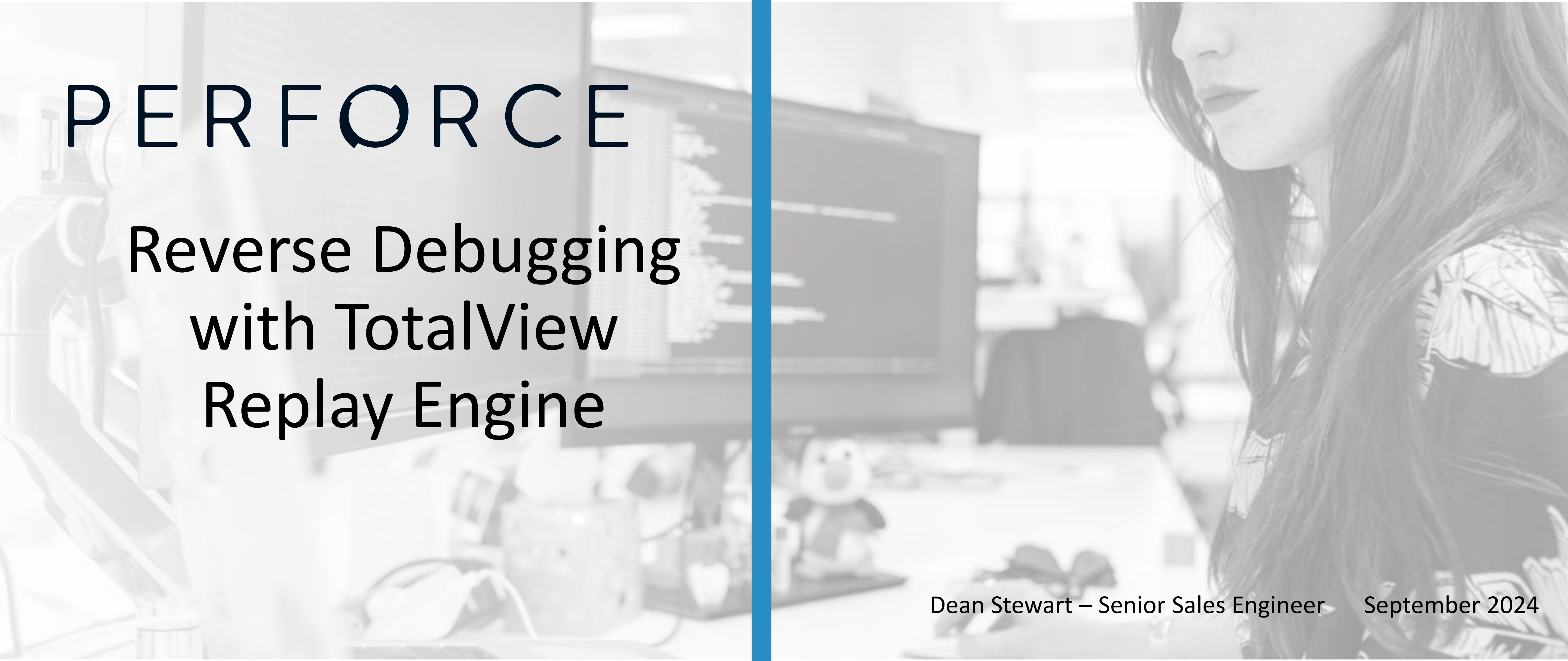


PERFORCE

Reverse Debugging with TotalView Replay Engine

Dean Stewart – Senior Sales Engineer September 2024

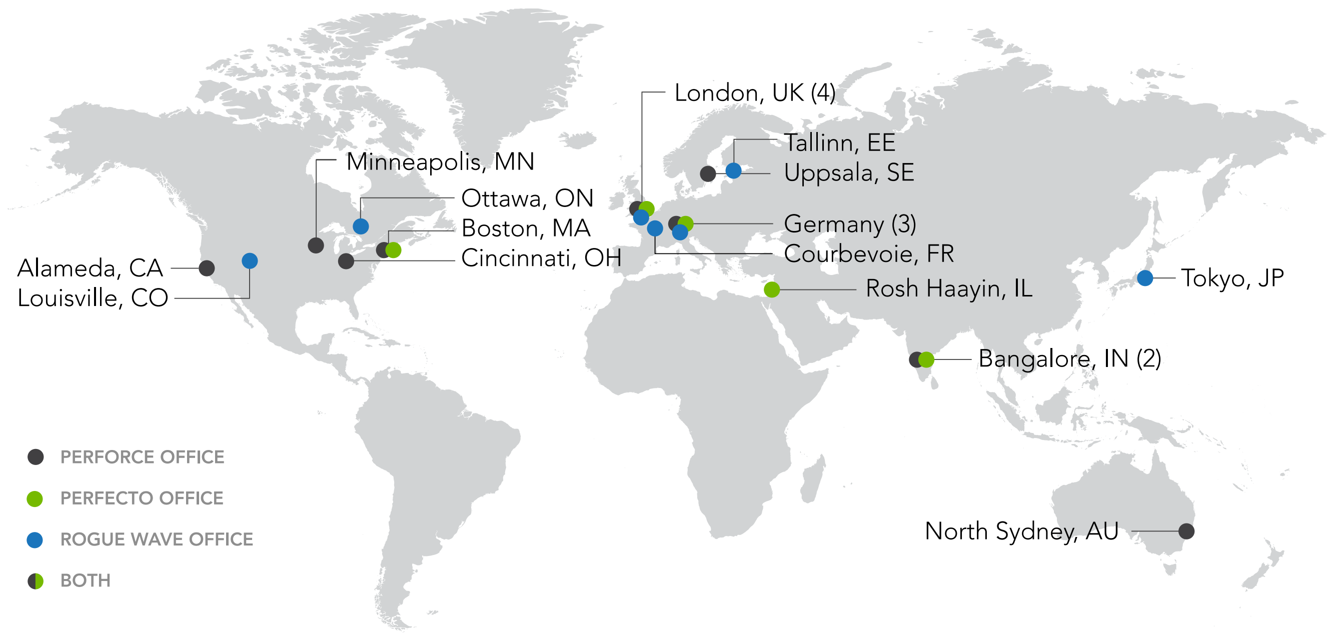
Agenda

- Introduction to Perforce
- TotalView's Features
- Reverse debugging with ReplayEngine
- Demo
- Resources
- Q&A

Introduction to Perforce

Perforce Global Footprint

- Customers in 80 countries
- 9,000 customers worldwide
- More than 250 of the Fortune 500
- Customers deploying multiple products
- 25+ offices and 4 data centers which give us global reach
- Over 1,400 employees in 25 countries



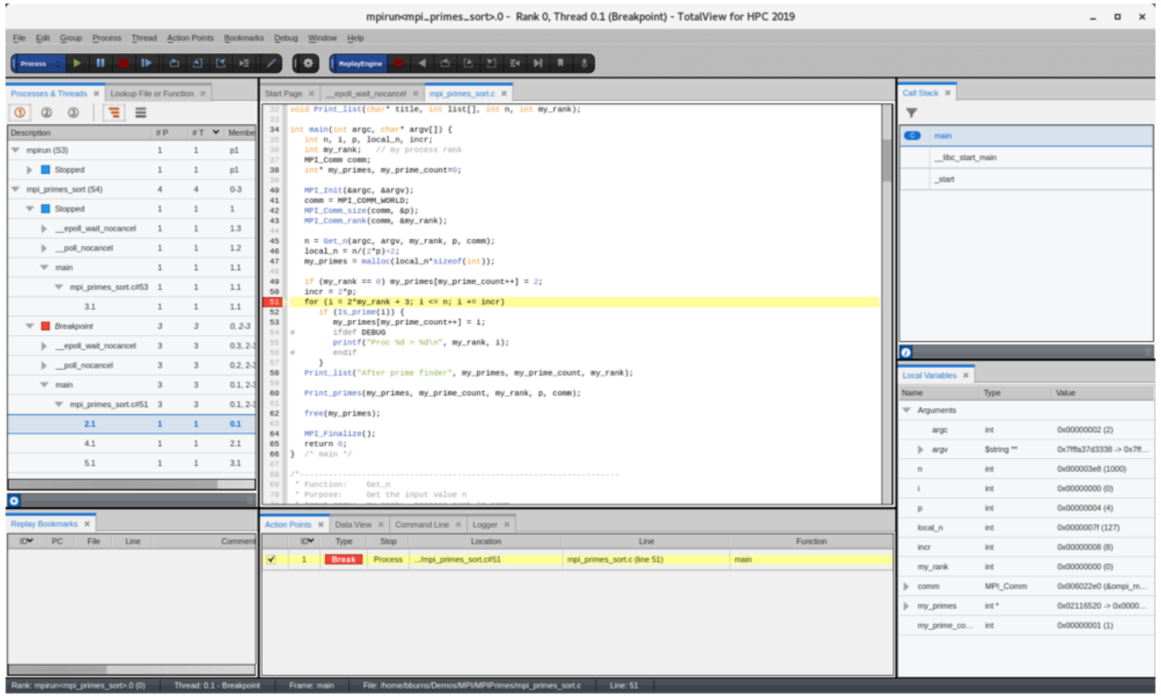
Perforce Products

Agile Management	Code Management & Collaboration	Application Mgmt. & Components	Automated Testing
Helix ALM	Helix Core	Puppet	Helix QAC
Hansoft	Methodics	Akana	Klocwork
Gliffy	Helix4Git	OpenLogic	Perfecto
	JRebel	Zend	BlazeMeter
	TotalView	Visualization	
		SourcePro	
		IMSL	

TotalView’s Features

HPC Debugging With TotalView

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



Description	# P	# T	Membe
mpirun (S3)	1	1	p1
Stopped	1	1	p1
mpi_primes_sort (S4)	4	4	0-3
Stopped	1	1	1
__epoll_wait_nocancel	1	1	1.3
__poll_nocancel	1	1	1.2
main	1	1	1.1
mpi_primes_sort.c#53	1	1	1.1
3.1	1	1	1.1
Breakpoint	*3*	*3*	*0, 2-3*
__epoll_wait_nocancel	3	3	0.3, 2-3
__poll_nocancel	3	3	0.2, 2-3
main	3	3	0.1, 2-3
mpi_primes_sort.c#51	3	3	0.1, 2-3
2.1	1	1	0.1
4.1	1	1	2.1
5.1	1	1	3.1

```
void Print_list(char* title, int list[], int n, int my_rank);

int main(int argc, char* argv[]) {
   int n, i, p, local_n, incr;
   int my_rank;   // my process rank
   MPI_Comm comm;
   int* my_primes, my_prime_count=0;

   MPI_Init(&argc, &argv);
   comm = MPI_COMM_WORLD;
   MPI_Comm_size(comm, &p);
   MPI_Comm_rank(comm, &my_rank);

   n = Get_n(argc, argv, my_rank, p, comm);
   local_n = n/(2*p)+2;
   my_primes = malloc(local_n*sizeof(int));

   if (my_rank == 0) my_primes[my_prime_count++] = 2;
   incr = 2*p;
   for (i = 2*my_rank + 3; i <= n; i += incr)
      if (Is_prime(i)) {
         my_primes[my_prime_count++] = i;
#        ifdef DEBUG
         printf("Proc %d > %d\n", my_rank, i);
#        endif
      }
   Print_list("After prime finder", my_primes, my_prime_count, my_rank);

   Print_primes(my_primes, my_prime_count, my_rank, p, comm);

   free(my_primes);

   MPI_Finalize();
   return 0;
}  /* main */

/*-------------------------------------------------------------------
 * Function:    Get_n
 * Purpose:     Get the input value n
```

Call Stack
main
__libc_start_main
_start

ID	PC	File	Line	Comment

	ID	Type	Stop	Location	Line	Function
✓	1	Break	Process	../mpi_primes_sort.c#51	mpi_primes_sort.c (line 51)	main

Name	Type	Value
Arguments		
argc	int	0x00000002 (2)
argv	$string **	0x7fffa37d3338 -> 0x7ff...
n	int	0x000003e8 (1000)
i	int	0x00000000 (0)
p	int	0x00000004 (4)
local_n	int	0x0000007f (127)
incr	int	0x00000008 (8)
my_rank	int	0x00000000 (0)
comm	MPI_Comm	0x006022e0 (&ompi_m...
my_primes	int *	0x02116520 -> 0x0000...
my_prime_co...	int	0x00000001 (1)

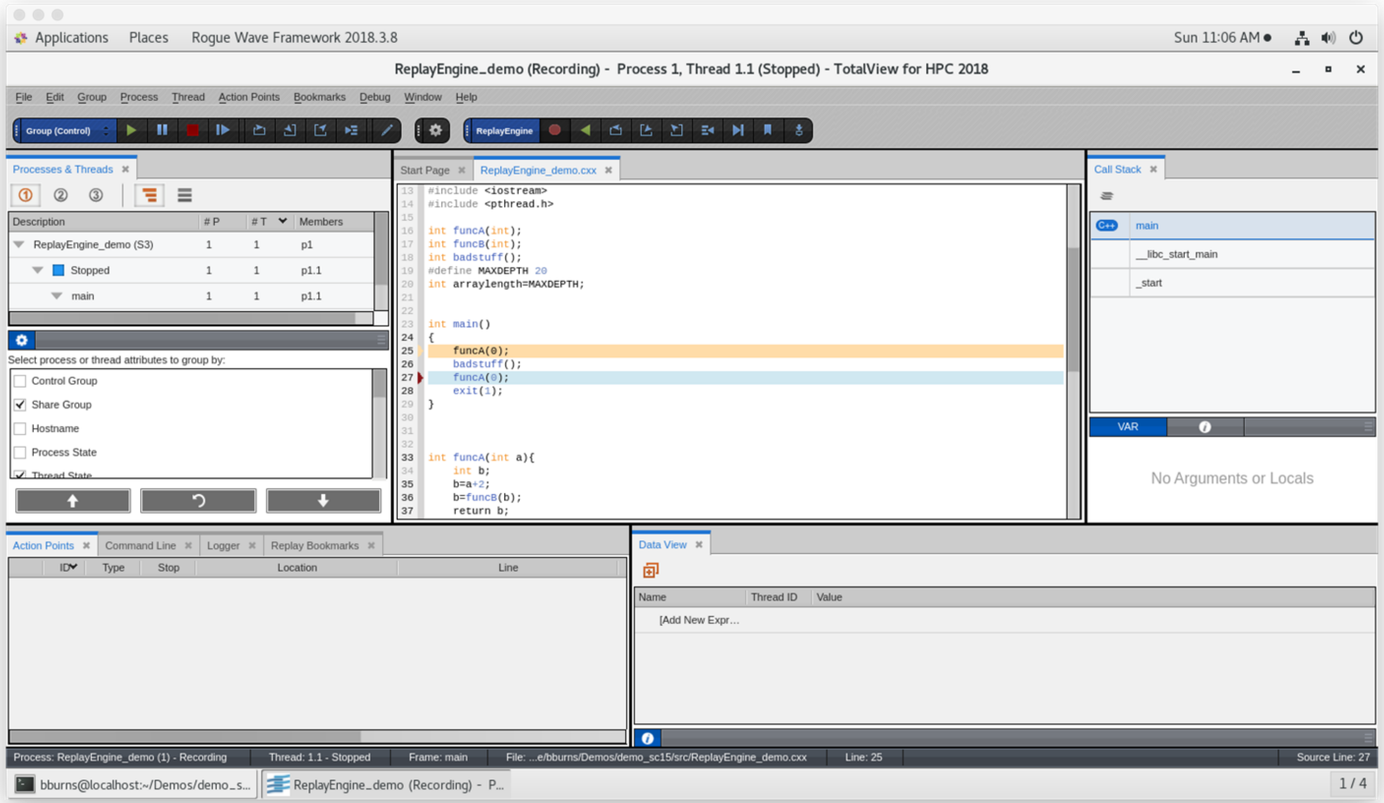
Rank: mpirun<mpi_primes_sort> 0 (0) | Thread: 0.1 - Breakpoint | Frame: main | File: /home/bburns/Demos/MPI/MPIPrimes/mpi_primes_sort.c | Line: 51

LANGUAGES	OPERATING SYSTEMS	APPLICATIONS	PLATFORMS
Ftn		MPI, NVIDIA CUDA, OpenMP, OpenACC	intel, NVIDIA, arm, AMD, Hewlett Packard Enterprise

Reverse Debugging with ReplayEngine

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.



Description	# P	# T	Members
ReplayEngine_demo (S3)	1	1	p1
Stopped	1	1	p1.1
main	1	1	p1.1

Select process or thread attributes to group by:

- [ ] Control Group
- [✓] Share Group
- [ ] Hostname
- [ ] Process State
- [✓] Thread State

```
#include <iostream>
#include <pthread.h>

int funcA(int);
int funcB(int);
int badstuff();
#define MAXDEPTH 20
int arraylength=MAXDEPTH;


int main()
{
    funcA(0);
    badstuff();
    funcA(0);
    exit(1);
}



int funcA(int a){
    int b;
    b=a+2;
    b=funcB(b);
    return b;
```

ID	Type	Stop	Location	Line

Name	Thread ID	Value
[Add New Expr…		

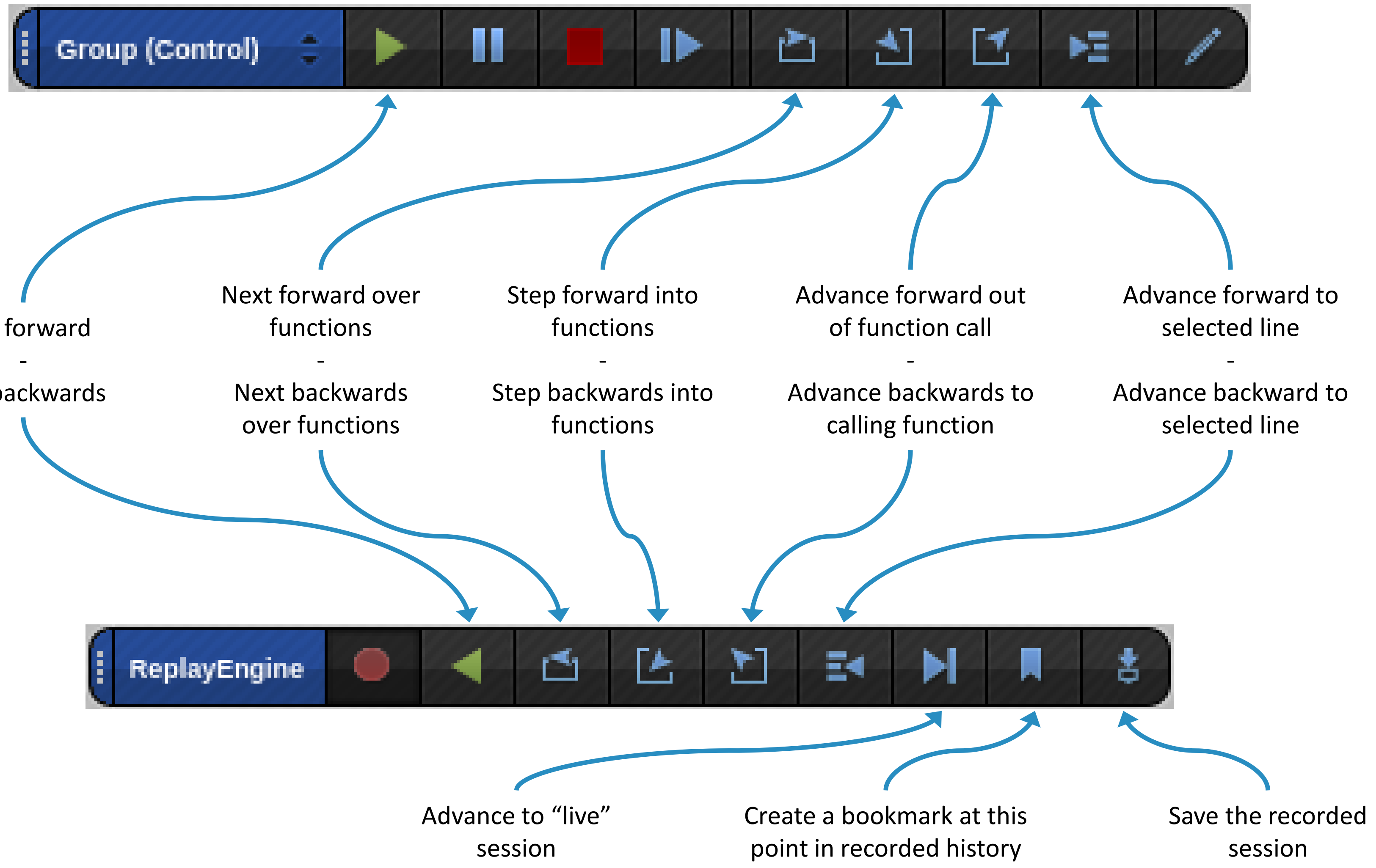
```
Process: ReplayEngine_demo (1) - Recording    Thread: 1.1 - Stopped    Frame: main    File: ...e/bburns/Demos/demo_sc15/src/ReplayEngine_demo.cxx    Line: 25    Source Line: 27
```

bburns@localhost:~/Demos/demo_s...   ReplayEngine_demo (Recording) - P...   1 / 4

Recording and Playback

- When ReplayEngine is saving state information, it is in **Record Mode**
- The saved state information is the program's execution history
- You can save the execution history at any time and reload the recording when debugging the executable in a subsequent session
- Using a ReplayEngine command, ether from the Toolbar or the CLI, shifts ReplayEngine into **ReplayMode**
- Debugging commands that do not work in ReplayMode include:
  - Changing a variable's value
  - Functions that alter memory
  - Running threads asynchronously

Reverse Debugging Controls

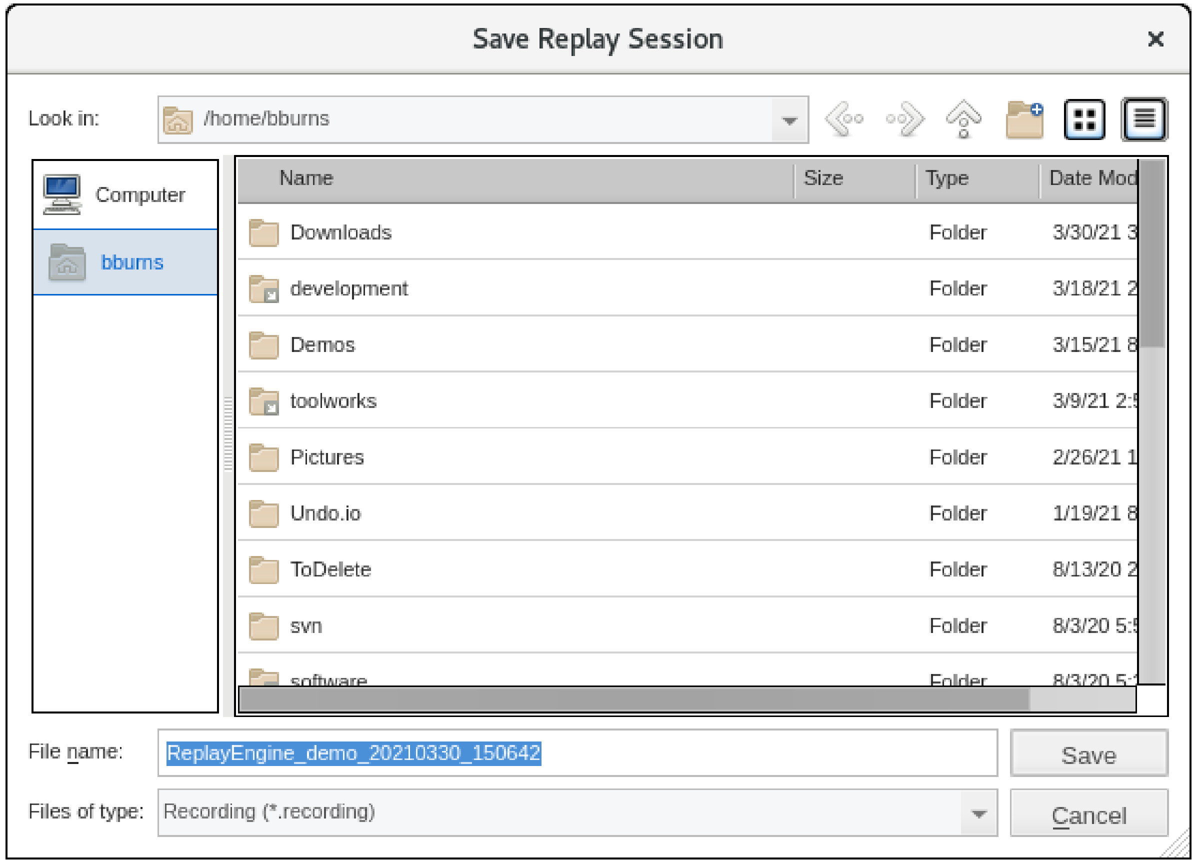

Group (Control)
ReplayEngine
Run forward
-
Run backwards
Next forward over functions
-
Next backwards over functions
Step forward into functions
-
Step backwards into functions
Advance forward out of function call
-
Advance backwards to calling function
Advance forward to selected line
-
Advance backward to selected line
Advance to "live" session
Create a bookmark at this point in recorded history
Save the recorded session

Saving and Loading Execution History

- TotalView can save the current ReplayEngine execution history to file at any time

- The saved recording can be loaded into TotalView using any of the following:
  - At startup, using the same syntax as when opening a core file:

    totalview *executable recording-file*
  - On the Start Page view by selecting Load Core File or Replay Recording File



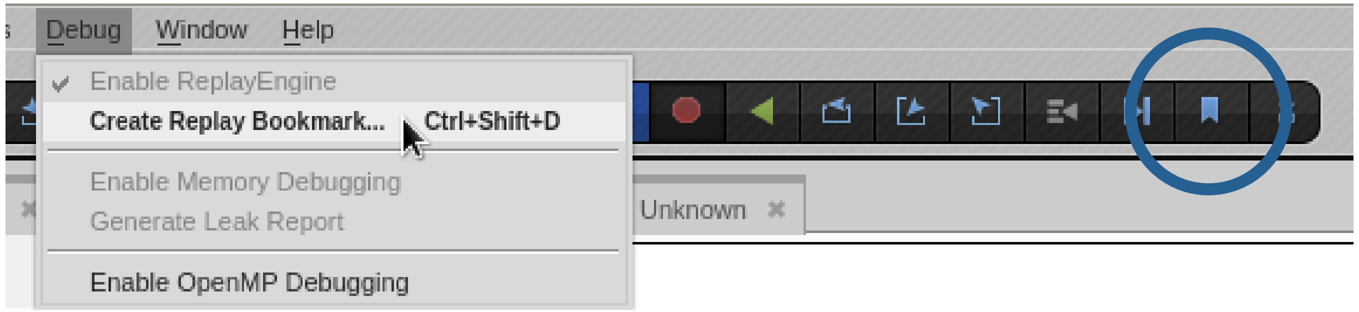
Name	Size	Type	Date Mod
Downloads		Folder	3/30/21 3
development		Folder	3/18/21 2
Demos		Folder	3/15/21 8
toolworks		Folder	3/9/21 2:5
Pictures		Folder	2/26/21 1
Undo.io		Folder	1/19/21 8
ToDelete		Folder	8/13/20 2
svn		Folder	8/3/20 5:5
software		Folder	8/3/20 5:1

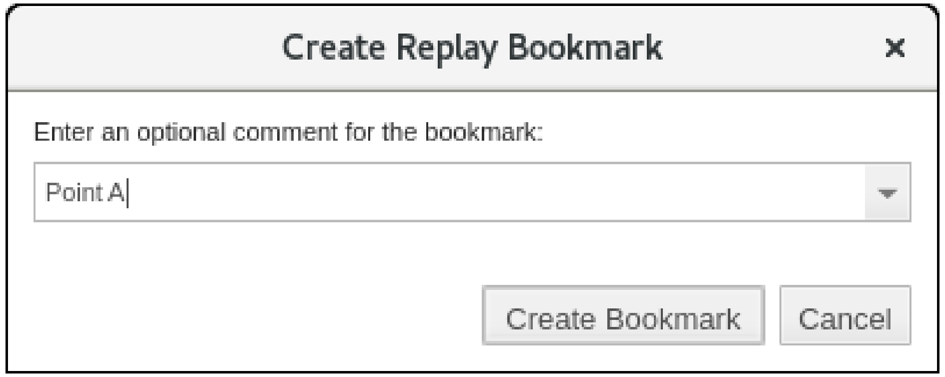
File name: ReplayEngine_demo_20210330_150642

Files of type: Recording (*.recording)

Replay Bookmarks

- Replay bookmarks mark a point in the execution of a program, allowing you to quickly jump back to that point in time

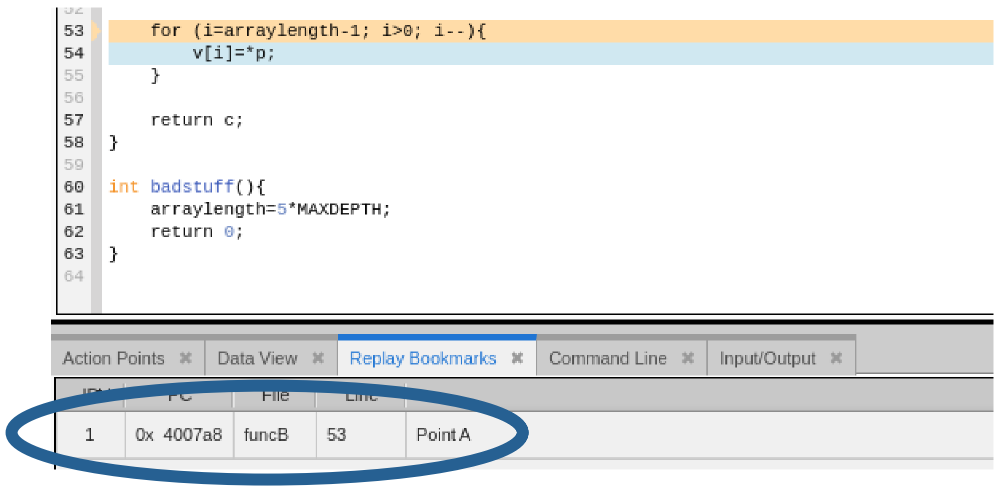




Enter an optional comment for the bookmark:

Point A

Creating a Replay Bookmark



```
53      for (i=arraylength-1; i>0; i--){
54          v[i]=*p;
55      }
56
57      return c;
58  }
59
60  int badstuff(){
61      arraylength=5*MAXDEPTH;
62      return 0;
63  }
64
```

ID	PC	File	Line	
1	0x 4007a8	funcB	53	Point A

Activating a Replay Bookmark

Setting Preferences for ReplayEngine

- You can set the following preferences for ReplayEngine
  - the maximum amount of memory to allocate to ReplayEngine
  - The preferred behaviour when the memory limit is reached

- Setting the maximum amount of memory. The default value ‘0’ specifies to limit the maximum size by available memory only.

dset TV::replay_history_size *value*

e.g. dset TV::replay_history_size 1024M

- Setting the preferred behaviour. By default, the oldest history is discarded so that recording can continue

dset TV::replay_history_mode **1** (Discard oldest history and continue recording)

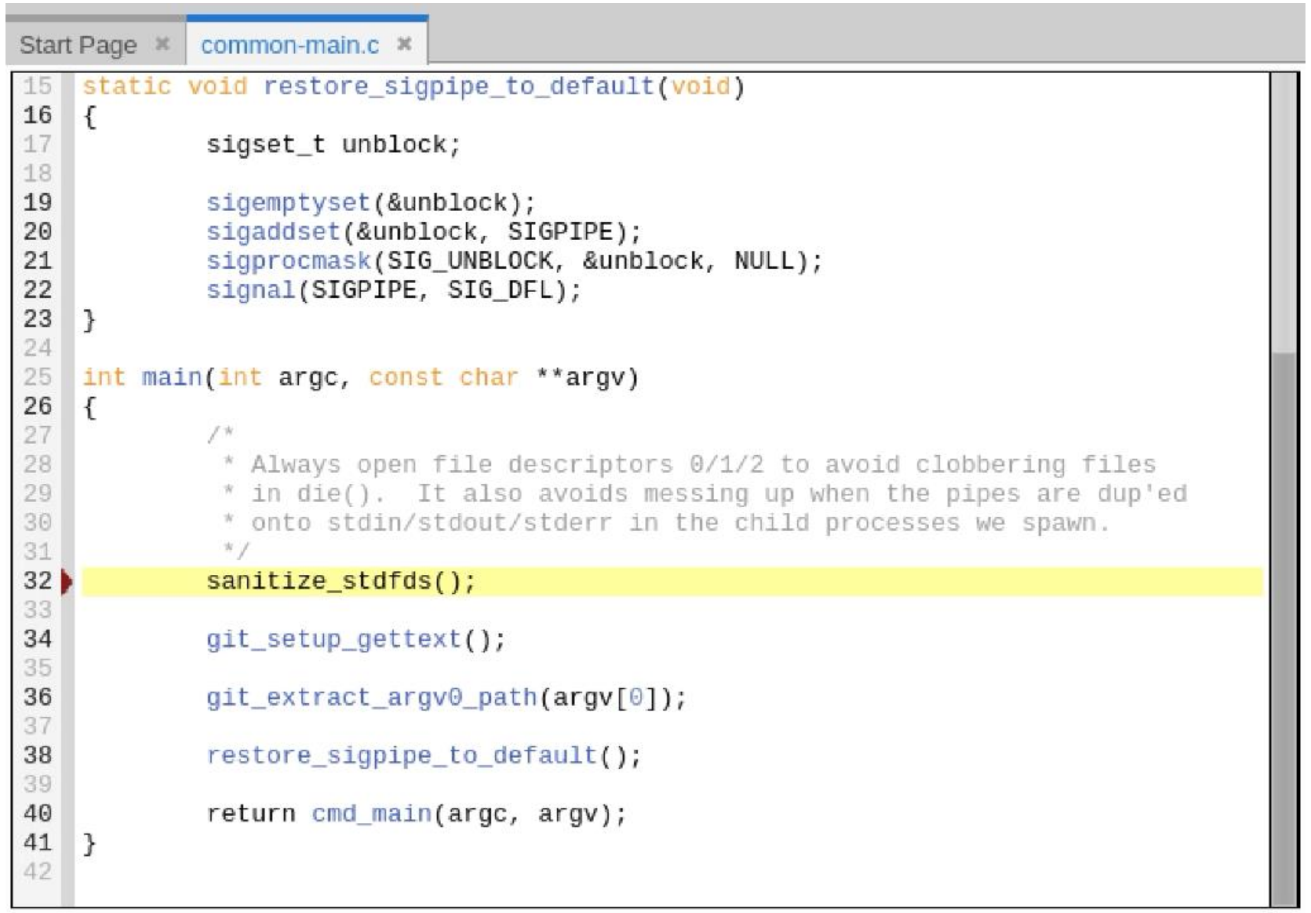
dset TV::replay_history_mode **2** (Stop the process when the buffer is full)

Benefits of ReplayEngine

- Time saved during development
- Finding and fixing intermittent bugs
- Helps with learning and understanding new code
- Supports continuous integration and collaboration
- Ease of use

Demo

Replay Engine Demo

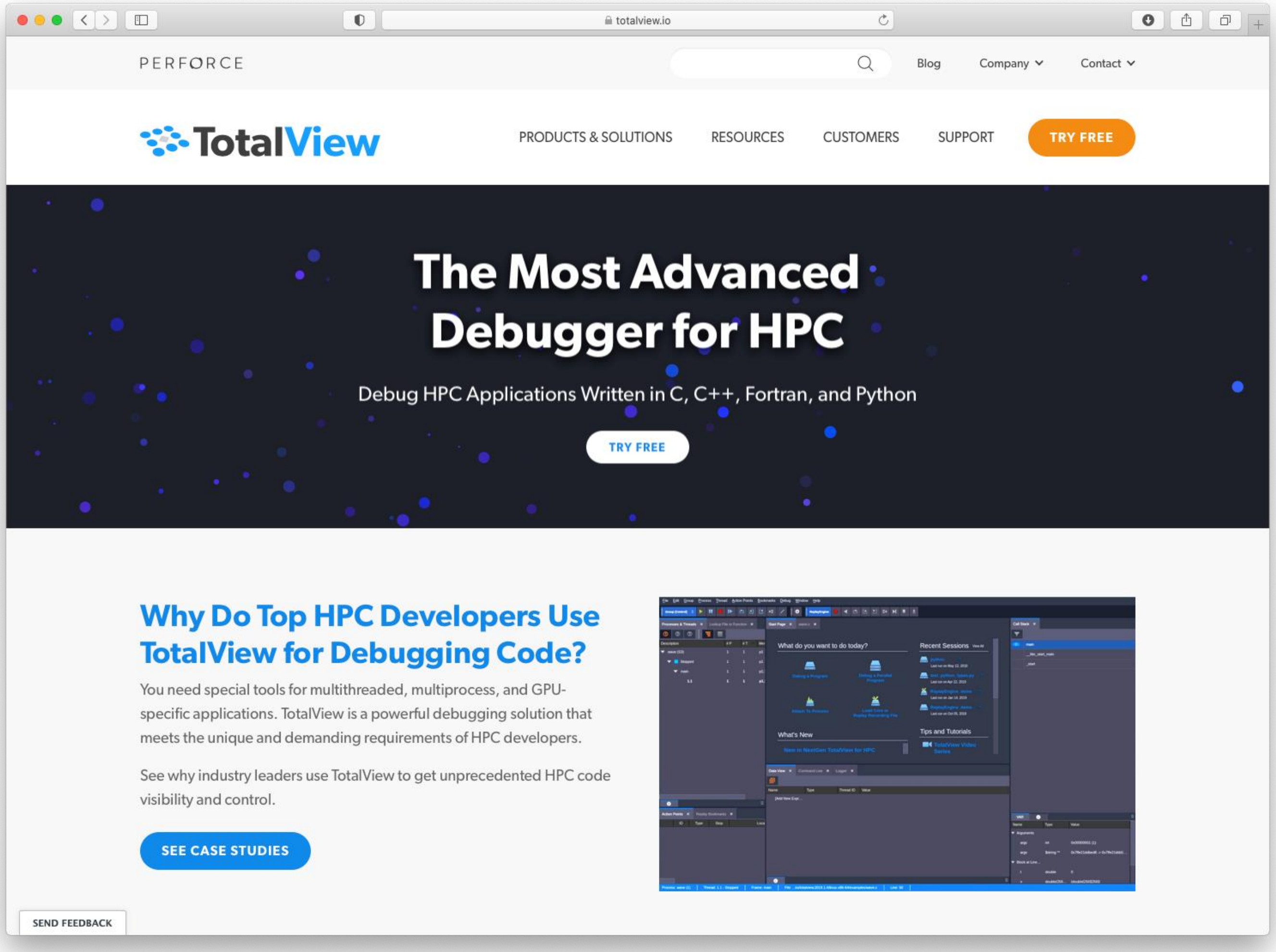


```
static void restore_sigpipe_to_default(void)
{
	sigset_t unblock;

	sigemptyset(&unblock);
	sigaddset(&unblock, SIGPIPE);
	sigprocmask(SIG_UNBLOCK, &unblock, NULL);
	signal(SIGPIPE, SIG_DFL);
}

int main(int argc, const char **argv)
{
	/*
	 * Always open file descriptors 0/1/2 to avoid clobbering files
	 * in die().  It also avoids messing up when the pipes are dup'ed
	 * onto stdin/stdout/stderr in the child processes we spawn.
	 */
	sanitize_stdfds();

	git_setup_gettext();

	git_extract_argv0_path(argv[0]);

	restore_sigpipe_to_default();

	return cmd_main(argc, argv);
}
```

Resources

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



totalview.io

PERFORCE Blog Company Contact

TotalView PRODUCTS & SOLUTIONS RESOURCES CUSTOMERS SUPPORT TRY FREE

The Most Advanced Debugger for HPC

Debug HPC Applications Written in C, C++, Fortran, and Python

TRY FREE

Why Do Top HPC Developers Use TotalView for Debugging Code?

You need special tools for multithreaded, multiprocess, and GPU-specific applications. TotalView is a powerful debugging solution that meets the unique and demanding requirements of HPC developers.

See why industry leaders use TotalView to get unprecedented HPC code visibility and control.

SEE CASE STUDIES

SEND FEEDBACK

Q&A

PERFORCE

Thank You

Contact Info

Susanne Horn

susanne.horn@smb-net.de

Dean Stewart

dstewart@perforce.com