cooley

- Installed at: `/soft/debuggers/totalview-2022-08-04/toolworks/totalview.2022.2.13/bin/totalview`
- Connect to Cooley (use `soft add +totalview` to setup Reverse Connect)
 - Get allocation first
 - `qsub -A ATPESC2022 -n 1 -q training -l`
 - `soft add +totalview`
 - `totalview -args aprun -np <N> ./demoMpi_v2 (*)`
 - `tvconnect aprun -np <N> ./demoMpi_v2 (*)`
 - (*) Supposed to work 😊
- Connect to Theta (use `module load totalview` to setup Reverse Connect)
 - `module swap PrgEnv-intel PrgEnv-cray ; module swap PrgEnv-cray PrgEnv-intel`
 - `setenv CRAYPE_LINK_TYPE dynamic`
 - Get allocation first
 - `qsub -A ATPESC2022 -n 4 -q debug-flat-quad -l`
 - `module load totalview totalview-support`
 - `totalview -args aprun -np <N> ./demoMpi_v2 (*)`
 - `tvconnect aprun -np <N> ./demoMpi_v2 (*)`
- Connect to ThetaGPU (use `module load totalview` to setup Reverse Connect)
 - Get allocation first
 - `qsub -A ATPESC2022 -n 1 -q single-gpu -l`
 - `module load totalview`

Hands-on labs

- Remotely connect to machine and enable Reverse Connection
- Copy `/grand/ATRESC2022/EXAMPLES/track-6-tools/TotalView/ATRESC2022-TV-labs.tar.gz`
- Programs are in `labs/programs/`

Labs:

- Lab 1 Debugger Basic
- Lab 2 Viewing, Examining, Watching and Editing Data
- Optional Lab 3 Examining and Controlling a Parallel Application (on Cooley)
 - Using remote connect (`tvconnect`)
 - `qsub -q training tvconnect.job`
 - Modify and submit `tvconnect.job` on your machine

Bonus lab: on thetaGPU: (`ssh -y thetagpusn1`)

- `qsub -l -n 1 -t 30 -q single-gpu -A ATPESC2022`
- `/usr/local/cuda/bin/nvcc -g -G tx_cuda_matmul.cu -o tx_cuda_matmul`
- `/grand/ATRESC2022/EXAMPLES/track-6-tools/TotalView/toolworks/totalview.2022.2.13/bin/tvconnect tx_cuda_matmul`

Remote submission of batch job.

- Submit job from TotalView (qsub 2022/labs/programs/tvconnect-thetaGPU.job)

- tvconnect-thetaGPU.job:

```
#!/bin/bash
```

```
#COBALT -t 30
```

```
#COBALT -n 4
```

```
#COBALT -q single-gpu
```

```
#COBALT -A ATPESC2022
```

```
module load totalview
```

```
tvconnect tx_cuda_matmul
```

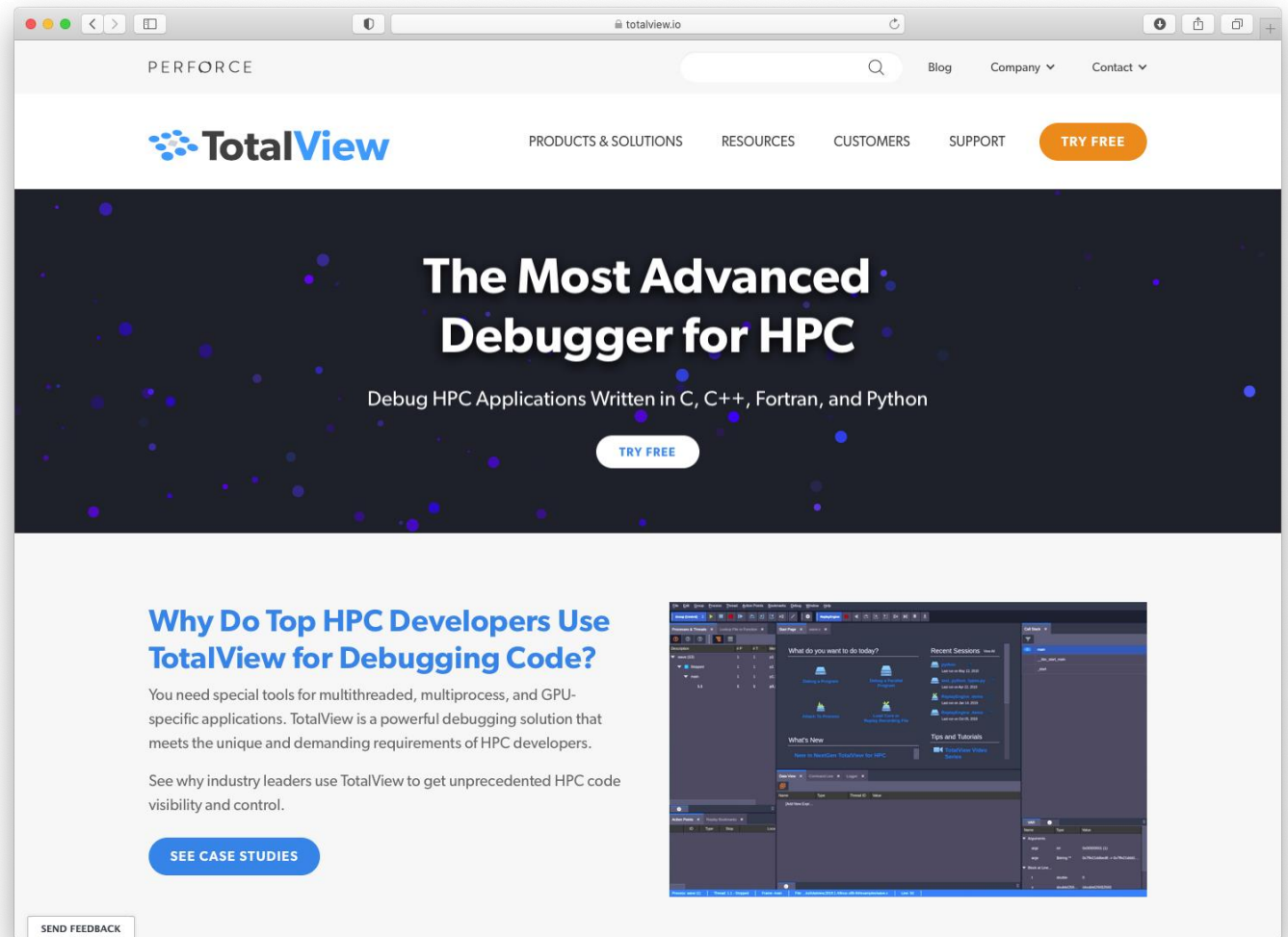
The screenshot shows the 'Remote Connections' configuration window in TotalView. The left sidebar contains a navigation menu with the following items: DISPLAY, ACTION POINTS, SEARCH PATH, PARALLEL, REMOTE CONNECTIONS (highlighted in blue), TOOL BAR, and LABS. The main content area is titled 'Remote Connections' and contains the following sections:

- Create a configuration to launch the TotalView debugger on a**
Provide the necessary information to reach the host and the debugger.
Select a configuration to edit: or [create a new configuration](#)
- Connection Name**
Provide a name for your connection.
- Remote Host(s)**
Provide a comma separated list of SSH hosts in the form: user@host1, user@host2
- Private Key File**
Specify an optional local private key (e.g. .pem or .ppk) file to use for the SSH connection.
- TotalView Remote Installation Directory**
Enter the path to the totalview executable or leave blank if totalview is in your path.
- Remote Command(s)**
Enter optional shell commands to execute or source a remote script before starting remote TotalView debugger

The 'Remote Command(s)' field and its label are circled in red.


TotalView Resources and Documentation

- TotalView website:
 - <https://totalview.io>
- TotalView documentation:
 - <https://help.totalview.io>
- TotalView Video Tutorials:
 - <https://totalview.io/support/video-tutorials>
- Other Resources:
 - Blog: <https://totalview.io/blog>



Summary

- Use of modern debugger **saves** you time.
- TotalView can help you because:
 - It's **cross-platform** (the only debugger you ever need)
 - Allow you to debug accelerators (GPU) and CPU in **one session**
 - Allow you to debug **multiple languages** (C++/Python/Fortran)



TotalView Resources and Documentation

TotalView Resources and Documentation

- TotalView website:
<https://totalview.io>
- TotalView documentation:
 - <https://help.totalview.io>
 - 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>





Q&A

Questions

- Any questions or comments?
 - Don't hesitate to reach out to me directly with any questions or comments!
 - **Email:** npiskun@perforce.com
- **Thank you for your time today!**