

Thomas Reichstein

reichstein@rz.rwth-aachen.de

Center for Computing and Communication

RWTH Aachen University

22.03.2010

Why use a debugger?

Before you start

Starting TotalView

Basic debugging

Multithreaded (OpenMP) debugging

MPI debugging

Advanced

- Memory debugging

- Replay Engine

# Why use a debugger?

---

If your program goes haywire, you may...

(... buy a magic wand)

... read the source code again and again and ...

... enrich your application with printf's

or

Use an adequate tool – a debugger.

Debugger enhance the productivity

## A comprehensive debugging solution for demanding parallel and multi-core applications

- wide compiler & platform support
  - **C, C++, Fortran 77 & 90, UPC**
  - **Linux, OS X, Unix**
  - Windows frontend (client)
- handles concurrency
  - **multi-threaded debugging**
  - **parallel debugging**
    - **MPI, PVM, others**
  - remote and client/server debugging
- **Integrated Memory Debugging**
- **Reverse Debugging available**
- **ReplayEngine**
- Supports a Variety of Usage Models
  - **powerful and easy GUI visualization**
  - CLI for scripting
  - long distance remote debugging
  - unattended batch debugging / GUI-free debugging with TVScript

## Before you start

---

Lean back and relax

Check your environment (ulimit -a):

- s Stack size (crucial for Fortran and OpenMP!)
- t CPU time
- v Address space and others
- c Core file size (crucial for debugging on core files)

Remove all objects and intermediate files

Rebuild with debugging info *ON* (-g)  
optimization *OFF* (-O0)

Problem still here? Use a debugger!

Initialise the environment:

```
$ module load totalview
```

Startup:

```
$ totalview
```

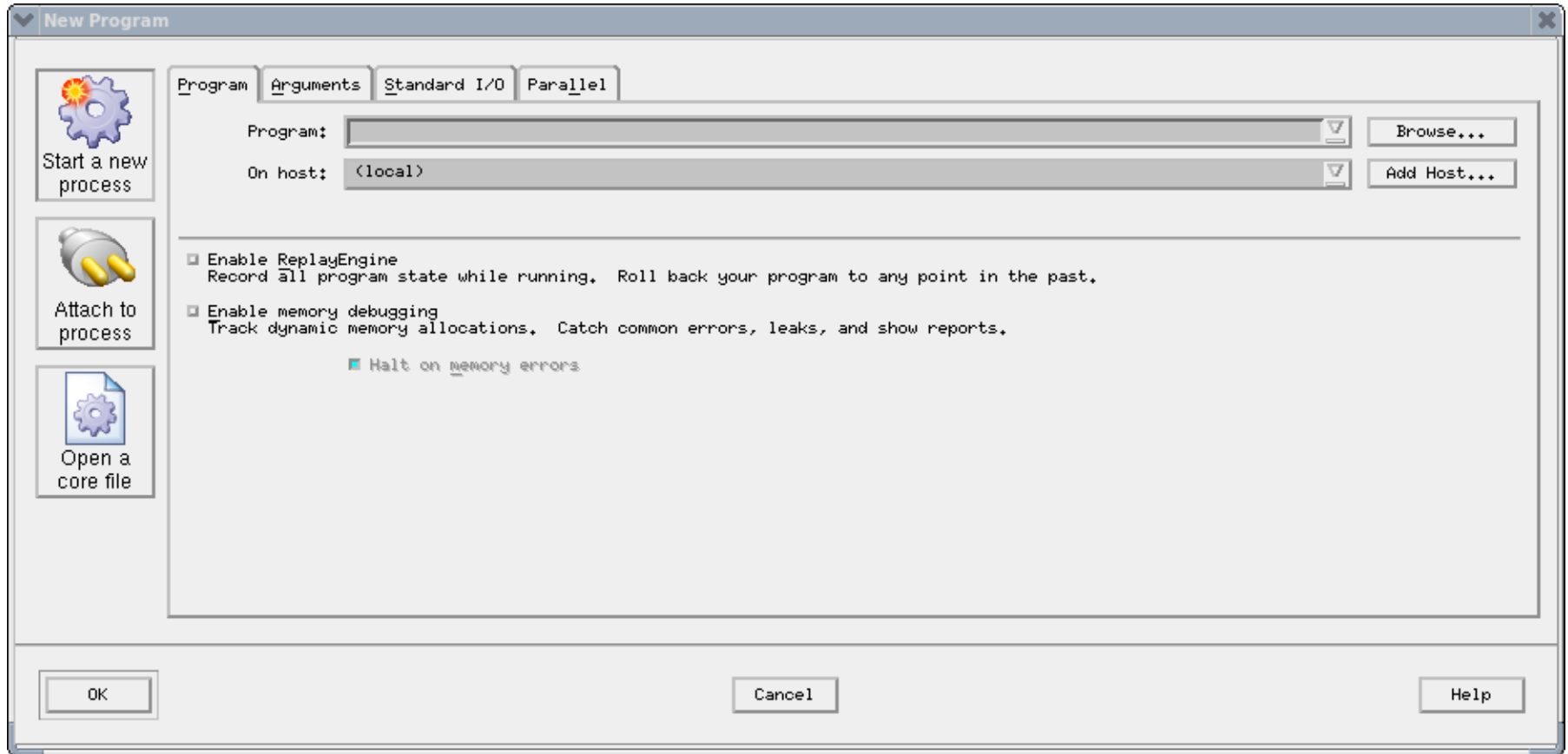
Main modes: start a new process

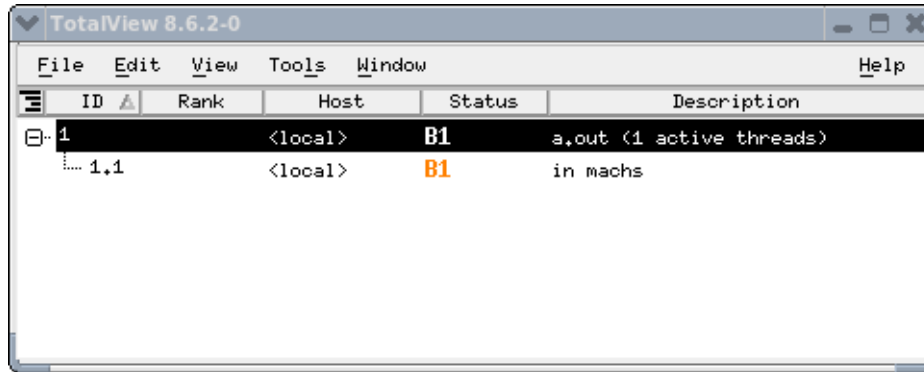
attach to a running process

load a core file

or load a binary called a.out directly:

```
$ totalview a.out -a <options of a.out>
```

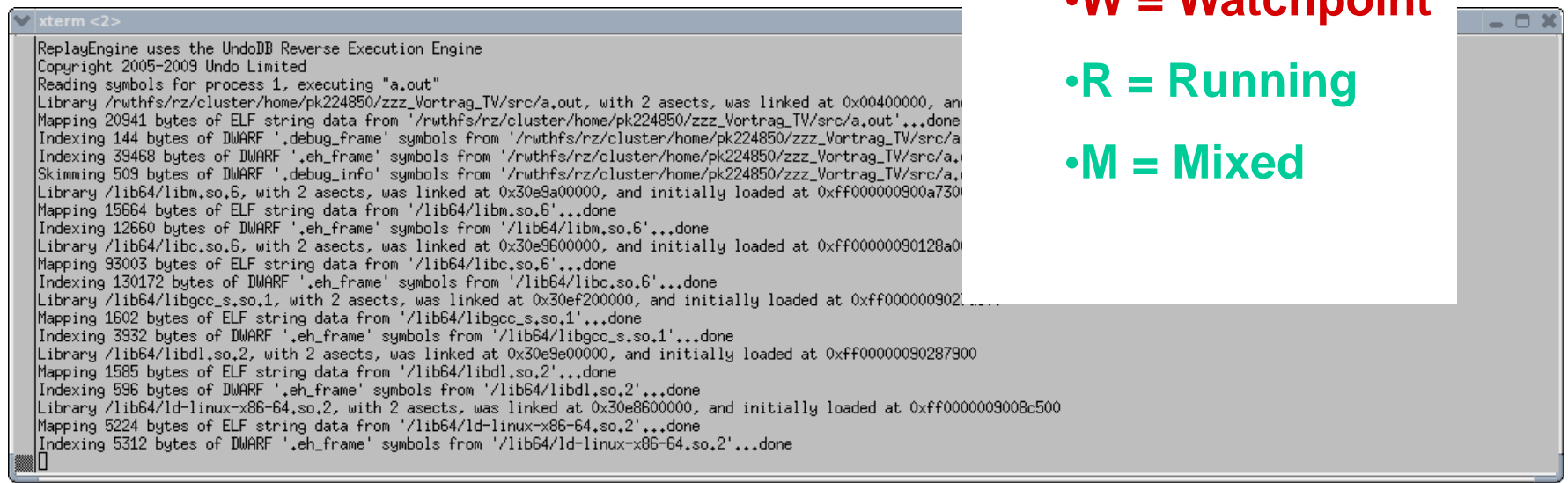




ID	Rank	Host	Status	Description
1		<local>	B1	a.out (1 active threads)
1.1		<local>	B1	in machs

## Status Info

- T = stopped
- B = Breakpoint
- E = Error
- W = Watchpoint
- R = Running
- M = Mixed



```
xterm <2>
ReplayEngine uses the UndoDB Reverse Execution Engine
Copyright 2005-2009 Undo Limited
Reading symbols for process 1, executing "a.out"
Library /rwthfs/rz/cluster/home/pk224850/zzz_Vortrag_TV/src/a.out, with 2 asects, was linked at 0x00400000, and initially loaded at 0x00400000
Mapping 20941 bytes of ELF string data from '/rwthfs/rz/cluster/home/pk224850/zzz_Vortrag_TV/src/a.out'...done
Indexing 144 bytes of DWARF '.debug_frame' symbols from '/rwthfs/rz/cluster/home/pk224850/zzz_Vortrag_TV/src/a.out'...done
Indexing 39468 bytes of DWARF '.eh_frame' symbols from '/rwthfs/rz/cluster/home/pk224850/zzz_Vortrag_TV/src/a.out'...done
Skimming 509 bytes of DWARF '.debug_info' symbols from '/rwthfs/rz/cluster/home/pk224850/zzz_Vortrag_TV/src/a.out'...done
Library /lib64/libm.so.6, with 2 asects, was linked at 0x30e9a00000, and initially loaded at 0xff000000900a7300
Mapping 15664 bytes of ELF string data from '/lib64/libm.so.6'...done
Indexing 12660 bytes of DWARF '.eh_frame' symbols from '/lib64/libm.so.6'...done
Library /lib64/libc.so.6, with 2 asects, was linked at 0x30e9a00000, and initially loaded at 0xff00000090128a00
Mapping 93003 bytes of ELF string data from '/lib64/libc.so.6'...done
Indexing 130172 bytes of DWARF '.eh_frame' symbols from '/lib64/libc.so.6'...done
Library /lib64/libgcc_s.so.1, with 2 asects, was linked at 0x30e9e00000, and initially loaded at 0xff00000090212800
Mapping 1602 bytes of ELF string data from '/lib64/libgcc_s.so.1'...done
Indexing 3932 bytes of DWARF '.eh_frame' symbols from '/lib64/libgcc_s.so.1'...done
Library /lib64/libdl.so.2, with 2 asects, was linked at 0x30e9e00000, and initially loaded at 0xff00000090287900
Mapping 1585 bytes of ELF string data from '/lib64/libdl.so.2'...done
Indexing 596 bytes of DWARF '.eh_frame' symbols from '/lib64/libdl.so.2'...done
Library /lib64/ld-linux-x86-64.so.2, with 2 asects, was linked at 0x30e9e00000, and initially loaded at 0xff0000009008c500
Mapping 5224 bytes of ELF string data from '/lib64/ld-linux-x86-64.so.2'...done
Indexing 5312 bytes of DWARF '.eh_frame' symbols from '/lib64/ld-linux-x86-64.so.2'...done
```

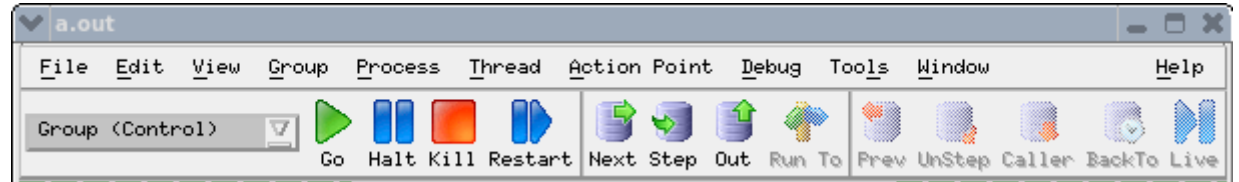


# Process Window

The screenshot displays the Process Window of a debugger, showing the state of a process named 'a.out'. The window is divided into several panes and includes a toolbar at the top.

- Toolbar:** Located at the top, it contains buttons for 'Go', 'Halt', 'Kill', 'Restart', 'Next Step', 'Out', 'Run To', 'Prev UnStep', 'Caller', and 'Back To Live'.
- Process and Thread Status:** Below the toolbar, it shows 'Process 1 (24077): a.out (At Breakpoint 1)' and 'Thread 1 (46912497811904) (At Breakpoint 1)'.
- Stack Trace Pane:** On the left, it lists the call stack: 'f90 machs', 'f90 bsp\_ppces\_1', 'main', '\_\_libc\_start\_main', and '\_start'.
- Stack Frame Pane:** On the right, it shows the function 'machs' with local variables: 'widerhol: 10 (0x0000000a)', 'maxiter: 20000000 (0x01312d00)', 'maxiter: <Bad address: 0x7fff00000001>', 'j: 1970169159 (0x756e6547)', 'i: 0 (0x00000000)', 'var: (REAL\*4, allocatable::(2', and 'j: 41 (0x00000029)'. It also shows 'Registers for the frame:'.
- Source Pane:** In the center, it displays the source code for 'Function machs in bsp\_PPCEs\_1.f90'. The current line is 'var = 3.14'.
- Tabbed Pane:** At the bottom, it shows 'Action Points', 'Processes', and 'Threads'. The 'Action Points' tab is active, showing a list of breakpoints.

# Stepping Toolbar



Go: run the program

Halt: pause the execution

Kill: ends the process

Restart: Kill + Go

Next: executes all code on line (e.g. subroutine call)

Step: executes line (e.g. dive in a subroutine)

Out: run up to return of current routine

Run to: runs to a selected line

There are a lot of ways:

Glance an Hover:

1. in Stack Frame right on top
2. hold the cursor over a variable

Dive to data window – updated at each stop!

- 3.1 right-click on a variable and then “dive”
- 3.2 double-click on a variable
- 3.3 View → Lookup Variable

Monitor via expression list

4. right-click on a variable and then  
“Add to Expression list”

# Viewing Examples

The screenshot shows a debugger interface with three main windows:

- Top Window (a.out):** Shows the process and thread information. The thread is "Thread 1 (5562) (At Breakpoint 4)".
- Left Window (j - machs - 1.1):** Shows the expression list. The expression "j" is entered, and its value "9 (0x00000009)" is displayed. The expression "i" is also visible with its value "5 (0x00000005)".
- Right Window (Stack Frame):** Shows the stack frame for the function "machs". It lists local variables: "j" with value "9 (0x00000009)" and "i" with value "5 (0x00000005)". It also shows registers: "%rax: 0x00000009 (9)", "%rdx: 0x01312d00 (2000000)", "%rcx: 0x41500000 (109576192)", "%rbx: 0x00000000 (0)", and "%rsi: 0x00000000 (0)".

Red boxes highlight the expression list in the left window and the local variables in the right window.

Expression list →

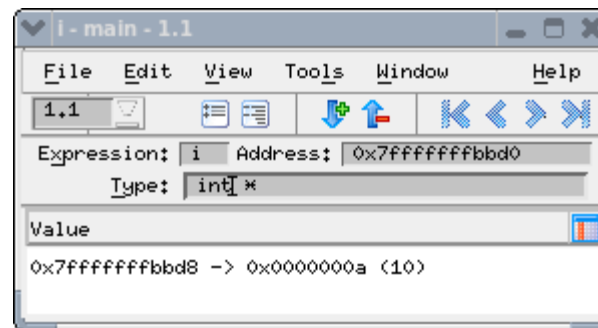
Expression	Value
j	9 (0x00000009)
i	5 (0x00000005)

# Looking on user-defined types

Just open in a new window (double-click)

Arrays may be shown with all elements

C/C++: an allocated array can be just a pointer to a memory location



debugger need help!

*"Type Casting"*

Edit the type of a variable

## *“Type Casting”*

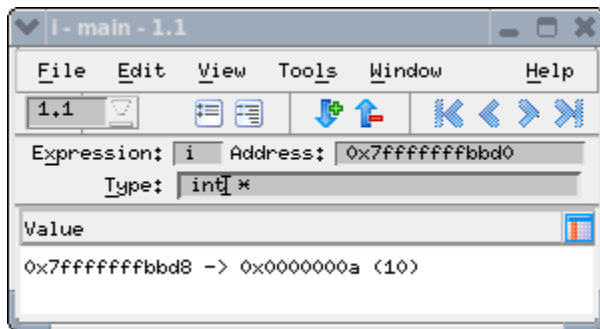
Edit the type of a variable

Type Casts Read from Right to Left

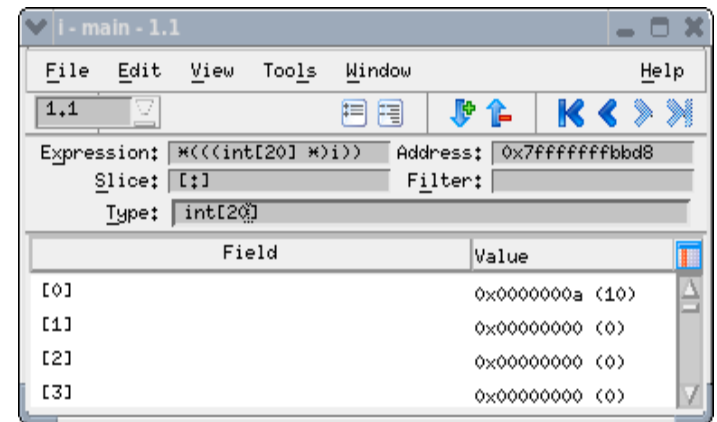
Examples:

`int[10]*`      Pointer to an array of 10 int

`int*[10]`      Array of 10 pointers to int

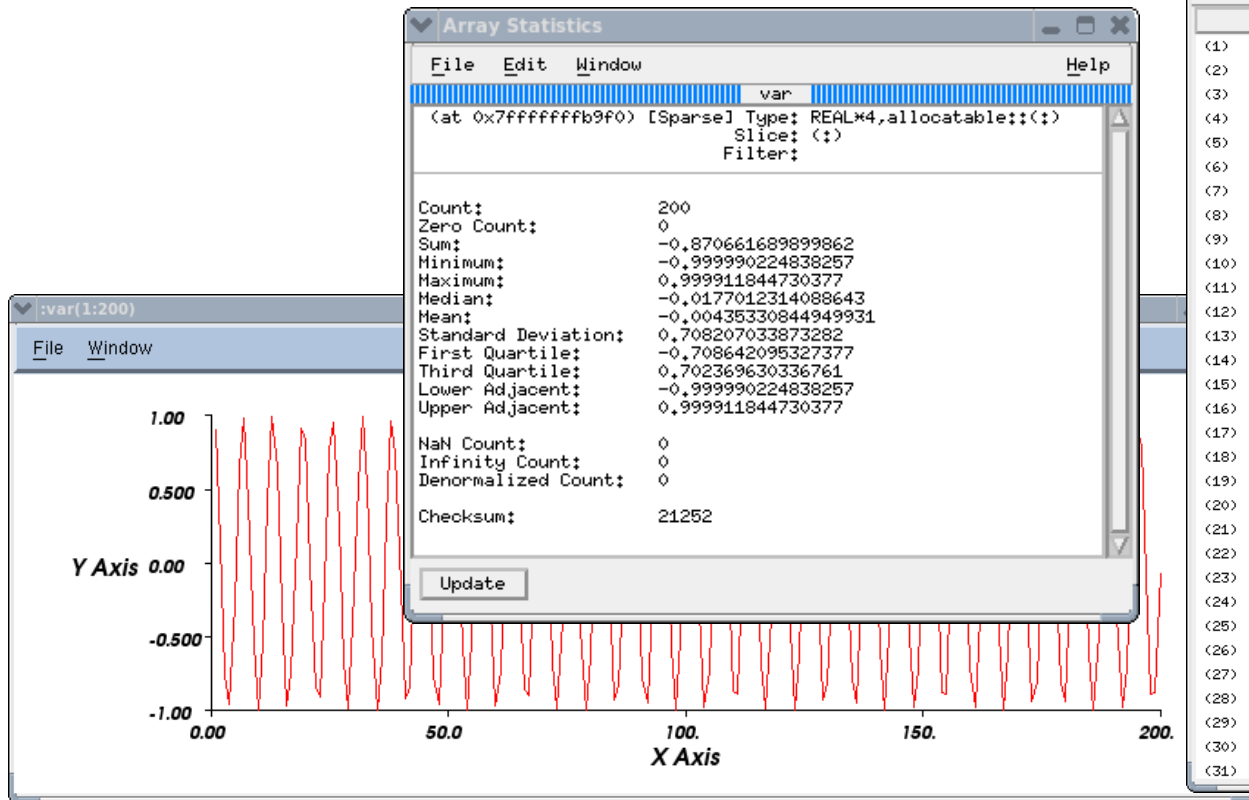
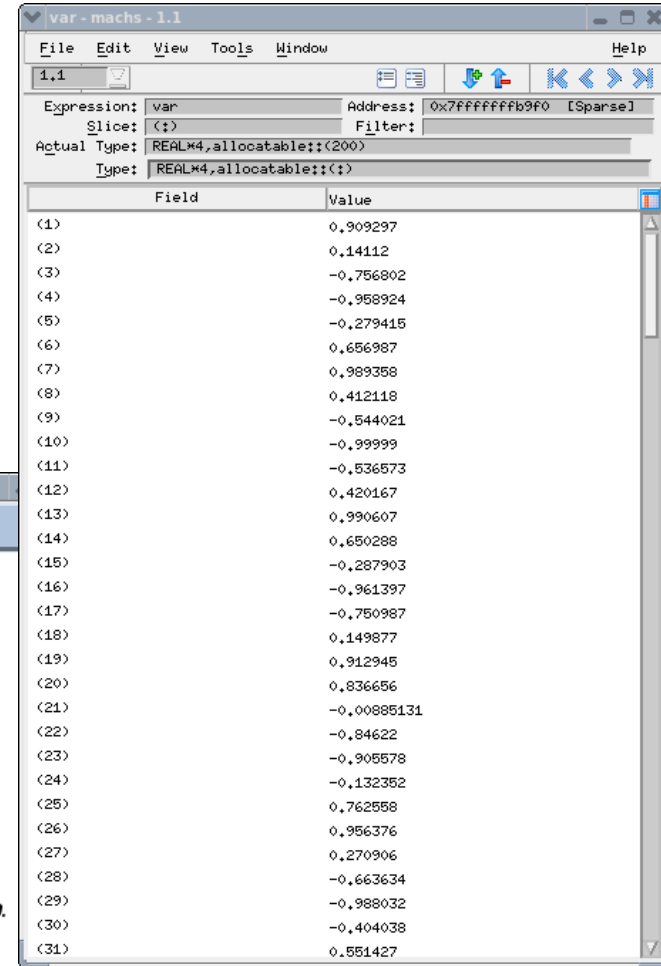


Type  
Casting



Tools → Visualize

Tools → Statistics

The variable viewer window displays a list of values for the variable 'var'. The values are listed in a table with columns 'Field' and 'Value'.

Field	Value
(1)	0.909297
(2)	0.14112
(3)	-0.756802
(4)	-0.958924
(5)	-0.279415
(6)	0.656987
(7)	0.989358
(8)	0.412118
(9)	-0.544021
(10)	-0.99999
(11)	-0.536573
(12)	0.420167
(13)	0.990607
(14)	0.650288
(15)	-0.287903
(16)	-0.961397
(17)	-0.750987
(18)	0.149877
(19)	0.912945
(20)	0.836656
(21)	-0.00885131
(22)	-0.84622
(23)	-0.905578
(24)	-0.132352
(25)	0.762558
(26)	0.956376
(27)	0.270906
(28)	-0.663634
(29)	-0.988032
(30)	-0.404038
(31)	0.551427

Click on the line number in source pane to set a breakpoint

Right-click on Breakpoint to get a menu

Breakpoints are collected in the tabbed pane

Press **GO** – program runs until just the next breakpoint and then stops

Temporary disabling a breakpoint: click on it in the tabbed pane

Remove a breakpoint: click on it in the source



Why use a debugger?

Before you start

Starting TotalView

Basic debugging

Multithreaded (OpenMP) debugging

MPI debugging

Advanced

Memory debugging

Replay Engine

# Parallelisation – the very basics

---

serial programs: only one Unit of Execution (UE)  
all instructions one by one in deterministic order



parallel programs: more than one UE



the order of instructions not determined between different UEs  
*shared memory* parallelisation (multithreading) – e.g. OpenMP  
a memory area usable from all threads  
each thread can have private memory

*private memory* parallelisation – e.g. MPI  
no shared memory - All memory private  
communication by explicit messages

Try to debug a *serial* version of the program first!

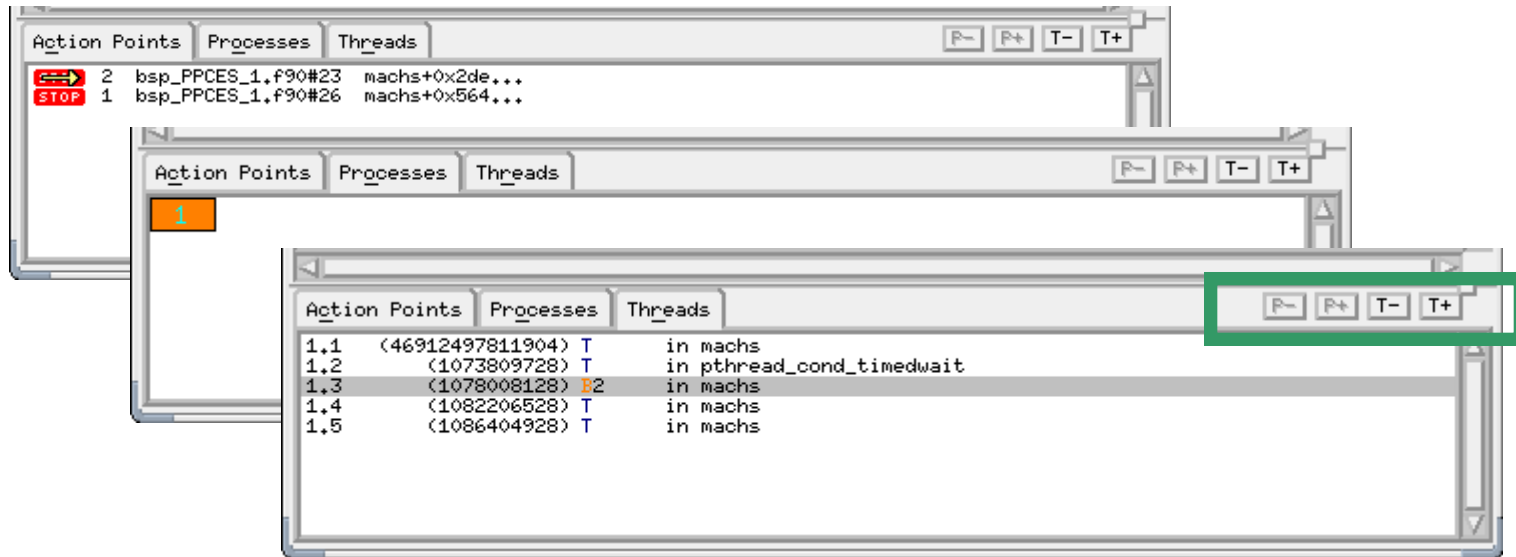
Many typical multithreaded errors may not (or not comfortable) be found with a debugger (for example *race condition*) → use threading tools!

Nevertheless, TV is better than no TV

Start as usual

Switching between Threads: T+/T- keys, Root Window

Note the Tabbed Pane on bottom of Main Window



*Action Points* tab: e.g. Breakpoints

*Processes*: all active processes (no MPI: only one)

*Threads*: all currently threads

(you may see some system threads as well as threads you defined)

# Thread Status and Navigation

Right-click in  
Root Window  
→ *Dive*  
or  
*Dive in New Window*

Process 1 (24259): a.out (At Breakpoint 2)

Thread 1 (46912497811904) (At Breakpoint 2)

Stack Trace

Stack Frame

Function "machs":

1179403647 (0x464c457) widerhol:

<Bad address: 0x0016a> maxiter:

20000000 (0x01312d00) maxiter:

Block "#b1":

1970169159 (0x756e654) j:

i:

var:

j:

var:

j:

Function machs in bsp\_PPCES\_1.f90

22 WRITE (\*,\*) "-----", i, maxiter

23 !\$OMP PARALLEL DO DEFAULT(SHARED)

24 DO j = 1, maxiter

25 var(j) = SIN(REAL(j + i))

26

27

28 END DO

29 !\$OMP END PARALLEL DO

30 END DO

31 END SUBROUTINE machs

Action Points Processes Threads

1.1 (46912497811904) B2 in machs

1.3 (1078008128) T in \_\_clone

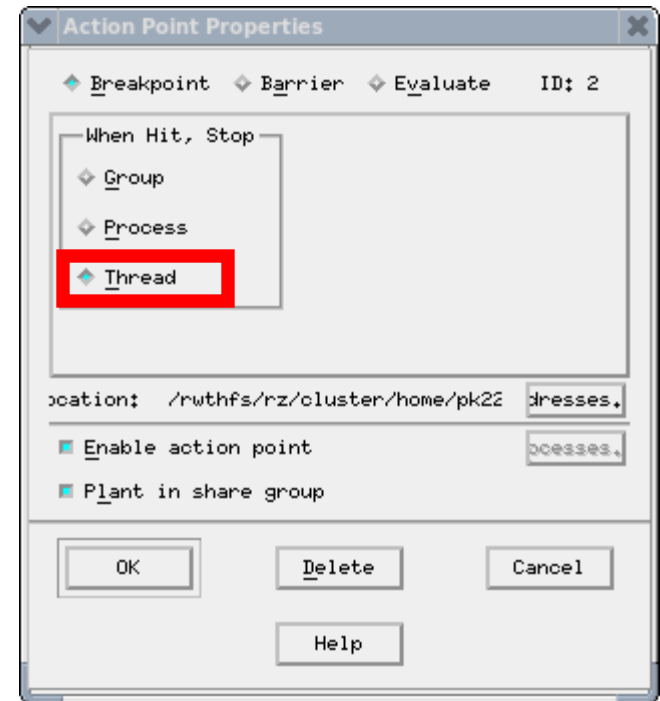
1.4 (1082206528) T in \_\_clone

1.5 (1086404928) T in \_\_clone

# Breakpoint properties

Default: same breakpoint to all threads  
right-click on Breakpoint → “Properties”

- “When hit, Stop”
  - **Group** – stop all threads in all processes if one hit the barrier (useful for MPI debugging)
  - **Process** – stop all threads of a process (Default)
  - **Thread** – stop only the thread which hit the barrier (other threads continue running and probably hits the barrier later)



# Looking at Variables across Threads

Let's see values of a variable in all threads

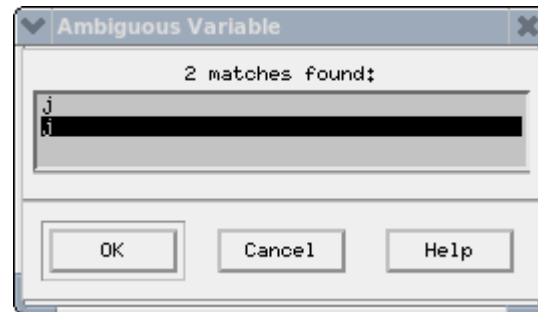
Right-click on a variable and “Across Threads”

Shared vars are equal

Private vars are  
sometimes “ambiguous”

→ Pick the right one!

```
20 var = 3.14
21 DO i = 1, widerhol
22   WRITE (*,*) "----- ", i, maxiter
23   !$OMP PARALLEL DO DEFAULT(SHARED)
24   DO j = 1, maxiter
25
26     var(j) = SIN(REAL(j + i))
27
28   END DO
29 !$OMP END PARALLEL DO
30 END DO
```



Thread	Value
1.1 (46912497811904)	1 (0x00000001)
1.2 (1073809728)	<Has no matching call frame>
1.3 (1078008128)	51 (0x00000033)
1.4 (1082206528)	101 (0x00000065)
1.5 (1086404928)	151 (0x00000097)

Why use a debugger?

Before you start

Starting TotalView

Basic debugging

Multithreaded (OpenMP) debugging

MPI debugging

Advanced

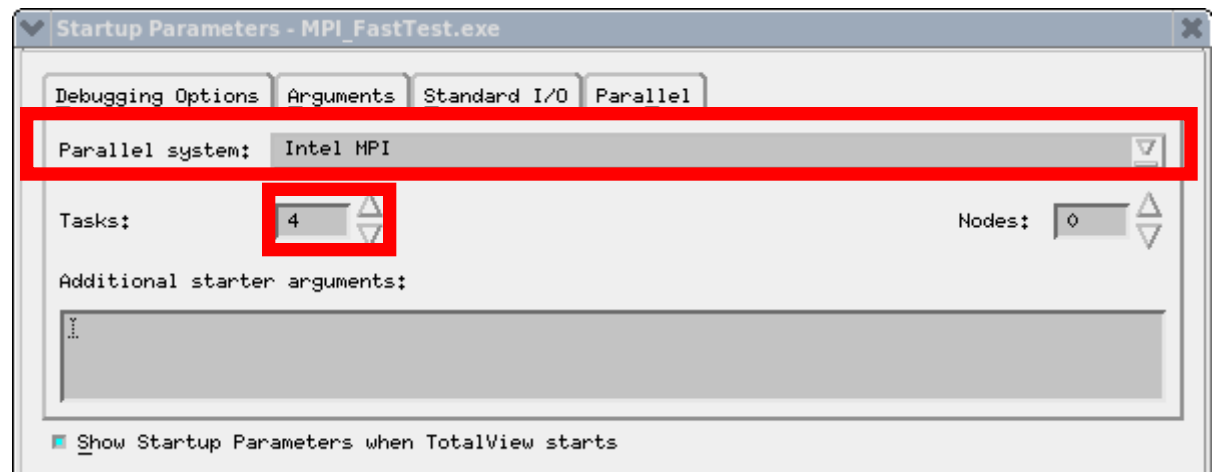
Memory debugging

Replay Engine



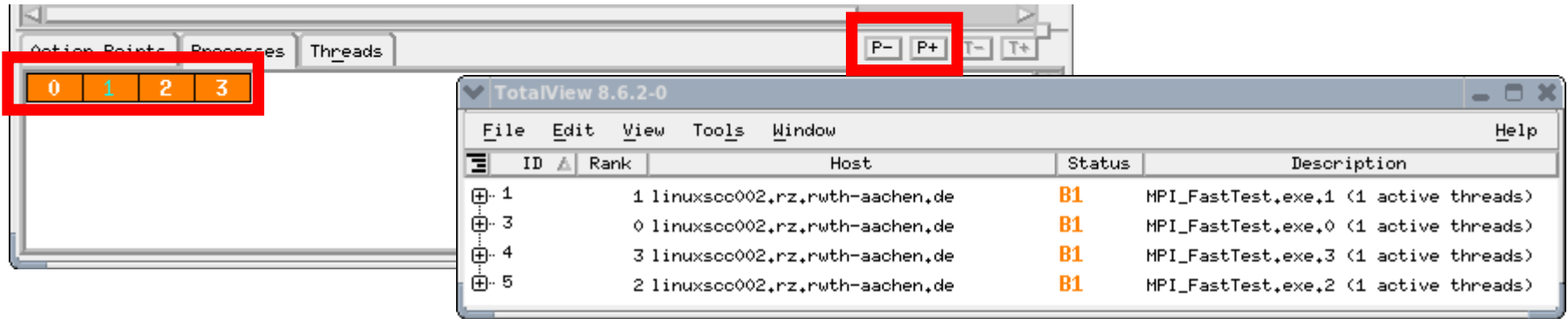
# Starting MPI debugging

New Launch: “\$ totalview mpi-a.out” and do settings  
Easy, intuitive but can’t detach and re-attach  
Not available on all platforms



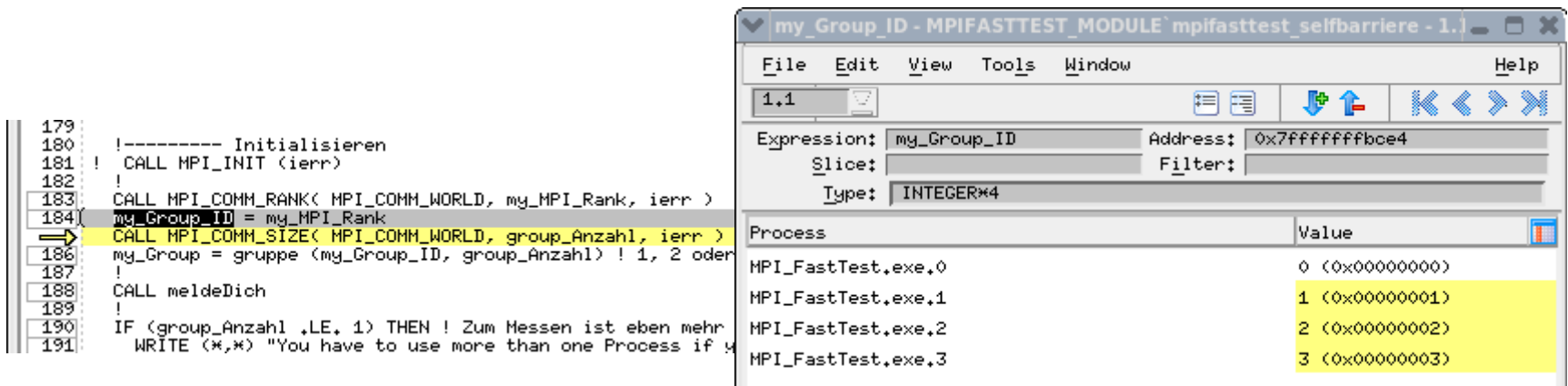
Classic Launch (the only way on BlueGene)  
e.g. Intel and Oracle MPI: `mpiexec -tv -np 16 MPI_FastTest.exe`  
Arguments depend on MPI vendor  
Attach/Attach to subset/Detach/Reattach possible  
Warning: don’t work if combinig Oracle MPI and Intel compiler

# Navigating and Looking at Variables



Let's see values of a variable in all processes:

Right-click on a variable and “Across Processes”



# View MPI Message Queues

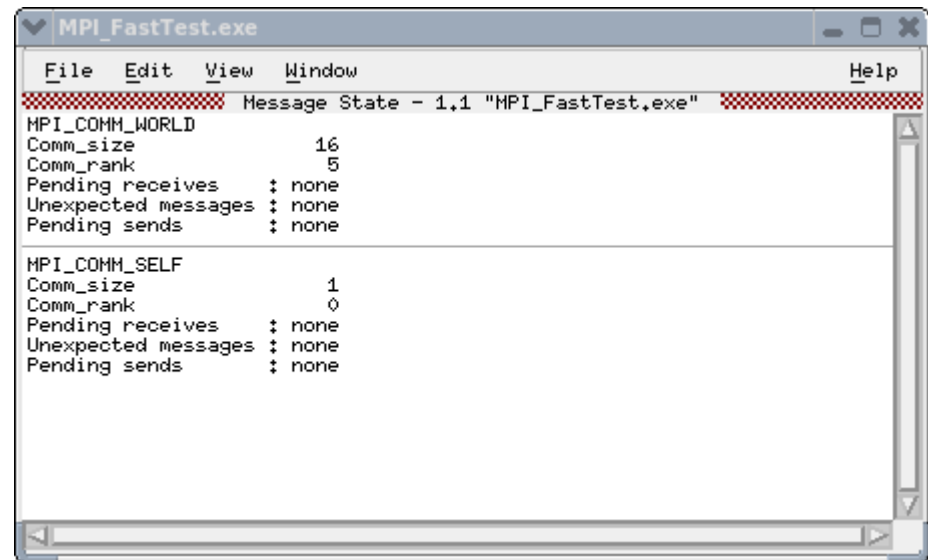
*“Tools” → “Message Queue”*

Provides information from the MPI layer

Unexpected messages

Pending Sends

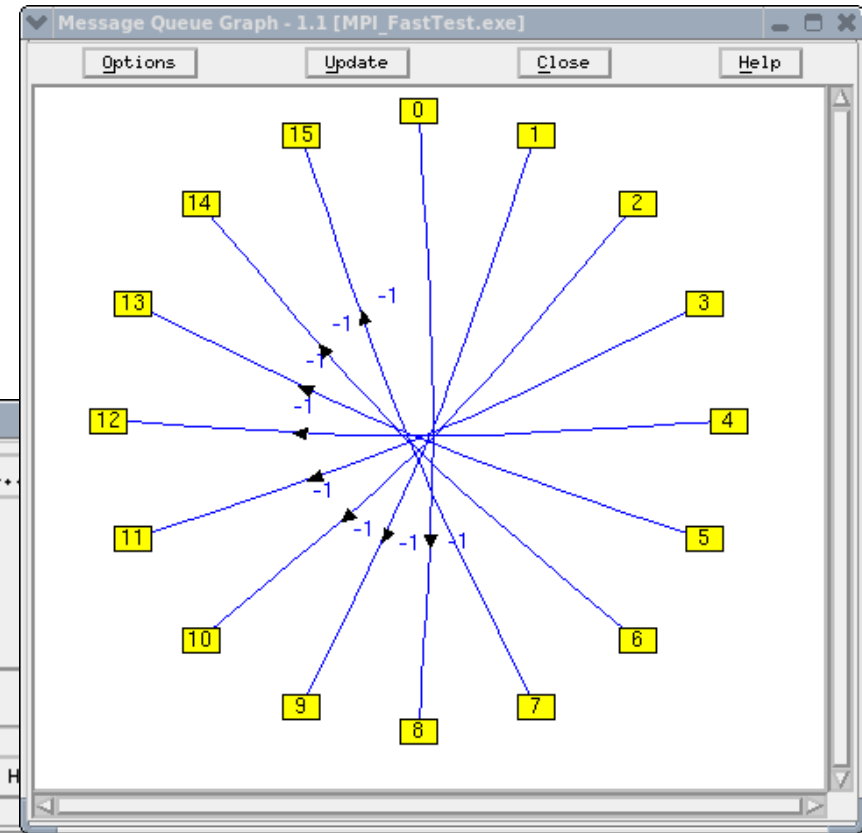
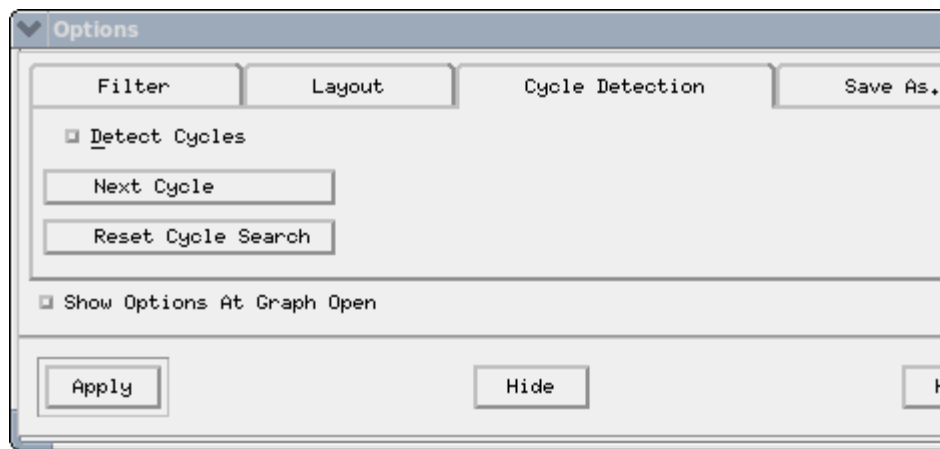
Pending Receives



*“Tools” → “Message Queue Graph”*

Hangs & Deadlocks  
Pending Messages  
Communication Patterns  
Find deadlocks:

*“Options” → „Cycle Detection“*



Why use a debugger?

Before you start

Starting TotalView

Basic debugging

Multithreaded (OpenMP) debugging

MPI debugging

Advanced

Memory debugging

Replay Engine

# Memory Debugging

---

Memory Bug: a mistake in the management of heap memory

- Memory Leak

- Double Free

- Overrun Array Bounds

TV nests in-between the program and malloc API

No instrumentation needed, low overhead

*Debug → Enable Memory Debugging*

*Debug → Memory Debugging Window*

Leak detection: Conservative Garbage Collection

Guard blocks: Memory corruption

Memory Reports, Consumption, Comparisons...

# Replay Engine

Based on “record and deterministic replay”

Records the changes to program state as they happen.

Replay the program’s previous execution.

*Prev.* Step backward over functions

*UnStep:* Step backward into functions

*Caller:* Advance backward out of current function,  
to before the call

*BackTo:* Advance backward to selected line

*Live:* Advance forward to “live” session



Using a debugger enhance productivity

TotalView helps you!

- Serial programs

- Threaded (OpenMP) programs

- MPI programs

- Hybrid programs

Last but not least: every program has errors...

- ... TotalView, too

- ... so don't panic if something doesn't work :-)



TotalView Users Manual PDF Format

[www.totalviewtech.com/pdf/other/TotalView\\_User\\_Guide.pdf](http://www.totalviewtech.com/pdf/other/TotalView_User_Guide.pdf)

TotalView Reference Guide PDF Format

[http://www.totalviewtech.com/pdf/other/totalview\\_reference\\_guide.pdf](http://www.totalviewtech.com/pdf/other/totalview_reference_guide.pdf)

TotalView Video Tutorials

<http://www.totalviewtech.com/support/videos.html>

TV Tech Developer Forum     <http://forum.totalviewtech.com/>

Demo Licenses and Product Downloads

<http://www.totalviewtech.com/download/index.html>

# The end

---

Thank you for your attention!

Questions?