TotalView

Advanced Parallel Debugging with TotalView

Nikolay Piskun Software Architect, TotalView products, Perforce Software August 2022

Agenda

HPC Debugging and Dynamic Analysis with TotalView



6

Combining HPC Debugging Technologies



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

Processes & Threads × Lookup Fi	le or Fun	ction ×			epol_wat_nocancel X mpi_primes_sort.c X	Call Stack ×		
0 2 3 =				33	<pre>int_list(char* title, int list[], int n, int my_rank);</pre>	T		
Description	# P	# T	✓ Membe	34	<pre>in(int argc, char* argv[]) { n, i, p, local_n, incr; }</pre>	C main		
▼ mpirun (S3)	1	1	p1	36	my_rank; // my process rank Comm comm;	_libc_star	main	
Stopped	1	1	p1	38	'my_primes, my_prime_count=0;			
▼ mpi_primes_sort (S4)	4	4	0.3	40	Init(&argc, &argv);	_start		
Topped	1	1	1	41 42	<pre>n = MPI_COMM_WORLD; Comm_size(comm, &p);</pre>			
_epoll_wait_nocancel	1	1	1.3	43	<pre>_comm_rank(comm, &my_rank);</pre>			
poll_nocancel	1	1	1.2	45	<pre>Get_n(argc, argv, my_rank, p, comm); l = = n/(270)/21</pre>			
▼ main	1	1	1.1	47	<pre>hl_n = n/(2*p)+2; rimes = malloc(local_n*sizeof(int));</pre>			
▼ mpi_primes_sort.c453		1	1.1	40	<pre>my_rank == 0) my_primes[my_prime_count++] = 2;</pre>			
* mp_prines_sor.co.s. 3.1	1	1	1.1	50	= 2*p; (i = 2*my_rank + 3; i <= n; i += incr)			
T Breakpoint	3	3	0.2-3	52 53	<pre>f (Is_prime(1)) { my_primes[my_prime_count++] = 1; }</pre>			
				54	<pre>ifdef DEBUG printf("Proc xd > xd\n", my_rank, i);</pre>			
epoil_wait_nocancel	3	3	0.3, 2-3	56	endif	0		
poll_nocancel	3	3	0.2, 2-3	58	<pre>t_list("After prime finder", wy_primes, wy_prime_count, wy_rank);</pre>	Local Variables ×		
'▼ main	3	3	0.1, 2-3	50 60	<pre>nt_primes(my_primes, my_prime_count, my_rank, p, comm);</pre>	Name	Туре	Value
mpi_primes_sort.c#53	1 3	3	0.1, 2-3	61 62	(my_primes);			
2.1	1	1	0.1	63	<pre>/inalize();</pre>	argo	int	0x00000002 (2)
4.1	1	1	2.1	65	samately, in 0; sin */	⇒ argv	Sstring **	0x7fffa37d3338 -
5.1	1	1	3.1	67		n	int	0x000003e8 (100
		_		60	tion: Get_n		int	0x00000000 (0)
•				70	ose: Get the input value n		int	0x00000004 (4)
Replay Bookmarks X				Action	Data View X Command Line X Logger X	local_n	int	0x00000007f (127
ID* PC File Line			Comment		Type Stop Location Line Function	incr	int	
				V	Break Process/mpi_primes_sort.c#51 mpi_primes_sort.c (line 51) main			0x0000008 (8)
						my_rank	int	0x00000000 (0)
						▶ comm	MPI_Comm	0x006022e0 (∨
						my_primes	int *	0x02116520 -> 0
						my_prime_co	int	0x00000001 (1)

Supported Technologies...



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

				tx_basic_data - Process 1, Thread 1.1 (Stopped) - TotalView 2020 _ • ×						
Eile Edit Group Process Thread	Action Poir	nts <u>D</u> ebug	g <u>W</u> indow <u>H</u> elp							
Group (Control)	▶ ⊵	1			nplayEngine 🥌 🗲 🖆 🕑 🖂 🕅 🗏 🕏					
Processes & Threads × Lookup File or I	Function ×	Docume	ents ¥ OpenMP ¥	Start P	age × tr_basic_data.cox ×	Call Stack ×				
0 2 3 = =				36) myStruct;	T				
Description			Members	- 38 1	struct simple_struct2 {	C++ main				
v tx_basic_data (S3)	1		pl	- 40	int foo[5]; char a;	libc_star	t_main			
Stopped	1		p1	41 42	<pre>int* dynInts; } myStruct2;</pre>	_start				
Stopped	1		p1.1	43	<pre>static char *statString = "This is a static string";</pre>					
▼ main	1	1	p1.1	45	int main(int argc, char** argv)					
tx_basic_data.cxx#48	1	1	p1.1	47	{					
1.1	1	1	p1.1	48	<pre>int i = argc; int *nullPtr = 0;</pre>					
				50 51	<pre>char **dynStrings; double dbl_array[10];</pre>					
				52 53	float flt_array[10];					
				54	<pre>for (i = 0; i < 10; i++) {</pre>					
				56	dbl_array[i] = i * 2.4;					
				57 58	<pre>flt_array[i] = i * 4.5; };</pre>					
				59 60	myStruct.a = 'a';					
				61	myStruct2.a = 'a';					
						-				
					Fiew X Command Line X Input/Output X Logger X					
				Name		-				
				[A	dd New Expression]	-				
						0	-			
						Local Variables ×				
Ó						Name	Туре	Value		
						 Arguments 				
Action Points X Replay Bookmarks X				-		argc	int	0x0000001(1)		
ID♥ Type Stop		Location				▶ argv	\$string **	0x7ffef7863308 -> 0x		
						▼ Block at Line 44	3			
						i	int	0x00000000 (0)		
						▶ nullPtr	int *	0x7ffef7863300 -> 0x		
Decessor 1 (2005) to basis data Th		005			ie: /home/bburns/Demos/lv_tests/src/tx_basic_data.cxxLine: 48		_			

Process and Threads View

Processes & Threads 🗱			
0 2 3 = =			
Description	# P	#T ¥	Members
 tx_fork_loop (S3) 	4	4	p1-4
Breakpoint	4	4	p1-4
The 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
The stopped Stopped	4	8	p1.1, p3.1, p1-2
select_nocancel	2	3	p1-2.2, p2.3
v <unknown line=""></unknown>	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,
<pre>v tx_fork_loop.cxx#682</pre>	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 X					
0 2 3 =					
Description	# P	#T 💙	Members		
tx_fork_loop (S3)	4	4	p1-4		
Breakpoint	4	4	p1-4		
The Breakpoint	4	4	p2.1, p4.1, p3.2,		
0					
Select process or thread attributes to group by	y:				
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					

っ

....

Source View

```
Start Page  🕷
            tx_fork_loop.cxx 🗱
1088 #else
          whoops = pthread_create (&new_tid, &attr, (void*(*)(void*))forker, (void *)local_fork_count);
1089
1090
        else
          whoops = pthread_create (&new_tid, NULL, (void*(*)(void*))forker, (void *)local_fork_count);
1091
1092
      #endif
1093
        if (whoops)
1094
          {
            printf("pthread_create failed; result=%d, errno=%d\n", whoops, errno);
1095
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 */
     } /* fork_wrapper */
1102
1103
1104
       /****
1105
      int main (int argc, char **argv)
1106
1107 {
1108
        int fork_count = 0;
        int args_ok = 1;
1109
        int arg_count = 1;
1110
1111
        char *arg;
1112
        pthread_mutexattr_t mattr;
1113
        signal (SIGFPE, sig_fpe_handler);
1114
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
                                                                                                  Aa
                                                                                                         "w"
                                                                                                                 Ð
```

Call Stack View and Local Variables View

Call Stack ×					
Ŧ					
C ++	funcB				
C++	funcA				
C++	funcB				
C++	funcA				
C++	funcB				
C++	funcA				
C++	funcB				
C++	funcA				
C++	funcB				
() ()	funcA				

Local Variables × Lookup File or Function ×					
Name	Туре	Value			
 Arguments 					
b	int	0x00000012 (18)			
 Block at Line 47 					
с	int	0x00000014 (20)			
i	int	0x00000000 (0)			
v 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

Openi	OpenMP * Action Points * Data View * Replay Bookmarks * Command Line * Input/Output *						
	ID♥	Туре	Stop	Location	Line	Function	
\checkmark	1	Break	Process	/ReplayEngine_demo.cxx#27	ReplayEngine_demo.cxx (line 27)	main	
\checkmark	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

41 42	int operator< (const fu	<pre>char * n) : value(v), name(n) {} nny_key & that) const { return value < that.value; }</pre>
43 44 45 46	}; int breakpoint() {	Modify Evaluation Point ×
47 48 49 50	<pre>return 1; } int main()</pre>	Modify Evaluation Point ID: 1 I Enabled
51 52 53 54	<pre>{ map<int,int> m1; int i; for (i = 1; i <= 60; i+-</int,int></pre>	Evaluate this expression at location: $tx_ttf_map.cxx#54$
55 56 57 58 59 60 61 62 63 64 65	<pre>{ fill = i*i; map<int, int=""> *m2; map<int, int=""> *m2; m2 = new map<int, int="">; (*m2)[4] = 4*4; (*m2)[5] = 5*5; (*m2)[6] = 6*6;] }</int,></int,></int,></pre>	<pre>if (i > 5 && i < 9) { printf("Adjusting i to %d\n", i+1); i = i + 1; } if (i == 10) { printf("Skip out to line 59 when i = %d\n", i); goto 59; } </pre>
66		Enter an expression, for example: if (i == 20) \$stop
Bookma ion	urks ×	Language: C++
	tx_ttf_map.cxx (line 54	DELETE MODIFY EVALUATION POINT CANCEL

Preferences

File > Preferences Menu

Or

"Gear" Toolbar Item



		Preferences	×
	DISPLAY	Display Settings	
	ACTION POINTS	Customize user interface display settings.	
Q,	SEARCH PATH	Appearance	
	PARALLEL	Choose the best interface style for your development.	
^و مح	REMOTE CONNECTIONS		
×	TOOL BAR	Light Dark	
2	LABS	User Interface Style Choose the type of user interface.	
		New Interface	
		Modern, dockable style user interface with improved low to medium scale multi-process and multi-thread dynamic analysis and debugging.	
		Classic Interface	
		Traditional, dedicated window for very high-scale multi-process dynamic analysis and debugging.	
		Font Size Choose the font size for the user interface.	
		Small Medium Large X-La	rge
		Display Settings changes will take affect the next time the product is started.	INCEL

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.

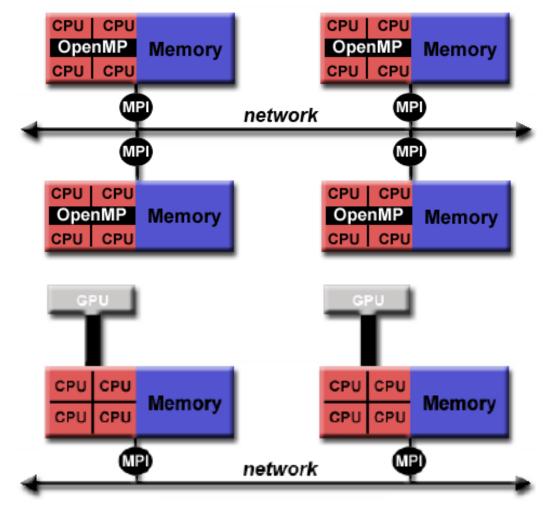


Image from U.S. Department of Energy by Lawrence Livermore National Laboratory

Debugging Hybrid Models – MPI/OpenMP/GPU

Hybrid Debugging with TotalView

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

	mpirun <mpi_array_broken>.0 - Rank 0, Thread 0.1 (Breakpoint) - TotalView 2020</mpi_array_broken>	_ • :
Eile Edit Group Process Thread Action Points	jookmarks <u>D</u> ebug <u>W</u> indow <u>H</u> elp	
Group (Control) 🔰 📔 📕 🗈 🛃	5 ME / ReplayEngine 🥌 🗲 🖆 🕑 🖄 E4 🕅 🕱	
Documents ×	Start Page × read nocancel × mpl bug4.c ×	Call Stack ×
open Documents:	<pre>48 offset = chunksize + leftover; 49 for (dest=1; dest<numtasks; dest++)="" pre="" {<=""></numtasks;></pre>	T
Start Page	50 MPI_Send(&offset, 1, MPI_INT, dest, tag1, MPI_COMM_WORLD); 51 MPI_Send(&daffset], chunksize, MPI_DOUBLE, dest, tag2, MPI_COMM_WORLD);	i main
read_nocancel <no source=""></no>	<pre>52 printf("Sent %d elements to task %d offset= %d\n",chunksize,dest,offset); 53 offset = offset + chunksize;</pre>	libc_start_main
mpl_bug4.ce/bburns/Demos/MPI/LLNLMPIExamp		_start
	<pre>56 /* Master does its part of the work */ 57 offset = 0;</pre>	
	57 mysum = update(offset, chunksize+leftover, taskid); 59	
	60 /* Wait to receive results from each task */ 61 for (i=1; icnumtasks; i++) {	
	<pre>01 Tor(l=1;1<mutasks;1++) &status);="" &status);<="" 1,="" 60="" 61="" 62="" 63="" 64="" 65="" 66="" 67="" 68="" 69="" mpi_comm_world,="" mpi_int,="" mpi_rcmm_world,="" mpi_recv(&offset,="" source="1;" source,="" taq1,="" td="" {=""><td></td></mutasks;1++)></pre>	
	64 MPI_Recv(&data[offset], chunksize, MPI_DOUBLE, source, tag2,	
Processes & Threads *	67 68 /* Get final sum and print sample results */	
0 2 3 =	<pre>69 printf("Sample results: \n"); 70 offset = 0;</pre>	-
escription # P # T ¥ Memb		0
r mpirun (S3) 1 1 p1	<pre>73 printf(" %e",data[offset+j]); 74 printf("\n");</pre>	Local Variables × Lookup File or Function ×
Running 1 1 p1	75 offset = offset + chunksize; 76 }	Name Type Value
▼ <unknown address=""> 1 1 p1.1</unknown>	77 printf("*** Final sum= %e ***\n",sum);	▼ Arguments
▼ <unknown 1="" addr="" p1.1<="" td=""><td>79 } /* end of master section */</td><td>argc int 0x00000001 (1)</td></unknown>	79 } /* end of master section */	argc int 0x00000001 (1)
1.1 1 1 p1.1	81 82	argv \$string ** 0x7ffd6289c708 -> 0x7ffd6289e09e -> "mpi
mpi_array_broken (S4) 4 4 0-3	83 /***** Non-master tasks only *****/ 84	numtasks int 0x00000004 (4)
Breakpoint 1 1 0	85 if (taskid > MASTER) {	taskid int 0x0000000 (0)
▼ main 1 1 0.1	<pre>87 /* Receive my portion of array from the master task */ 88 source = MASTER;</pre>	dest int 0x00000004 (4)
mpi_bug4.c#69 1 1 0.1	89 MPI_Recv(&offset, 1, MPI_INT, source, tag1, MPI_COMM_WORLD, &status); 99 MPI_Recv(&data[offset], chunksize, MPI_DOUBLE, source, tag2,	offset int 0x00e4e1c0 (15000000)
2.1 1 1 0.1	91 MPI_COMM_WORLD, &status);	i int 0x0000004 (4)
▼ Running 3 3 1-3	Action Points × Data View × Replay Bookmarks × Command Line × Logger × Input/Output ×	j int 0x0000000 (0)
v <unknown address=""> 3 3 1-3.1</unknown>	ID* Type Stop Location Line Function	tag1 int 0x0000002 (2)
v <unknown 1-3.1<="" 3="" addr="" p=""></unknown>	Image: Image: Second secon	tag2 int 0x00000001 (1)
3.1 1 1 1.1	✓ 2 Break Process/mpi_bug4.ct/69 mpi_bug4.ct (line 69) main	source int 0x0000003 (3)
4.1 1 1 2.1		chunksize int 0x004c4b40 (5000000)
5.1 1 1 3.1		leftover int 0x00000000 (0)
		mysum double 24999995000000
Rank: 0 (3180) mpirun <mpi array="" broken="">.0 Thre</mpi>	d: 0.1 (0x7/599dbedb40) - Breakpoint Frame: main File: /home/bburns/Demos/MPI/LNLMPIExamples/mpi buq4.c Line: 69	sum double 0 Source Line: 64

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

tx_cuda_i	matmul - Process 1, No current thread - TotalView for HPC 2019X (on microway2.totalviewte Desug Window Help	ch.com)	 × ×
Group (Control) : 🕨 📕 📕 🖿 🖄 🗹	HE 🖊 🚺 Replaying 🖷 🛋 🖆 🖄 🗄 🗄 🕹		
GPU (Logical) Block 0 + 0 + 0 + Thread 0 + 0 +	0 (GPU (Physical) Device 0 (SM 0 (Warp 0 Lane 0)		
③ ② ③ ■ 154 Description #P #T ▼ Nembers 156 V tx_cuda_matmul (S3) 1 1 1 Honexistent 1 1 1 1	<pre>for [int col = '0; col < A.vidth; col+') printf ('[ssd][ssd] sf\n', row, col, A.elements[row * A.stride + col]); } // vultiply an m*n matrix with an n*p matrix results in an m*p matrix. // usage: tx_cuda_mathul [m [n [n]]] // m, n, and p default to 1, and are multiplied by BLOCK_SIZE. int main[int argc, char *Aargv) { // cudasetDevice[0]; const int n = BLOCK_SIZE * (argc > 1 7 atoi(argv[1]) : 1); const int n = BLOCK_SIZE * (argc > 2 7 atoi(argv[2]) : 1); const int n = BLOCK_SIZE * (argc > 3 7 atoi(argv[2]) : 1); matrix A = cons_Matrix(m, p); matrix B = cons_Matrix(m, p); matrix C = cons_Matrix(m, p); matrix(A, s, c); print_Matrix(C, 'c'); print_matrix(C, 'c'); return 0; </pre>	Cal Stack * No current thread Local Variables *	
Action Points ¥ Replay Bookmarks ¥ DY Type Step Location	Vew X Command Line X Logger X Input/Oxput X e Type Thread D Value [Add New Expression]		

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.

GPU Status					
ntrol Group 👻 👸					
ariables	Status				
Devices					
Device(0)					
V SM(\$sm)	for sm in [0 34] step by 2	2			
	and warp in {1 1 0 0 2 2 2	02021231101}			
Warp(\$warp)					
🔍 Lane(\$lane)	for lane in {0 1 4 5}				
	Line = tx_cuda_precise_di	iverge.cu#240			
	Function = update_data_e	ntry			
	State = diverged				
	GPU Status Conf	iguration Ontio	ns (Process	1) (on microwa	y1) 🛛
_ Displa		Group By		IF Sort By	
	ate	state		state	
de		dev		dev	
	nction	function		function	
			_		
	lew Filter:	JL SIII][] 5111	
include	▼ state	* ==	- diverged	- AND	- ADD
Filters:					
inc	lude state == breakpoint				AND ≍
inc	lude state == diverged				AND ×
		BEOFF			0411051
		RESET	ок	PREVIEW	CANCEL

AMD / ROCm GPU Debugging

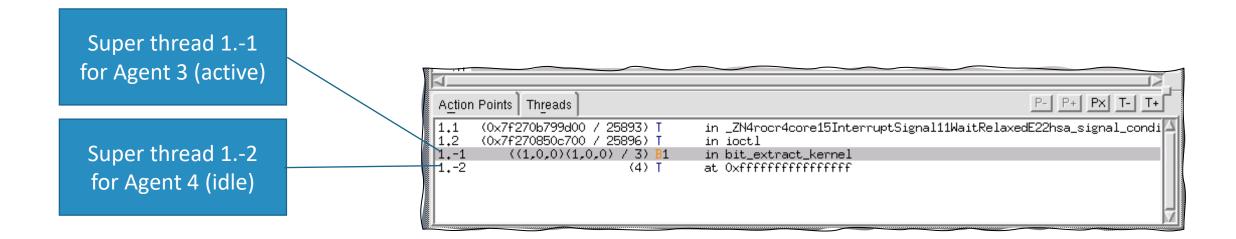
TotalView ROCm GPU Support

- 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"

<pre>./tx_hip_bit_extract</pre>	x e
File Edit View Group Process Thread Action Point Debug Tools Window	<u>H</u> elp
Group (Control) V Go Hait Kill Restart Next Step Out Run To Record GoBack Prev UnStep Caller BackTo Live	Save
Logical Work-group: 1 👌 , 0 👌 , 0 👌 Work-item: 1 👌 , 0 🍦 , 0	
Process 1 (25893): tx_hip_bit_extract (At Breakpoint 1)	
Thread -1 (3) ((1,0,0)/(1,0,0)): @TEMP@ROCM@.tx_hip_bit_extract (At Breakpoint 1) <breakpoint trap=""></breakpoint>	
<pre>C++ bit_extract_kernel, FP=6000000000000000000000000000000000000</pre>	
Function bit_extract_kernel in tx_hip_bit_extract.cpp	E P
<pre>37 38global void bit_extract_kernel(uint32_t* C_d, const uint32_t* A_d, size_t N) { 39 39 39 39 39 40 size_t offset = (hipBlockDim_x * hipBlockDim_x + hipThreadIdx_x); 40 41 42 42 43 44 42 44 45 44 45 44 45 46 44 45 46 46 46 47 46 46 47 46 46 4 4 4 4 4 4</pre>	
48 3 49 3 50	
Action Points Threads P-[P+[Px] T-]	T+ [
1 [/tx_hip_bit_extract.cpp#44] tx_hip_bit_extract.cpp#44 bit_extract_kernel+0x8c4	
1	

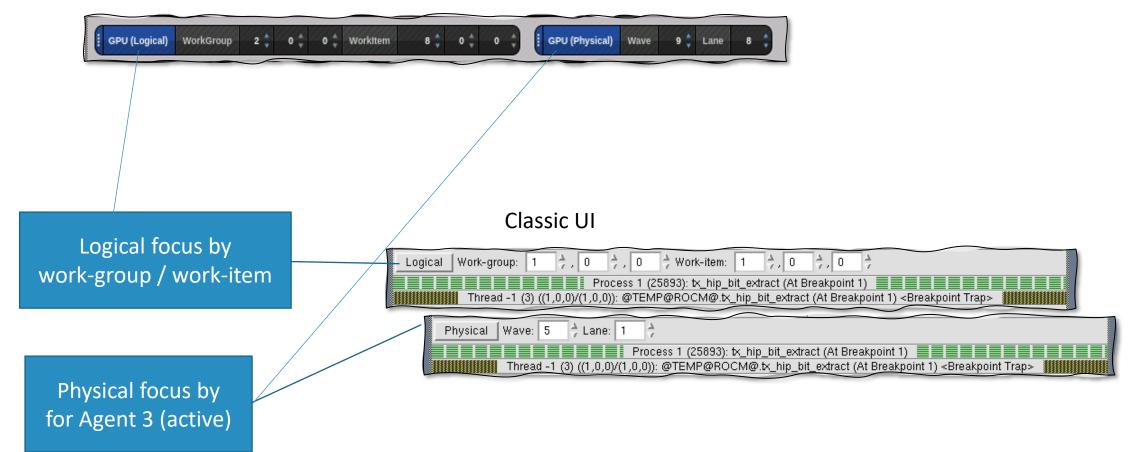
ROCm GPU Agents Are Represented As TotalView Threads

- 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



ROCm GPU Focus Control

New UI



Logical and Physical Focus, and Grid Dimensions

Grid dimensions

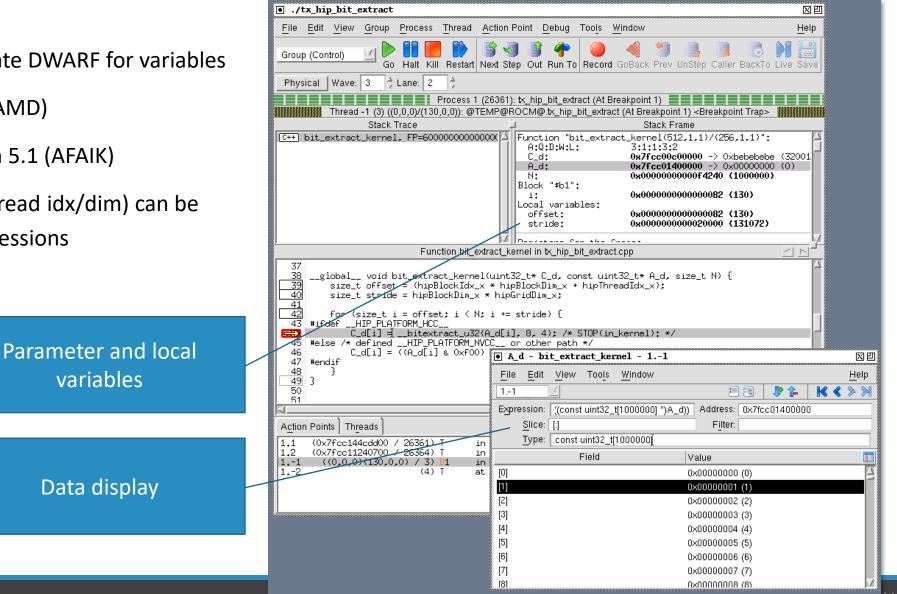
Logical focus displayed in thread status



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



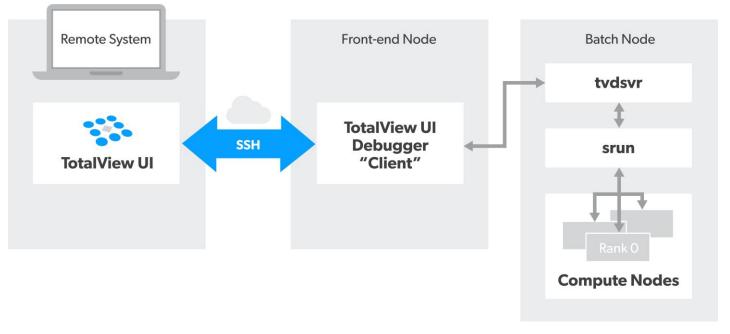
Debugging AMD GPUs with TotalView for AMD GPUs

- 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

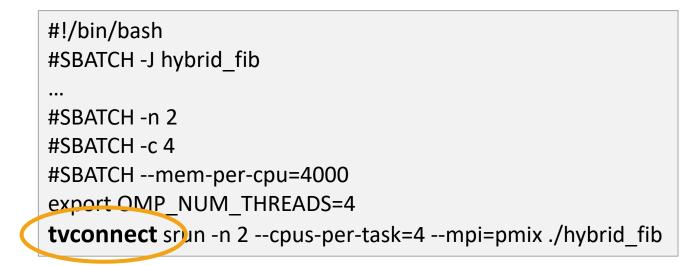


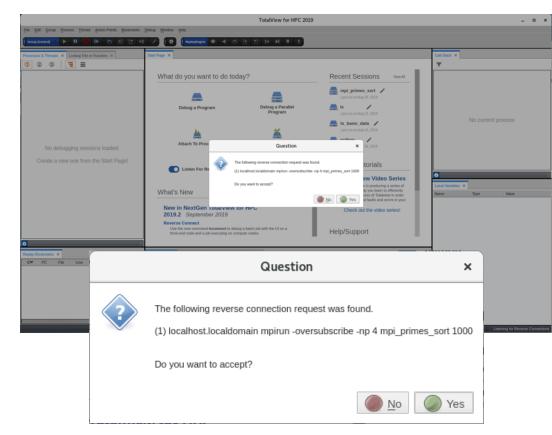
See it in action: <u>https://totalview.io/video-tutorials/how-use-remote-user-interface-debugging</u>

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.





ANL Connect Demo

- TotalView Reverse Connect Demo
- TotalView Remote Debugging Demo

Reverse Debugging

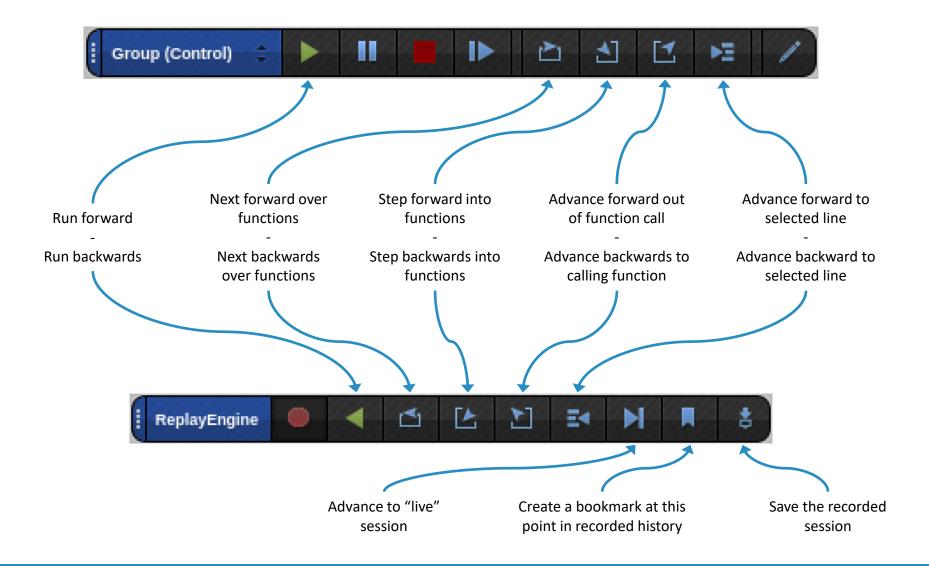
Reverse Debugging With TotalView

- 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.

Start	Page * common-main.c *
15	<pre>static void restore_sigpipe_to_default(void)</pre>
16	{
17	sigset_t unblock;
18	
19	sigemptyset(&unblock);
20	<pre>sigaddset(&unblock, SIGPIPE);</pre>
21	<pre>sigprocmask(SIG_UNBLOCK, &unblock, NULL);</pre>
22	<pre>signal(SIGPIPE, SIG_DFL);</pre>
23	}
24	
	<pre>int main(int argc, const char **argv)</pre>
26	{ /*
27	
28 29	* Always open file descriptors 0/1/2 to avoid clobbering files * 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	<pre>sanitize stdfds();</pre>
33	
34	git_setup_gettext();
35	
36	<pre>git_extract_argv0_path(argv[0]);</pre>
37	
38	restore_sigpipe_to_default();
39	
40	return cmd_main(argc, argv);
41	}
42	



Reverse Debugging Controls



Memory Debugging

TotalView HPC Memory Debugging

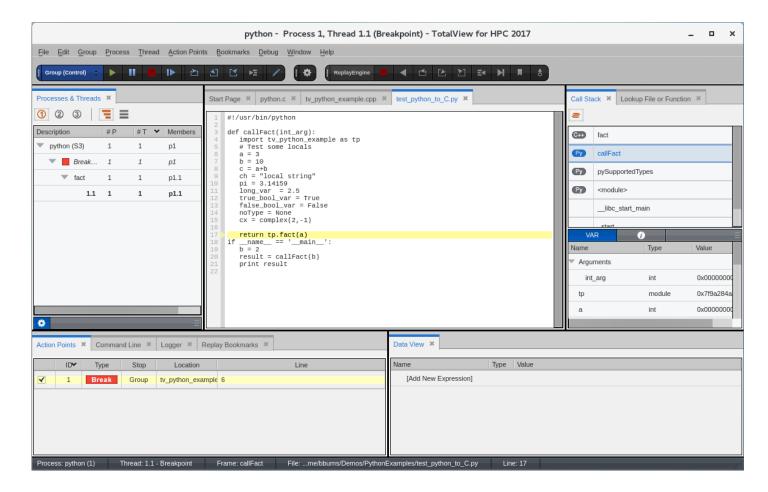
- Easily find memory leaks and other memory errors
- Understand heap usage
- Detect malloc/free new/delete API misuse
- Detect buffer overruns
- Understand where memory is being used
- Remote and MPI debugging

	git - Process 1, Thread 1.1 (Br	eakpoint) - TotalView 2021	_ •
Edit Group Process Thread Action Point	s Debug Window Help		
roup (Control) 🗧 🕨 📗 🖿			
esses ¥ .ookup File o ¥ Doc ¥ Dpe	* Start Page * wrapper.c * common-main.c * tv_heap_breakpoint.c *	Leak Report <p1> ×</p1>	Call Stack #
2 3 =	26 die("attempting to allocate %"PRIUMAX" over limit %"PRIUMAX 27 (uintmax_t)size, (uintmax_t)limit);	Process: git	T
ription # P	# 28 } 29 return 0;	Drag tab to detatch. Report may be placed side-by-side with source.	TV_HEAP_notify_breakpoint_here
r Breakpoint 1	1 30 } 31 31 32 try_to_free_t set_try_to_free_routine(try_to_free_t routine)	Process Bytes V Count Begin Address End Address B	TV_HEAP_notify_tv
Breakpoint 1	<pre>1 32 try_to_free_t set_try_to_free_routine(try_to_free_t routine) 33 { 1 34 try_to_free_t old = try_to_free_routine;</pre>	Process Bytes Count Begin Address End Address E Process 1 (5095): git 1157.18 KB 8959	C TV_HEAP_notify_event
TV_HEAP_notify_breakpoi 1	1 35 if (!routine) 1 36 routine = do_nothing;	wrapper.c 1157.18 KB 8959	TV_HEAP_malloc_interposer_fini
v tv_heap_breakpoint.c#59 1	37 try_to_free_routine = routine; 1 38 return old;	▼ xstrdup 201 6	TV_HEAP_linux_ia_fini
1.1 1	39 } 40	Line 43 201 6	_dl_fini
	41 char *xstrdup(const char *str) 42 {	▶ xrealloc 332 6	run_exit_handlers
	<pre>43 char 'ret = strdup(str); 44 if (!ret) { </pre>	▶ do_xmalloc 17.27 KB 133	exit
	45 try_to_free_routine(strl + 1); 46 ret = strdup(str); 47 if (Iret)	Backtrace:	handle_builtin
	48 die("Out of memory, strdup failed"); 49 }	ID Function Line # Source Information	C run_argv
	50 return ret; 51 }	▶ 383	C cmd_main
	52 53 static void *do_xmalloc(size_t size, int gentle)	▶ 93	r main
	54 { 55 void *ret;	▶ 92	libc_start_main
	56 57 if (memory_limit_check(size, gentle))	85	
	<pre>58</pre>	▶ 38 ▶ 36	Name Type Value
	61		▼ Arguments
	Action Points * Data View * Replay Bookmarks * Command Line * Input/Output *		event T 0x7fff8fb77840 -> (
	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) dl. \Leftrightarrow		
s: 1 (5095) git Thread: 1.1 (0x7/4/1be2a7-	t0) - Breakpoint Frame: TV HEAP notify breakpoint here File:works/totalview.2021.1.16/src/tv heap breakpo	int.c Line: 59	Source Line: 43

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

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

	<pre>void * read(void *temp) {</pre>			
28	char *ret;			
29	FILE *file1;			
30	char *str;			
31	pthread_mutex_lock(&read_mutex	:);		
32	<pre>sleep(5);</pre>			
33	pthread_mutex_lock(&write_mute	x);		
34	<pre>printf("\n Opening file \n");</pre>			
35	<pre>file1=fopen("temp","r");</pre>			
36	<pre>str=(char *)malloc(10*sizeof(c</pre>			
37	<pre>fscanf(file1,"%s",str);</pre>	Data View		
38	<pre>printf("\n Message from file i</pre>			
39				
40	<pre>fclose(file1);</pre>	Ð		
41			1	1
42	<pre>pthread_mutex_unlock(&write_mu</pre>	Name	Thread ID	Value
43	pthread_mutex_unlock(&read_mut			
43	pthread_mutex_unlock(&read_mut		1.2	(pthread_mutex_t)
okmark			1.2	(pthread_mutex_t) (structpthread_mutex_s)
L		read_mutex		
okmark	s ¥	▼ read_mutex	1.2	(structpthread_mutex_s)
okmark	s ×	▼ read_mutex	1.2	(structpthread_mutex_s)
okmark	s ¥	▼ read_mutex ▼data lock	1.2 1.2	(structpthread_mutex_s)

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

			Sessio	on Editor		
	Attach to Running Pro	gram(s)				
Sessi	ion Name					
[Enter	or select a session name, e.g. myprogram	n with ReplayEngi	ne]			~
Pro	cesses					
Host	localhost.localdomain (local)			Q [Search lis	st]	٥
Progra	m	▲ State	PID	PPID	Host	Path
	bash	S	18793	2617	localhost.locald	/usr/bin/
-	bash	S	28796	2617	localhost.locald	/usr/bin/
	▼ rwcoreapp	S	53344	28796	localhost.locald	/mnt/development/toolworks/tota
	tvdengine	R	53414	53344	localhost.localdom	a: /mnt/development/toolworks/totalvieu
	▼ orterun	S	53302	28796	localhost.locald	/mnt/development/software/open
	mpi_wave	S	53313	53302	localhost.locald	/home/bburns/Demos/MPI/LLNL
PID				Program		
PID			REQUIRED	File Name		REQUIRED
53302				/mnt/development/s	software/openmpi/openm	npi-3.1.0/bin/orterun 👻 Browse
Deb	oud Options					RESET ATTACH CANCEL

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

1 2 3 = = = = = = = = = = = = = = = = = =	# P	#T ¥	Members
 ThreadWorkers (S3) 	1	1	pl
Teakpoint	1	1	p1.5
Worker::doWork	1	1	p1.5
1.5	1	1	p1.5
💌 📕 Running	1	1	p1.1
vunknown address>	1	1	p1.1
1.1	1	1	p1.1
The Stopped	1	3	p1.2-4
poll_nocancel	1	3	p1.2-4
1.2	1	1	p1.2
1.3	1	1	p1.3
1.4	1	1	p1.4
Concess or thread attributes to g	roup by:	_	
Control Group			
✓ Share Group			
Hostname			
Process State			

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/TotalVIew/
 - www.totalview.io/downloads
- Copy /grand/ATRESC2022/EXAMPLES/track-6tools/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

		Preferences ×
	DISPLAY	Remote Connections
	ACTION POINTS	Create a configuration to launch the TotalView debugger on a remote host.
Q	SEARCH PATH	Provide the necessary information to reach the host and the debugger.
	PARALLEL	Select a configuration to edit: Cooley-ANL 👻 or create a new configuration
[%]	REMOTE CONNECTIONS	
×	TOOL BAR	Connection Name Provide a name for your connection.
ፚ	LABS	Cooley-ANL
		Remote Host(s) Provide a comma separated list of SSH hosts in the form: user@host1, user@host2 RecyumeD
		piskun@cooley.alcf.anl.gov
		Private Key File Specify an optional local private key (e.gpem or .ppk) file to use for the SSH connection.
		/my/path/my_key.pem BROWSE
		TotalView Remote Installation Directory Enter the path to the totalview executable or leave blank if totalview is in your path.
		/opt/toolworks/totalview.2020.3.0/bin
		Remote Command(s) Enter optional shell commands to execute or source a remote script before starting remote TotalView debugger
		soft add +totalview
		Delete Configuration
		OK APPLY CANCEL

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 -I
 - 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 -I -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

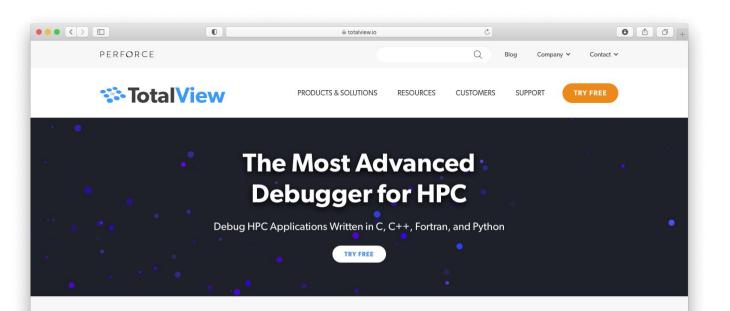
module load totalview

tvconnect tx_cuda_matmul

DISPLAY	Remote Connections				
ACTION POINTS	Create a configuration to launch the TotalView debugger on a				
SEARCH PATH	Provide the necessary information to reach the host and the debugger.				
PARALLEL	Select a configuration to edit: ThetaGPU - or create a new configuration				
REMOTE	Connection Name				
TOOL BAR	Provide a name for your connection.				
LABS	ThetaGPU				
	Remote Host(s) Provide a comma separated list of SSH hosts in the form: user@host1, user@host2				
	piskun@theta.alcf.anl.gov, piskun@thetagpusn1				
	Private Key File Specify an optional local private key (e.gpem or .ppk) file to use for the SSH connection.				
	/my/path/my_key.pem				
	TotalView Remote Installation Directory Enter the path to the totalview executable or leave blank if totalview is in your path. /grand/projects/ATPESC2022/EXAMPLES/track-6-tools/TotalView/toolworks/totalview.2022.2.13/bin				
	Remote Command(s) Enter optional shell commands to execute or source a remote script before starting remote TotalView debugger				
(qsub 2022/labs/programs/tvconnect-thetaGPU.job				

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



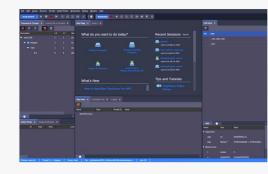
Why Do Top HPC Developers Use TotalView for Debugging Code?

You need special tools for multithreaded, multiprocess, and GPUspecific 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



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: <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







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!