

Debugging with TotalView



Tim Cramer

cramer@rz.rwth-aachen.de

Why to 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.

 - Debuggers will enhance your productivity.

What is TotalView?

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

■ TotalView is a GUI-based Debugger

- I don't like slides with one screenshot after the other
- I want to do this as interactive as possible
- If you want: Start your Laptops and login
- Login to RWTH Compute Cluster using the NX client
- All examples should be distributed in the hpclabXX accounts
- Slides for interactive sessions will be online with screenshots

Before you start

- **Lean back and relax**

- **Check your environment**

 - increase ulimits (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!**

TotalView in the RWTH Environment

■ Initialize the environment and startup:

→ `$ module load totalview`

→ `$ totalview`

■ or load a binary directly (here called a.out):

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

■ Main modes:

→ Start a new process

→ Attach to a running process

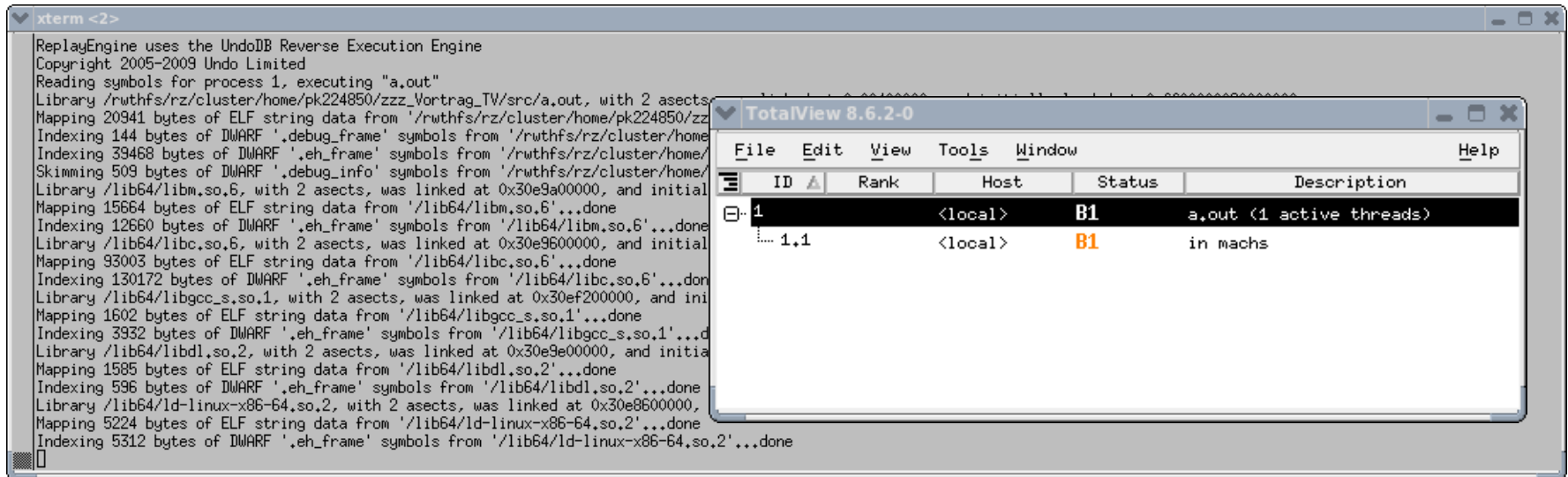
→ Load a core file (Post-Mortem)

The screenshot shows the TotalView Process Window for a.out. The window title is 'a.out'. The menu bar includes File, Edit, View, Group, Process, Thread, Action Point, Debug, Tools, Window, and Help. The toolbar contains icons for Go, Halt, Kill, Restart, Next Step, Out, Run To, Prev UnStep, Caller, and BackTo Live. A status bar at the top shows 'Process 1 (24077): a.out (At Breakpoint 1)' and 'Thread 1 (46912497811904) (At Breakpoint 1)'. The main area is divided into several panes:

- Stack Trace Pane:** Shows a list of stack frames with addresses and function names: f90 machs, f90 bsp_ppces_1, main, __libc_start_main, and _start.
- Stack Frame Pane:** Displays details for the current frame 'machs', including frame pointers (FP), local variables (widerhol, maxiter, i, j), and registers for the frame.
- Source Pane:** Shows the source code for the function 'machs' in 'bsp_PPCEs_1.f90'. Line 19, 'ALLOCATE(var(maxiter))', is highlighted.
- Tabbed Pane:** Shows a list of action points, with '1 bsp_PPCEs_1.f90#20 machs+0x178' selected.

Callouts with arrows point to the following elements:

- Toolbar:** Points to the toolbar icons.
- Process and Thread Status:** Points to the status bar at the top.
- Stack Trace Pane:** Points to the stack trace list.
- Stack Frame Pane:** Points to the details of the current stack frame.
- Source Pane:** Points to the source code editor.
- Tabbed Pane:** Points to the action points list.

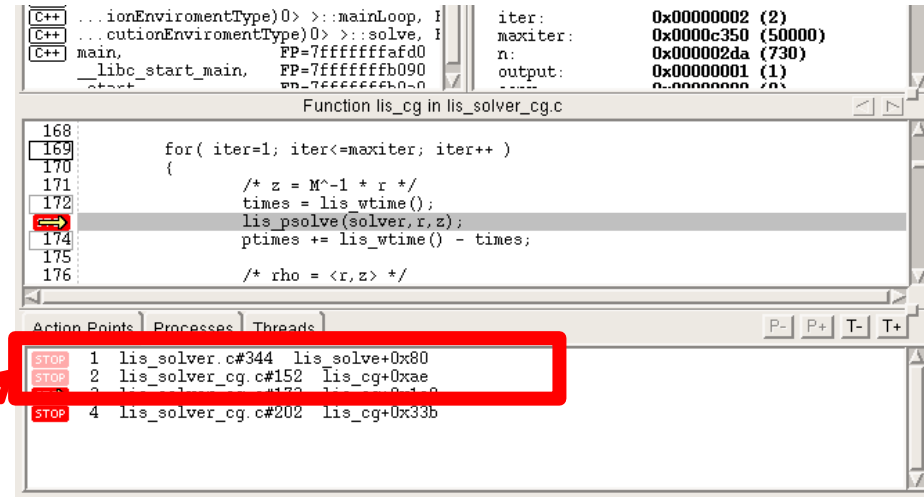


Status Info

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

■ Breakpoints (just presented)

- Interrupt execution when reaching a specific code line
- Conditional Breakpoints possible
- Set by clicking in the source pane
- Temporary disabling is possible

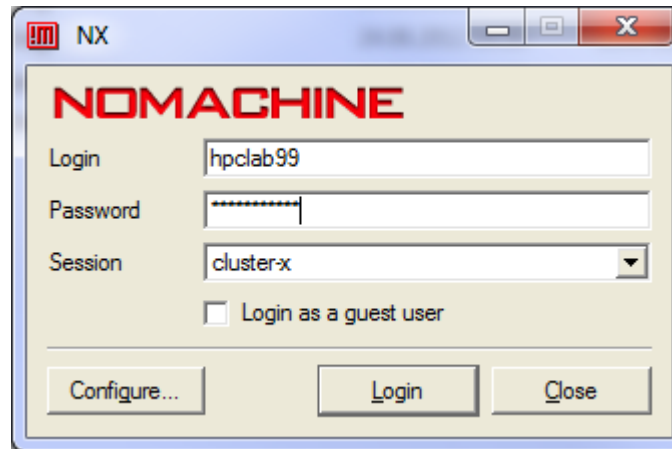


■ Watchpoints

- Interrupt when a change occurs to a specific memory location
- Conditional watchpoints possible
(e.g. only stop if the sign of the value changes or specified threshold reached)

Live Demo Startup

L1: Login with NX & Startup TotalView



```
Terminal
File Edit View Search Terminal Help
hpclab99@cluster-x:~[509]$ cd totalviewlabs/livedemo/l1_basics/
hpclab99@cluster-x:~/totalviewlabs/livedemo/l1_basics[510]$ module load totalvie
w
Loading totalview 8.9.2-2 [ OK ]
hpclab99@cluster-x:~/totalviewlabs/livedemo/l1_basics[511]$ totalview
```

L1: TotalView Window

The screenshot displays the TotalView 8.9.2-2 interface. On the left, a terminal window shows the following text:

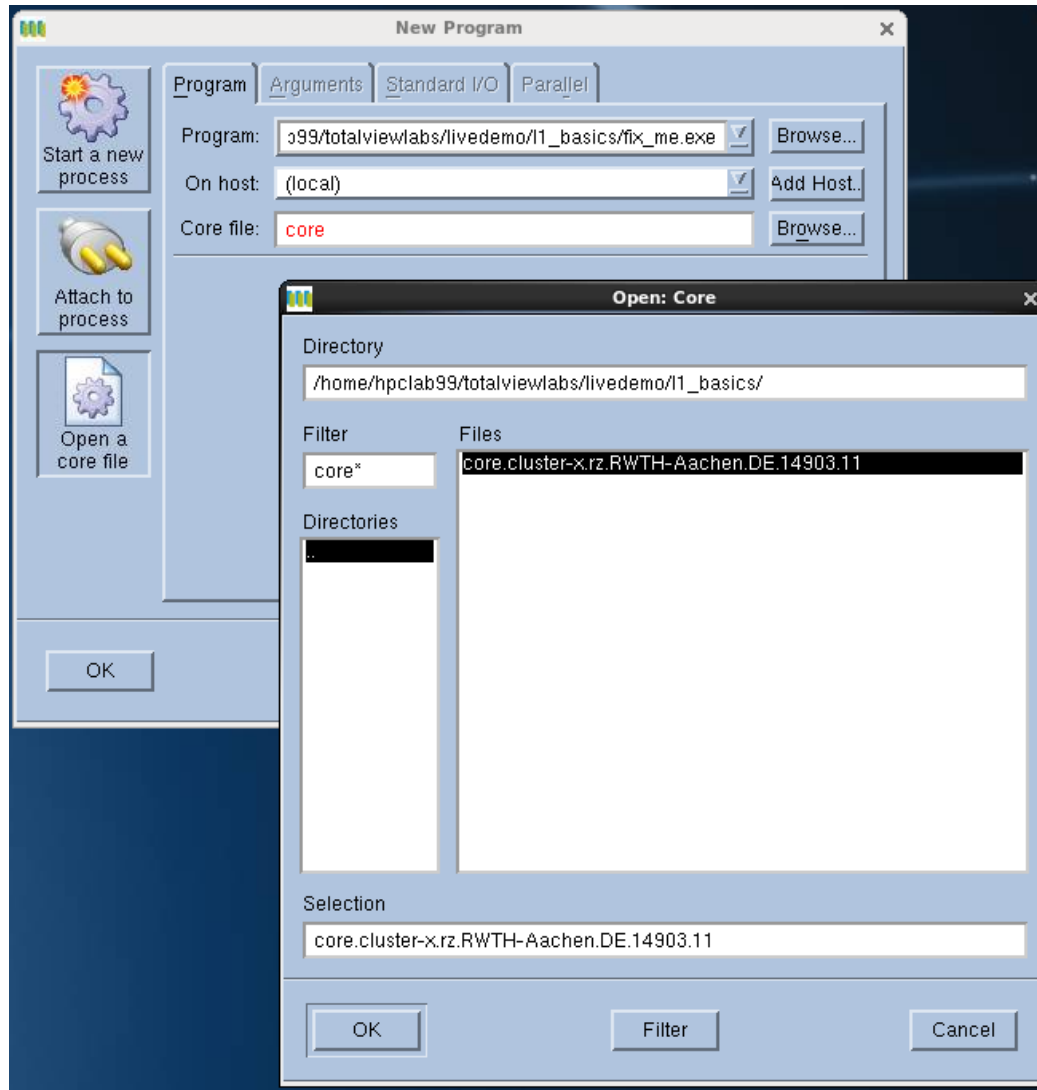
```
hpclab99@cluster-x:~[509]$ cd totalviewlabs/livedemo/l1 basics/  
hpclab99@cluster-x:~/totalviewlabs/livedemo/l1  
w  
Loading totalview 8.9.2-2  
hpclab99@cluster-x:~/totalviewlabs/livedemo/l1  
Linux x86_64 TotalView 8.9.2-2  
Copyright 2010-2012 by Rogue Wave Software Inc.  
Copyright 2007-2010 by TotalView Technologies,  
Copyright 1999-2007 by Etnus, LLC.
```

On the right, the 'New Program' dialog box is open, showing the 'Program' tab. The 'Program' field is empty, and the 'On host' field is set to '(local)'. The 'Standard I/O' and 'Parallel' tabs are also visible. The dialog includes several checkboxes for debugging options:

- Enable ReplayEngine
Record all program state while running. Roll back your progr:
- Enable memory debugging
Track dynamic memory allocations. Catch common errors, leaks, and shc
 - Halt on memory errors
- Enable CUDA memory checking
Detect global memory addressing violations and misaligned g

The dialog also features a sidebar with icons for 'Start a new process', 'Attach to process', and 'Open a core file', along with 'OK', 'Cancel', and 'Help' buttons at the bottom.

L1: Post-Mortem Debugging (1/2)



L1: Post-Mortem Debugging (2/2)

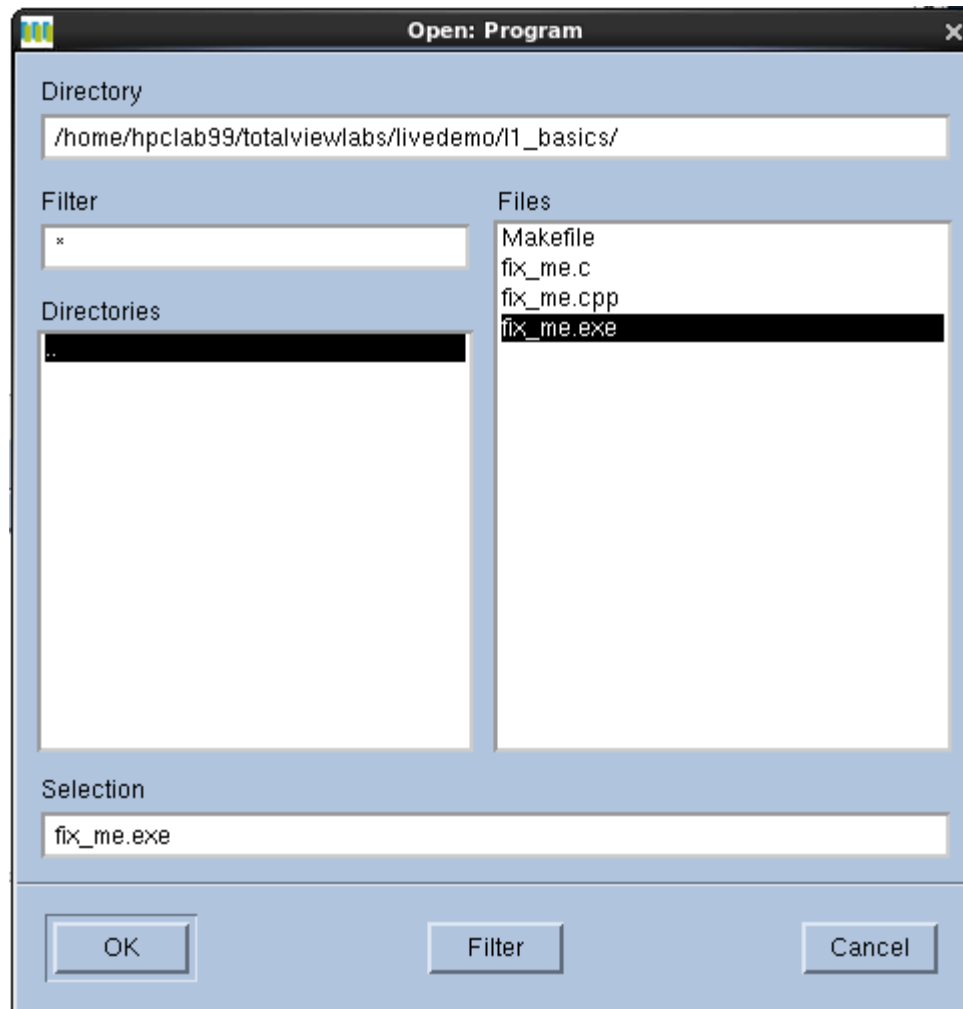
The screenshot shows the TotalView 8.9.2-2 interface. The main window displays the source code for 'fix_me.c'. The 'init' function is highlighted, and the line 'a[i] = (double)i;' is marked with a red box. The 'main' function is also visible. The 'Stack Trace' panel shows the call stack, and the 'Stack Frame' panel shows the current state of the program, including the function 'init' and its local variables. The 'Process List' panel shows the process 'fix_me.exe' with a status of 'E' (Error). The error message 'Thread 1 (13811) (Error) -<Segmentation violation:' is highlighted in red.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void init(double* a, const int count){
5     int i;
6
7     for (i=0; i<count; i++){
8         a[i] = (double)i;
9         printf("Init a[%d] with %f\n", i, a[i]);
10    }
11 }
12
13 int main(int argc, char **argv){
14     const int count = 100000;
15     double* a;
16
17     a = (double*)malloc(count);
18
19     init(a, count);
20 }
```

Function "init":
a: 0x00601010 -> 0
count: 0x000186a0 (100000)
Local variables:
i: 0x000071fe (29182)
Registers for the frame:
%rax: 0x0063a000 (6529024)
%rdx: 0x000186a0 (100000)
%rcx: 0x00000001 (1)
%ebx: 0x00000000 (0)

ID	Rank	Host	Status	Description
1		<local>	E	fix_me.exe (1 active threads)

L1: Running within TotalView

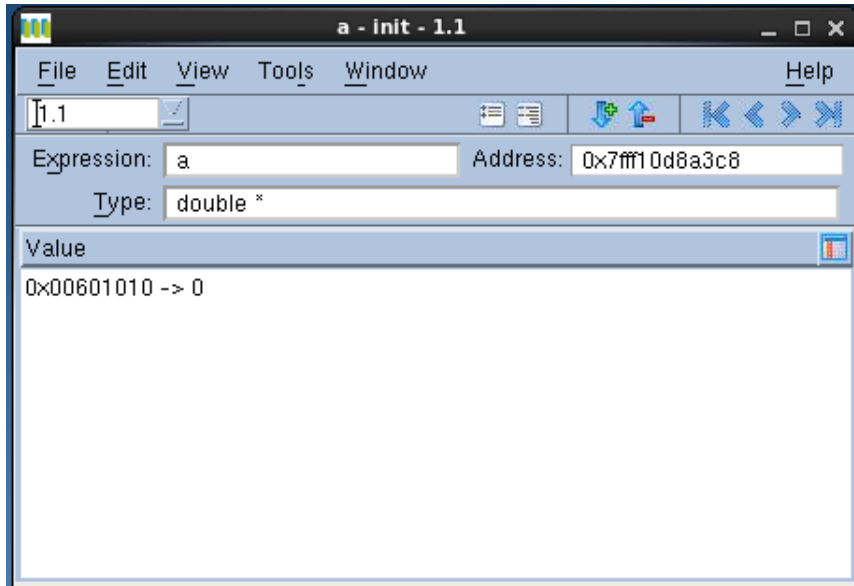


Setting a breakpoint

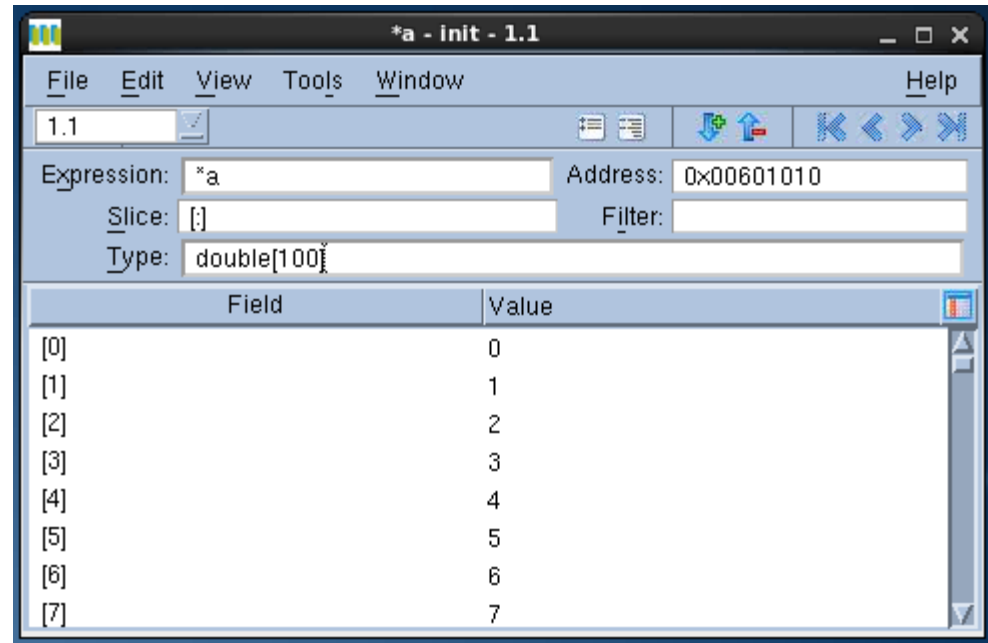
The screenshot displays the TotalView 8.9.2-2 IDE interface. The main window shows the source code for `fix_me.c` with a breakpoint set on line 19, `init(a, count);`. The Action Points panel at the bottom shows a breakpoint at `3 fix_me.c#19 main+0x30`. An inset window shows the 'TotalView 8.9.2-2' interface with a table containing one entry with status `B2`.

ID	Rank	Host	Status	Description
1		<local>	B2	/home/hpclub99/totalviewlabs/livec

Inspecting an array in C/C++



■ Typecast necessary



■ Helpful for big data arrays

The screenshot displays the TotalView debugger interface. On the left, the source code for a C program is shown, with a yellow arrow pointing to line 8: `a[i] = i;`. The code includes `<stdio.h>` and `<stdlib.h>`, and defines an `init` function and a `main` function. The `main` function allocates a double array `a` and calls `init(a, count)`.

In the center, a window titled `*a[0:99]` displays a data visualization of the array. The X-axis is labeled `X Axis` and the Y-axis is labeled `Y Axis`. Both axes range from 0.00 to 120.0. A red line represents the data, showing a linear relationship where the value at index `i` is `i`.

On the right, the `Stack Frame` window shows the current stack frame. Below it, a window titled `*a - init - 1.1` displays the memory dump for the array `a`. The expression is `*a`, the address is `0x00601010`, and the type is `double[100]`. The memory dump shows the following values:

Field	Value
[0]	0
[1]	1
[2]	2
[3]	3
[4]	4
[5]	5
[6]	6
[7]	7
[8]	8

Live Demo Watchpoints

■ Create a watchpoint for a[29]

The screenshot displays the TotalView debugger interface. The main window shows the source code of `main.c` with a breakpoint set at line 19, `init(a, count);`. A secondary window titled `a - main - 1.1` shows the watchpoint configuration for the expression `*(a)` at address `0x00601010`. The watchpoint is configured with a slice of `[:]` and a type of `double[1000]`. A table below shows the values of memory locations from `[27]` to `[36]`, all of which are `0`. A context menu is open over the `[29]` entry, with the `Create Watchpoint` option highlighted in red. The bottom status bar shows the current action point: `2 fix_me.c#19 main+0x30`.

Will interrupt as soon as a [29] changes

The screenshot shows the TotalView debugger interface. The main window displays the source code of `fix_me.c` with a watchpoint set on line 8, `a[i] = (double)i;`, where `i` is 29. The watchpoint is active, as indicated by the `STOP` icon and the `5 8 bytes @ 0x006010f8 ((a) [29])` entry in the Action Points pane.

The Stack Trace pane shows the following call stack:

- `init, FP=7fff89532800`
- `main, FP=7fff89532840`
- `__libc_start_main, FP=7fff89532900`
- `_start, FP=7fff89532910`

The Stack Frame pane shows the current function `init` with the following expression and type:

- Expression: `*(a)`
- Address: `0x00601010`
- Type: `double[1000]`

The memory view shows the values of the array elements `a`:

Field	Value
[27]	27
[28]	28
[29]	29
[30]	0
[31]	0
[32]	0
[33]	0
[34]	0
[35]	0
[36]	0

Parallel Debugging with TotalView

■ Parallel Debugging might be very hard

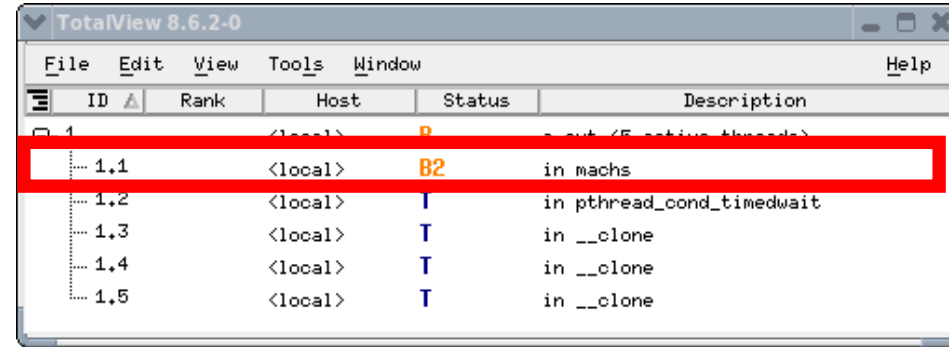
- Try to debug a *serial* version of the program first!
- Typical multithreading errors may not be found (e.g., *race conditions*)
- Some errors only occur
 - with optimized code (uninitialized variables?)
 - with many processes
 - outside of debug sessions (different timing)

■ Nevertheless, TV is better than no TV

- Stack frames for every process / thread in one GUI
- Switching between processes / threads (even for accelerators like GPGPUs)
- Variable inspection (and visualization) across all processes / threads
- Deadlock detection
- Visualization of the MPI message queue

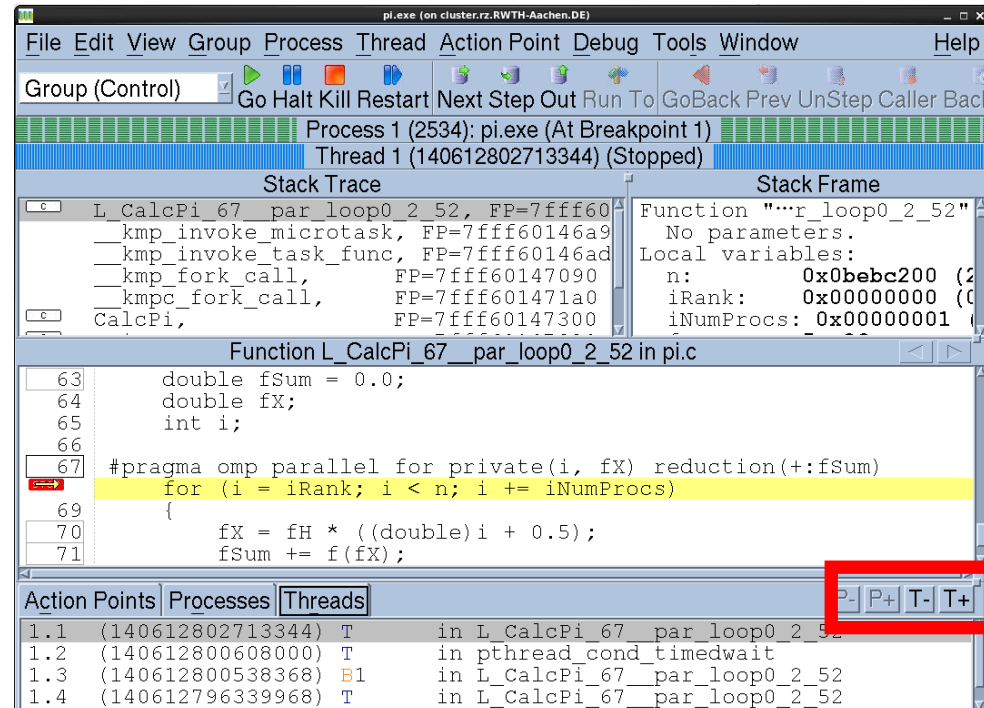
■ Overview of thread number

- Number of threads in root window
- some OpenMP implementations use an additional system thread



■ Thread state and navigation

- Step into a parallel region is not possible -> Set a breakpoint
- Easy switching between the threads with T- / T+

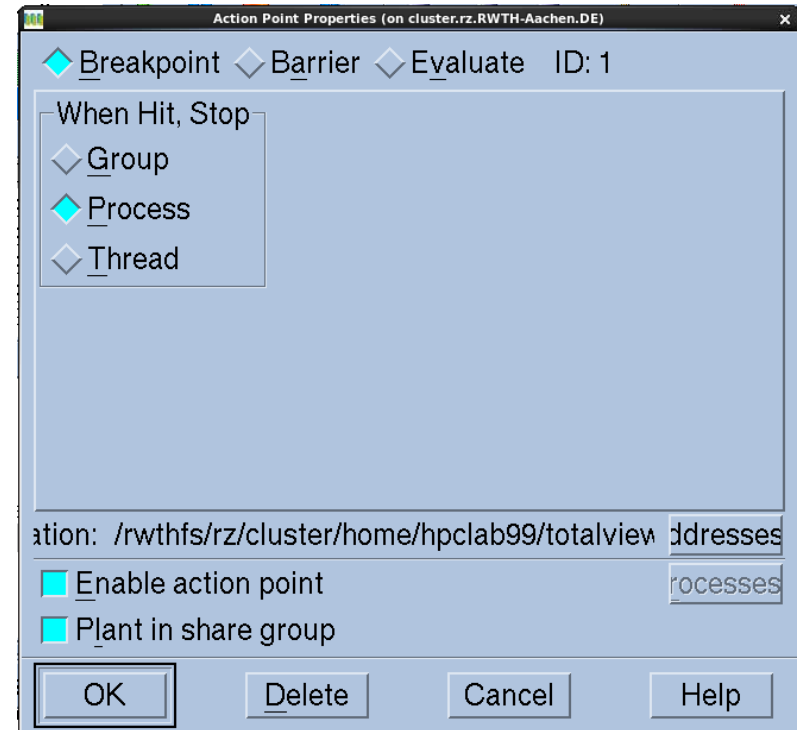


■ Breakpoint properties and barriers

→ Default behavior: Same breakpoint for all threads

→ Change “Properties”

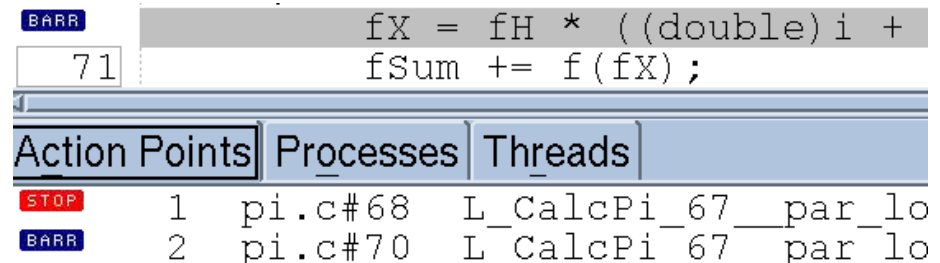
Group	Stop all threads of all processes
Process	Stop all threads of one process (default)
Thread	Stop only the current thread



→ Barriers

→ Similar to breakpoints

→ Help to synchronize



MPI and TotalView

■ Two startup methods for MPI jobs

→ New launch: `$ totalview mpi-a.out`

→ Set parameters in GUI

→ Easy and intuitive

→ No detaching or re-attaching possible

→ Not available on all platforms

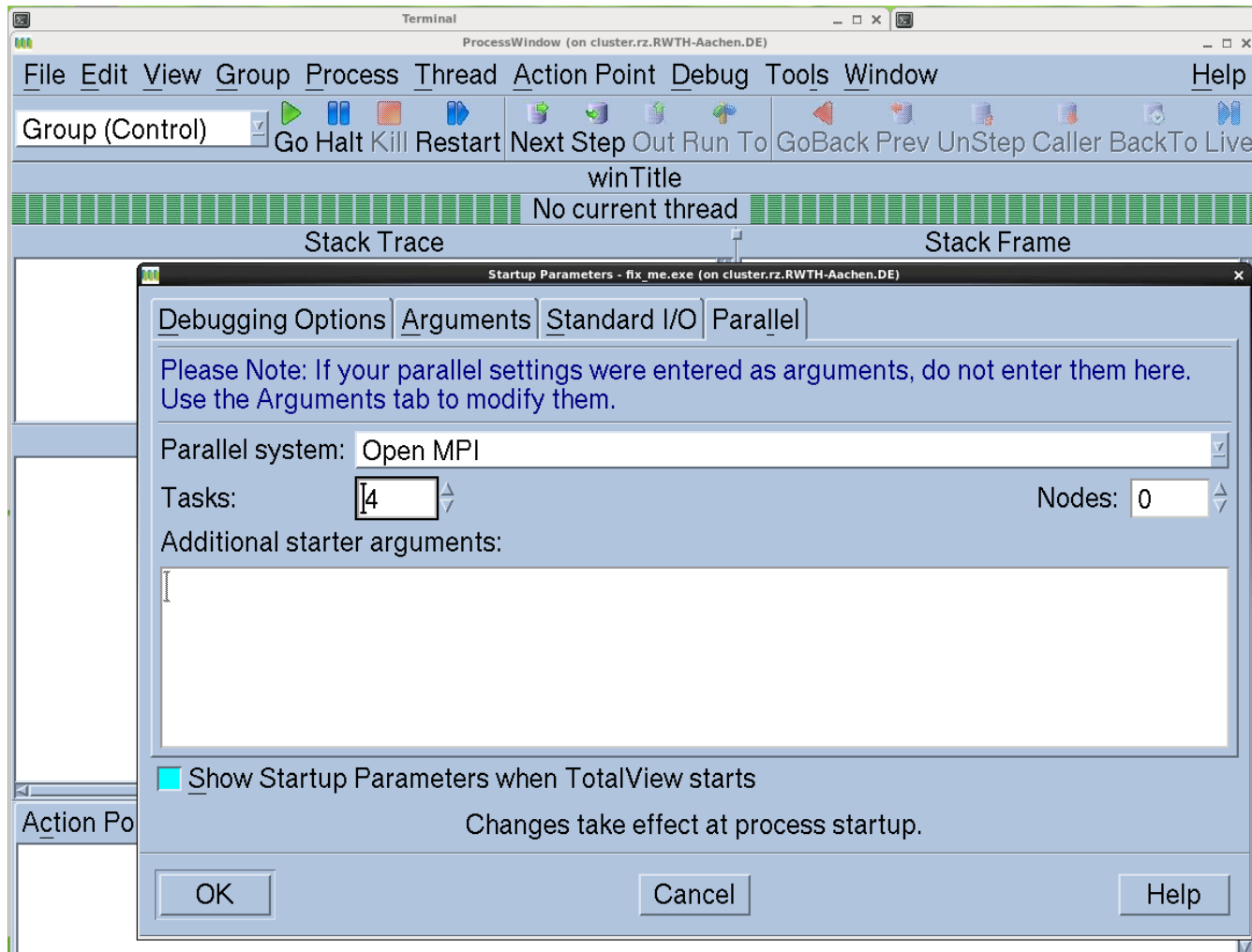
→ Classic launch: `$ mpiexec -tv -np 4 mpi-a.out`

→ Arguments depend on MPI vendor

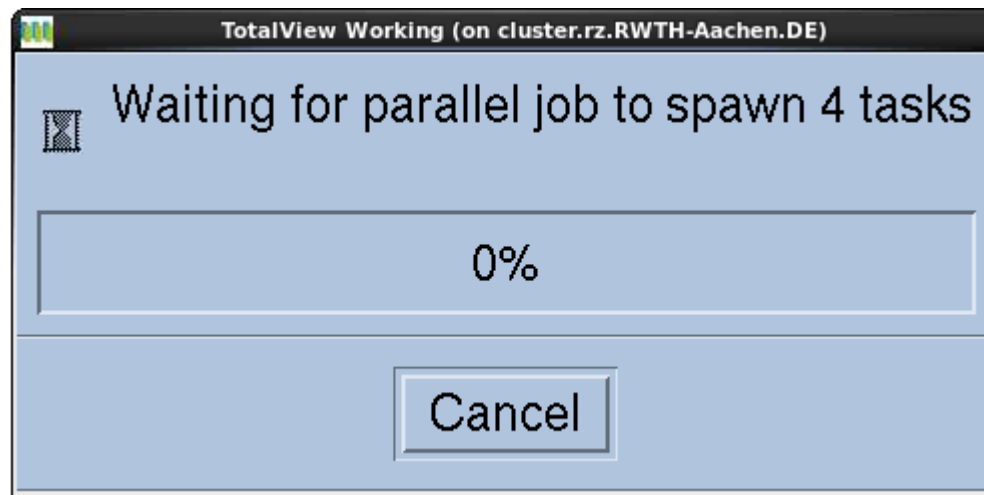
→ Attach / Attach to subset / Detache / Reattache possible

→ Warning: Not possible @ RZ at the moment

Live Demo Parallel Debugging



- Be patient



Rank 0: fix_me.exe.0 (Running)
Thread 1 (140037624768256): fix_me.exe (Running)

Stack Trace Stack Frame

Thread must be stopped for frame display.

TotalView 8.9.2-2 (on cluster.rz.RWTH-Aachen.DE)

	ID	Rank	Host	Status	Description
1		0	134.61.220.74	R	fix_me.exe.0 (3 active threads)
3		3	134.61.220.74	R	fix_me.exe.3 (3 active threads)
4		1	134.61.220.74	R	fix_me.exe.1 (3 active threads)
5		2	134.61.220.74	R	fix_me.exe.2 (3 active threads)

■ Press the “HALT” button

fix_me.exe.0 (on cluster.rz.RWTH-Aachen.DE)

File Edit View Group Process Thread Action Point Debug Tools Window Help

Group (Control) Go Halt Kill Restart Next Step Out Run To GoBack Prev UnStep Caller BackTo Live

Rank 0: fix_me.exe.0 (Stopped)

Thread 1 (140037624768256): fix_me.exe (Stopped)

Stack Trace

```
opal_progress, FP=7fff59ead5d0
mpi_request_default_wait_all, FP=7fff59ead5d0
mpi_coll_tuned_sendrecv_actual, FP=7fff59ead5d0
...d_barrier_intra_recurse_doubling,
...oll_tuned_barrier_intra_dec_fixed,
PMPI_Barrier, FP=7fff59ead750
main, FP=7fff59ead850
__libc_start_main, FP=7fff59ead910
_start, FP=7fff59ead920
```

Stack Frame

Registers for the frame:

```
%rax: 0x945e0430 (2489189424)
%rdx: 0x00015a05 (88581)
%rcx: 0x00000000 (0)
%rbx: 0x00000003 (3)
%rsi: 0x00000000 (0)
%rdi: 0x00000002 (2)
%rbp: 0x7fff59ead750 (140734701950800)
```

Function PMPI_Barrier in libmpi.so.1

```
0x7f5d0db2f60a: 0x00
0x7f5d0db2f60b: 0x4c movl %r12,%rdi
0x7f5d0db2f60c: 0x89
0x7f5d0db2f60d: 0xe7
0x7f5d0db2f60e: 0x33 xorl %eax,%eax
0x7f5d0db2f60f: 0xc0
→ 0x7f5d0db2f610: 0x41 call *376(%r12)
0x7f5d0db2f611: 0xff
0x7f5d0db2f612: 0x94
0x7f5d0db2f613: 0x24
0x7f5d0db2f614: 0x78
0x7f5d0db2f615: 0x01
0x7f5d0db2f616: 0x00
0x7f5d0db2f617: 0x00
```

Action Points Processes Threads P- P+ T- T+

Rank 0 is waiting in barrier

The screenshot shows the TotalView debugger interface for a process named 'fix_me.exe.0' on a cluster. The process is stopped at Rank 0. The main window displays the source code for 'Function main in fix_me.c'. The current execution point is at line 41, where the function is calling `MPI_Barrier(MPI_COMM_WORLD);`. This line is highlighted in yellow. The stack trace on the left shows the call stack, with 'main' at the top. The stack frame on the right shows the function's arguments and local variables, including 'argc' (1), 'argv' (pointing to memory), 'nprocs' (4), and 'myrank' (0). The code in the main window includes MPI send and receive operations, a barrier, and a data verification loop.

```
35     MPI_Send(&data, count, MPI_INT, i, tag, MPI_COMM_WORLD);
36   }
37 }
38 else {
39     MPI_Recv(&data, count, MPI_INT, 0, tag2, MPI_COMM_WORLD, &status);
40 }
41 MPI_Barrier(MPI_COMM_WORLD);
42
43 /* Check the data everywhere. */
44
45
46     if (myrank != 0){
47     for (i = 0; i < count; i++){
48     if (data[i] != 399) {
49     good_data = 0;
50     }
51     }
52     if (good_data == 0){
```

Rank 1 still waits for data

The screenshot shows the TotalView debugger interface for a process named 'fix_me.exe.1'. The process is stopped at the 'main' function. The stack trace shows the following frames:

Stack Trace	Stack Frame
mca_pml_ob1_recv, FP=7fff987f87b0	Function "main":
PMPI_Recv, FP=7fff987f8800	argc: 0x00000001 (1)
main, FP=7fff987f8910	argv: 0x7fff987f89f8 -> 0x7fff987fa157 -> "fix"
__libc_start_main, FP=7fff987f89d0	Local variables:
_start, FP=7fff987f89e0	nprocs: 0x00000004 (4)
	myrank: 0x00000001 (1)

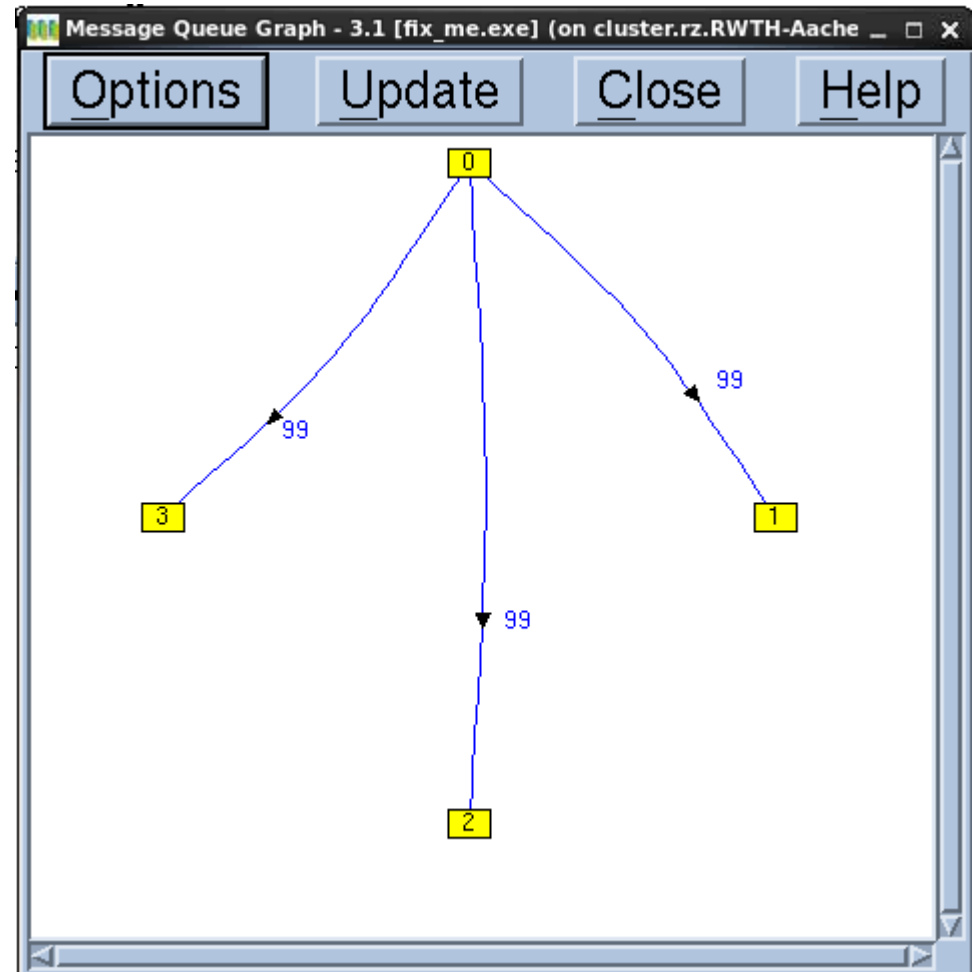
The source code for the 'main' function is displayed below. The current execution point is at line 39, where 'MPI_Recv' is called. The code is as follows:

```
30     } else
31     {
32         data[i] = i;          /* initialize all other data buffers from 0 to count */
33     }
34     if (myrank == 0) {
35         for (i = 1; i < nprocs; i++) {
36             MPI_Send(&data, count, MPI_INT, i, tag, MPI_COMM_WORLD);
37         }
38     } else {
39         MPI_Recv(&data, count, MPI_INT, 0, tag2, MPI_COMM_WORLD, &status);
40     }
41     MPI_Barrier(MPI_COMM_WORLD);
42     /* Check the data everywhere. */
43     for (i = 0; i < count; i++) {
44         if (myrank != 0) {
45             for (j = 0; j < count; j++) {
46                 if (data[j] != i) {
47                     printf("Data mismatch at rank %d, location %d\n", myrank, j);
48                 }
49             }
50         }
51     }
52 }
```

L3: MPI Debugging

■ Message Queue Graph

→ Rank 1, 2, 3 are waiting
for data on tag 99



Debugging of Large MPI Jobs

■ Limitations

- Each MPI process consumes a TotalView license token
- RWTH has only **50** licenses
- Try to debug a small example

■ Debugging of subset of the whole job

→ Attaching to subset possible

"File" -> "Preferences" -> "Parallel"

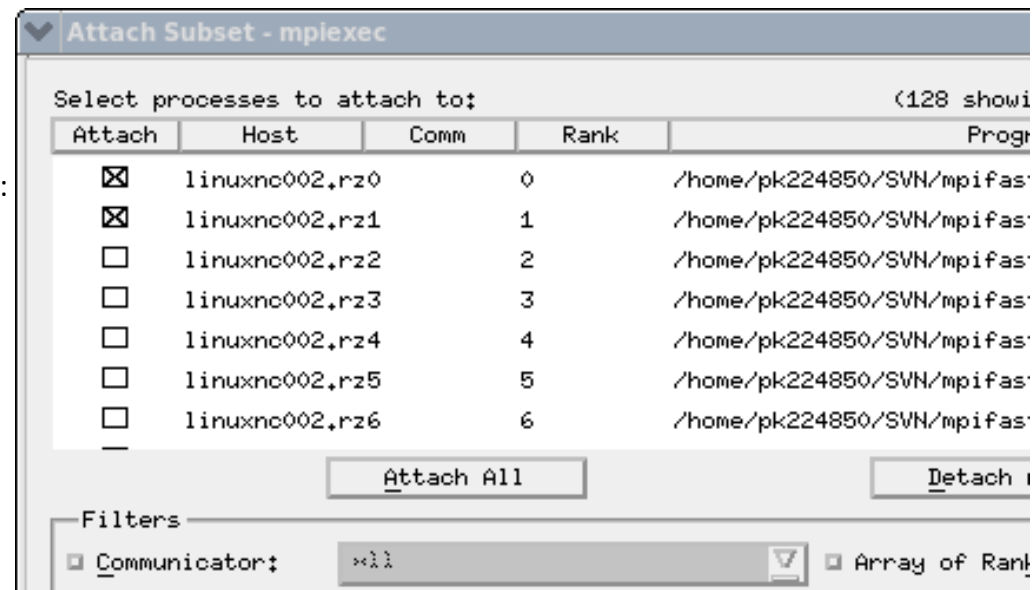
"When a job goes parallel" menu set:

[x] on "Ask what to do"

(instead of "Attach to all")

Choose a subset of processors in

"Group" -> "Attach Subset"



Message Queue Graph

“Tools” → “Message Queue Graph”

Hangs & Deadlocks

Pending Messages

Communication Patterns

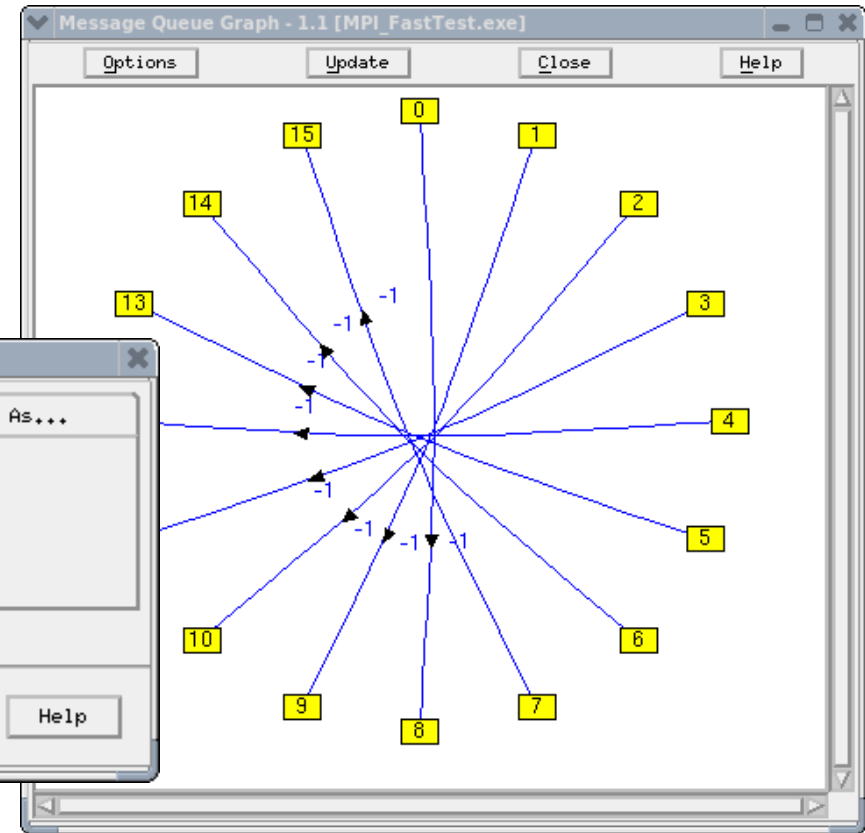
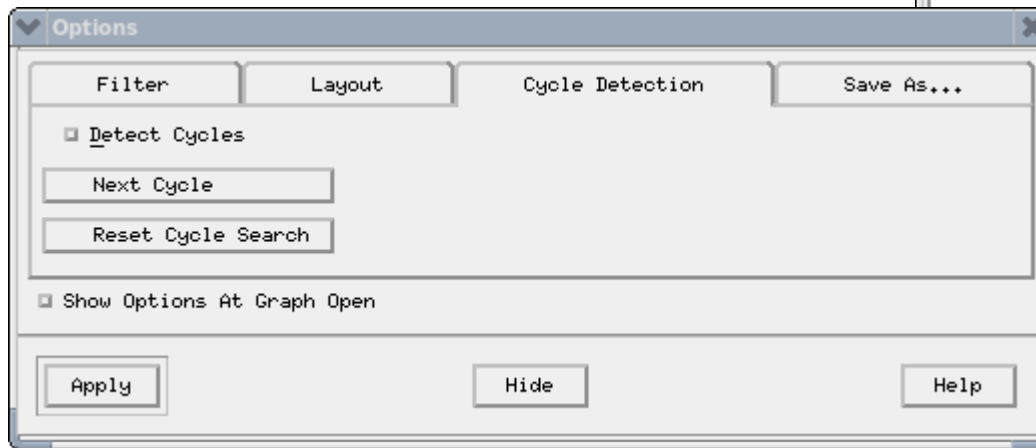
Find deadlocks:

“Options” → „Cycle Detection“

Green: Pending Sends

Blue: Pending Receives

Red: Unexpected Messages



■ Debugging for

→ Memory leaks
(Leak detection)

→ Double free

→ Invalid array
bounds

■ Memory reports

→ Consumption

→ Comparisons

→ ...

MemoryScape 3.2.0-0

File Tools Window Help

Home Memory Reports Manage Processes Memory Debugging Options Tips

Summary Leak Detection Heap Status Memory Usage Corrupted Memory Memory Comparisons

March 20, 2011

Save Data
Export Memory Data...

Heap Status Reports
Source Report
Backtrace Report

Other Reports Categories
Leak Detection Reports
Memory Usage Reports
Corrupted Memory Report
Compare Memory Usage

Other Tasks
Manage Filters

Process Selection
Process iMOOSE.stat2d (6739)

Heap Status Graphical Report

Options
 Detect Leaks Relative to Baseline Enable Filtering

Leaked Block

Heap Information Backtrace/Source Memory Content

Category	Bytes
Heap	
Deallocated	1231.29
Allocated	919.17
Leaked	12
Unfiltered	12
Filtered	
Hoarded	

Property	Value
Start Address	0x0070a4f0
End Address	0x0070a61f
Size	304
Type	Leaked
Filtered	No
Backtrace ID	175
Allocator	C
Owner	C

Category
Backtrace ID 175
Allocated
Corrupted Guard Block
Leaked
Deallocated
Guard Blocks
Hoarded

Summary

■ Using a debugger enhances productivity

■ TotalView helps you!

- Serial programs
- Threaded (OpenMP) programs
- MPI programs
- Hybrid programs
- GPGPUs

■ Online information

<http://www.roguewave.com/support/product-documentation/totalview.aspx#totalview>

<http://www.rz.rwth-aachen.de/hpc/primer>

Thank you for your attention!

Questions?

We have prepared several exercises.