

# Debugging with TotalView



Tim Cramer  
17.03.2016

# 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**

- **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!**

## ■ Initialize the environment and startup:

→ `$ module load totalview`

→ `$ totalview`

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

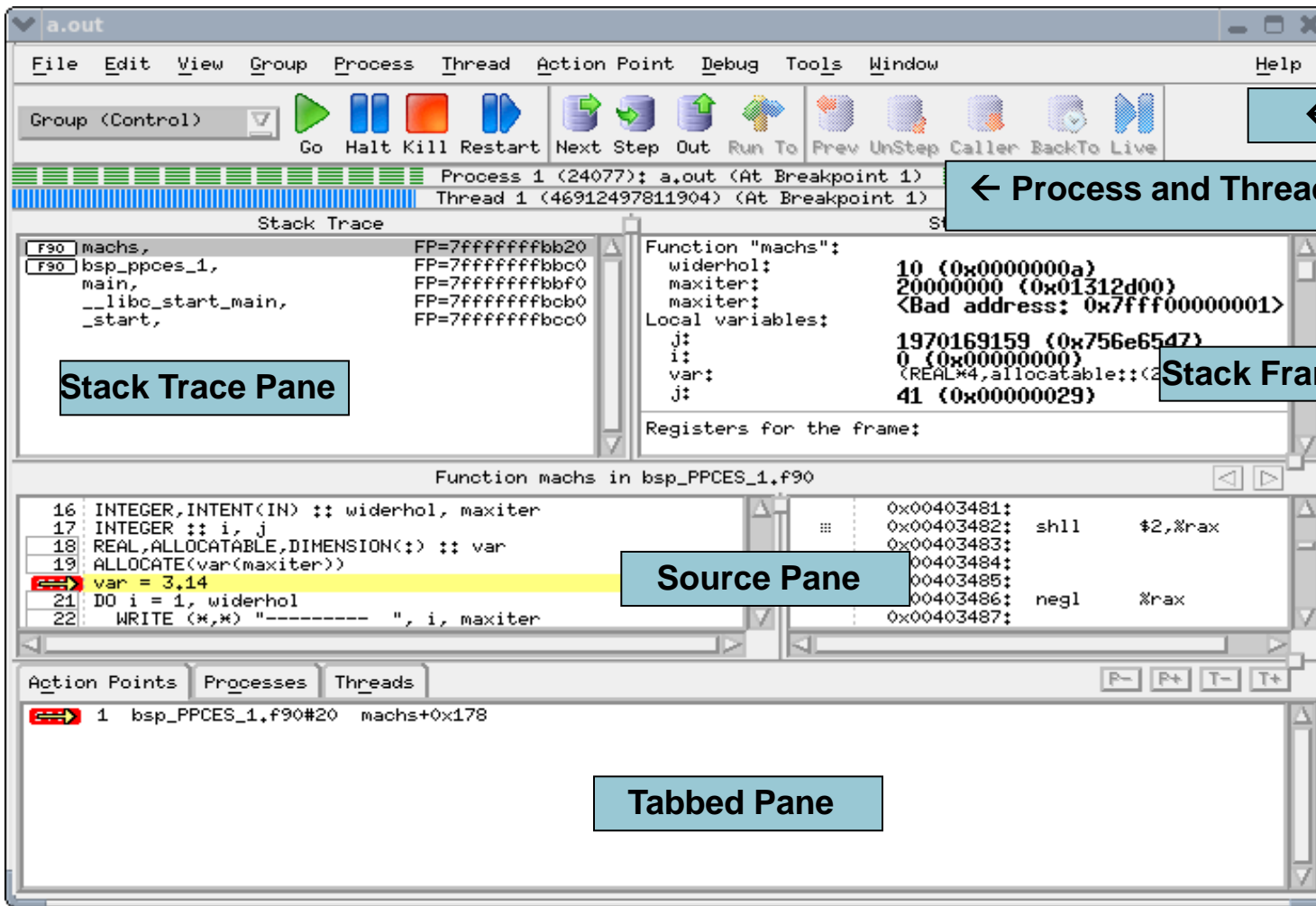
→ `$ totalview a.out -a <options of a.out>`

## ■ Main modes:

→ Start a new process

→ Attach to a running process

→ Load a core file (Post-Mortem)



← Toolbar

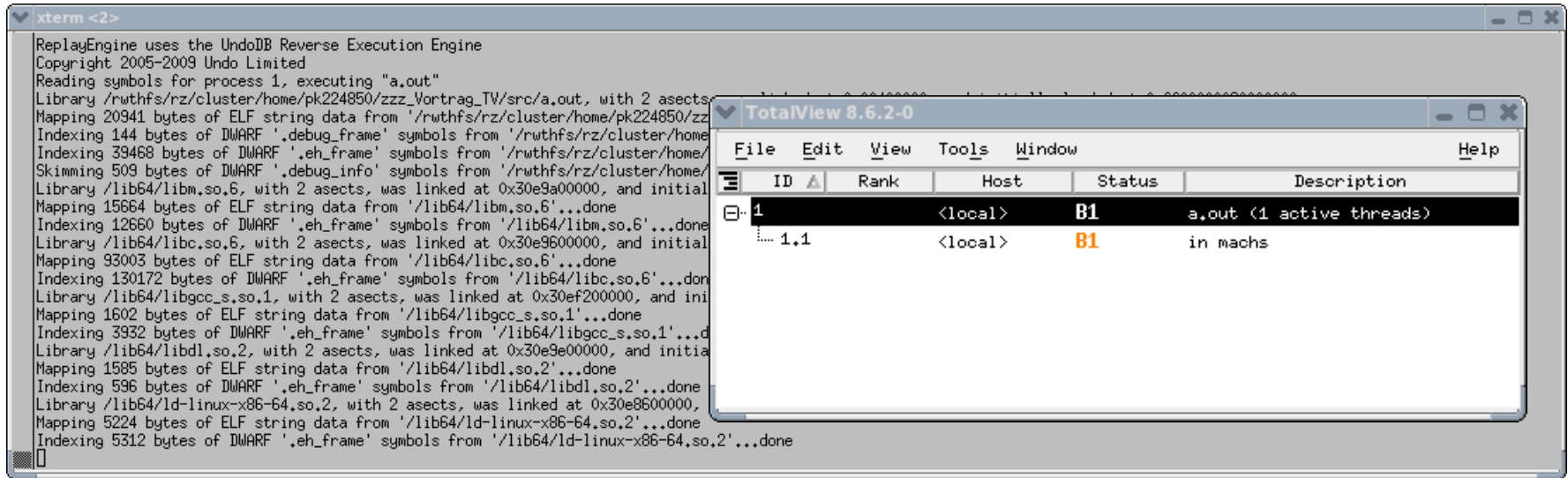
← Process and Thread Status

Stack Trace Pane

Stack Frame Pane

Source Pane

Tabbed Pane



## Status Info

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

## ■ 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

## ■ 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`

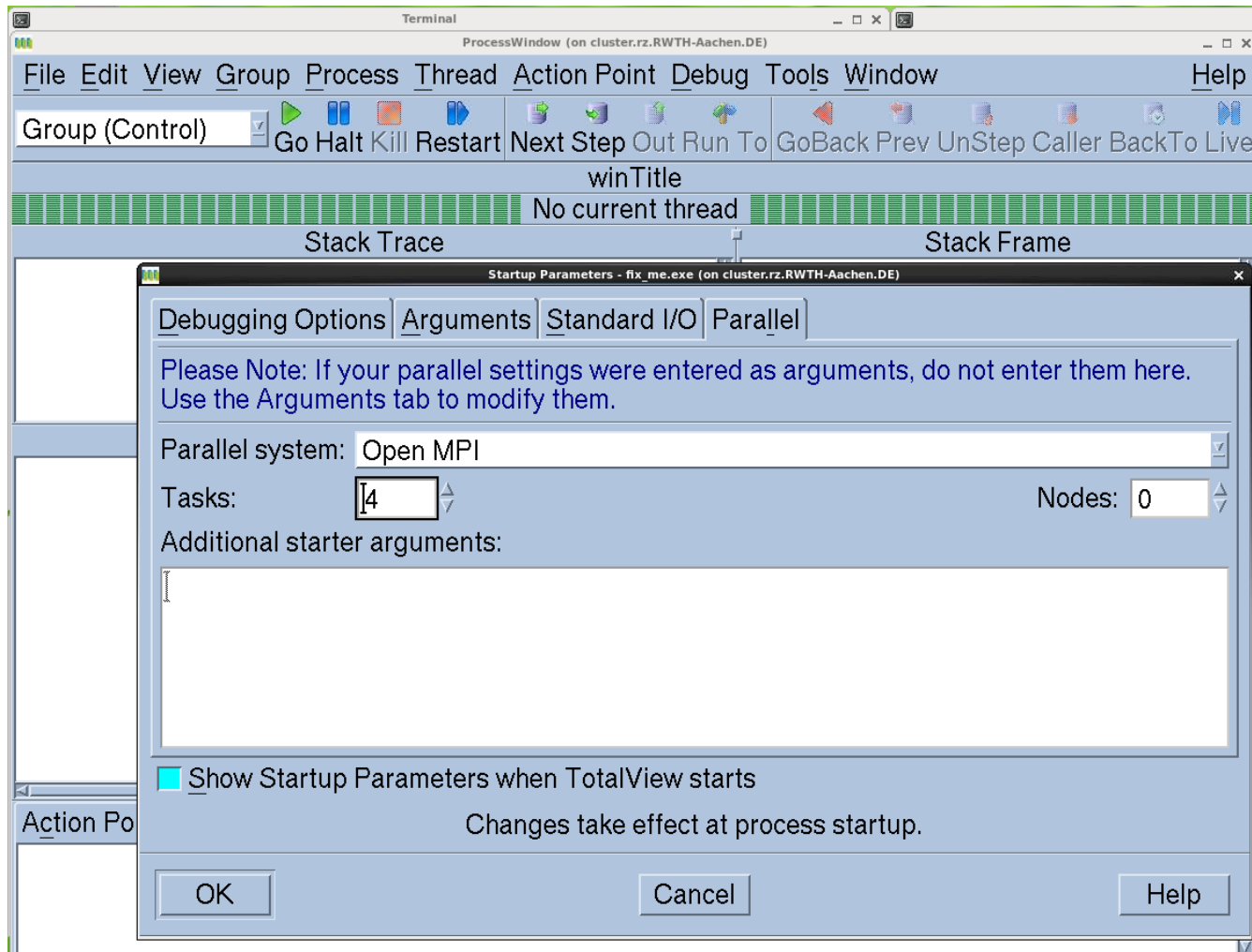
→ Arguments depend on MPI vendor

→ Attach / Attach to subset / Detache / Reattache possible

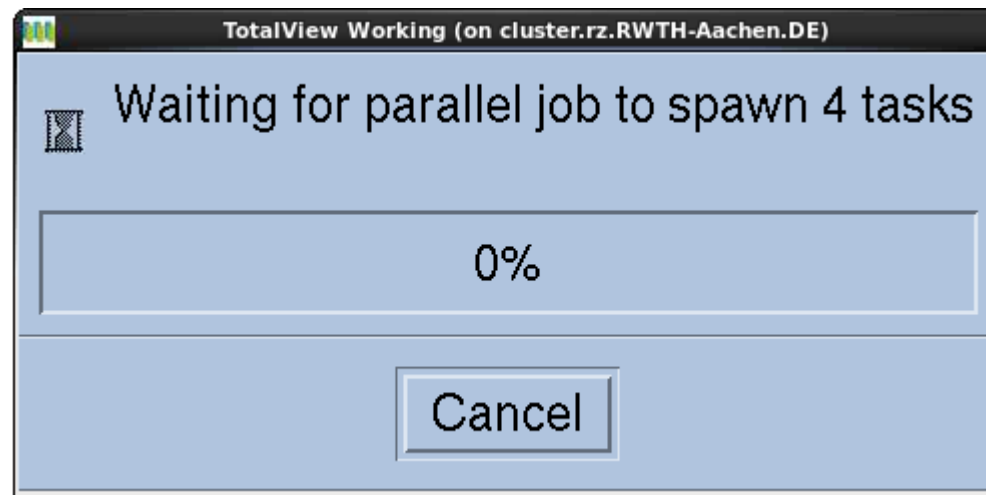




# Live Demo Parallel Debugging



- Be patient



Rank 0: fix\_me.exe.0 (Running)  
Thread 1 (140037624768256): fix\_me.exe (Running)

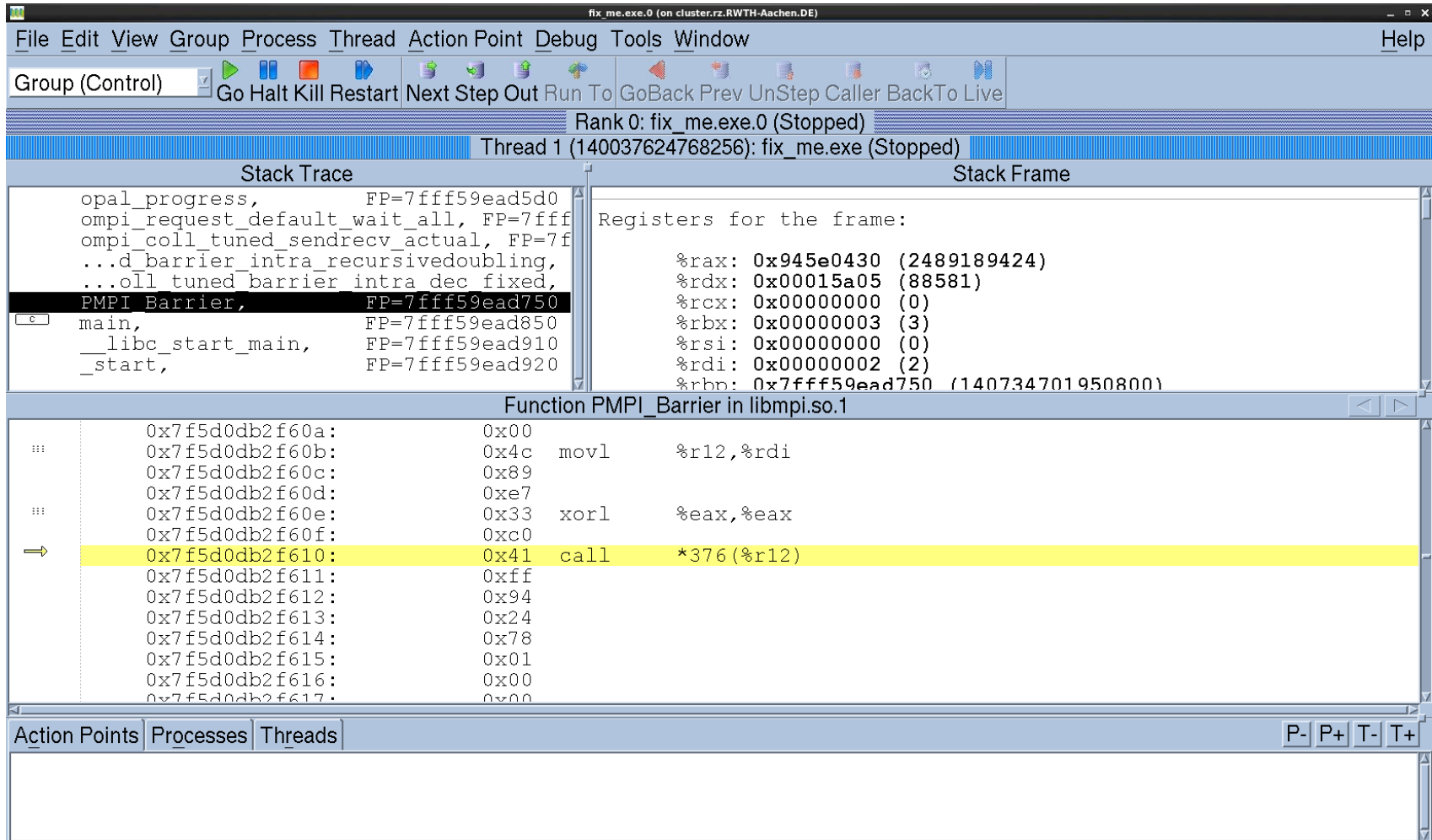
Stack Trace Stack Frame

Thread must be stopped for frame display.

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

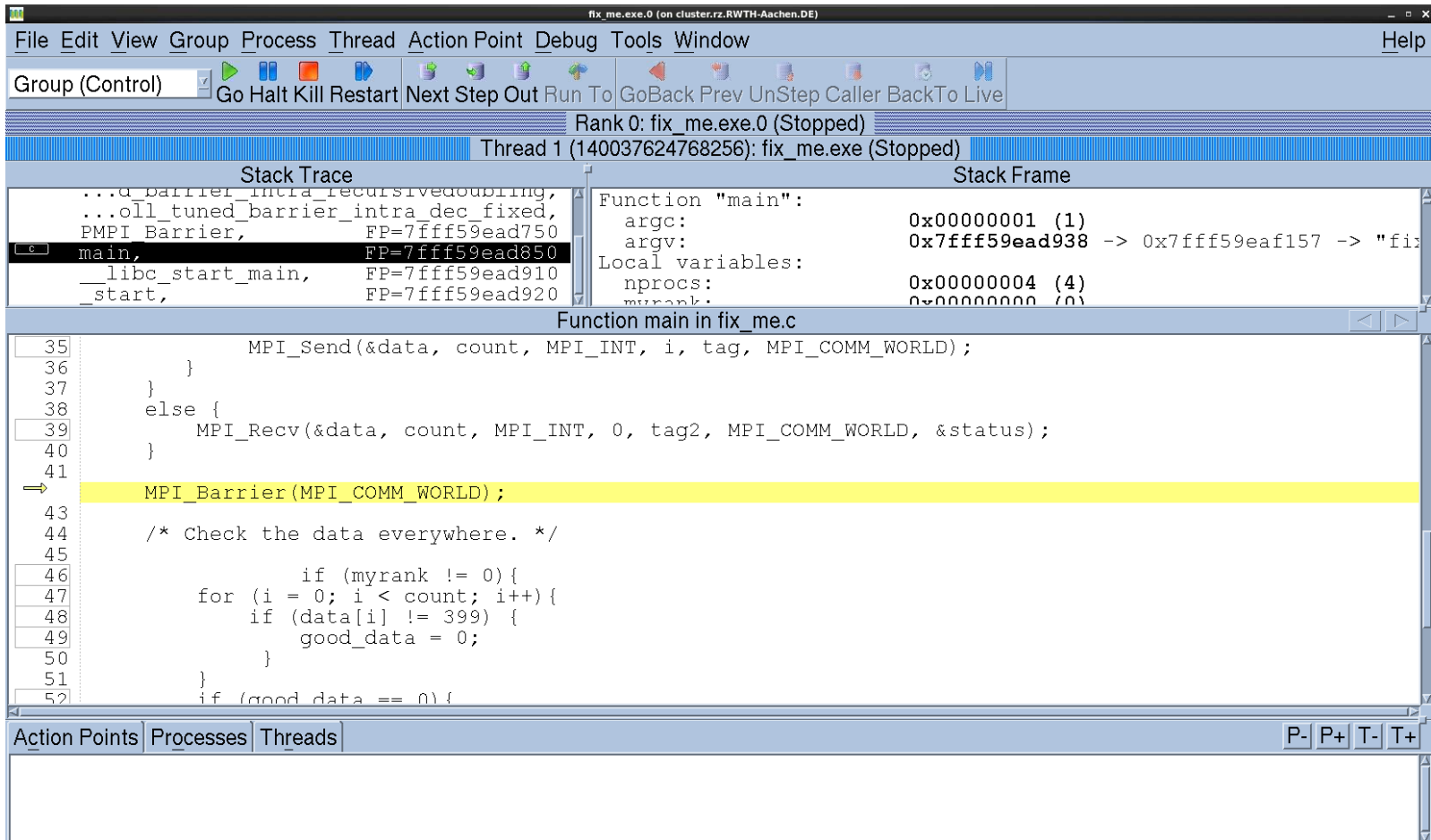
	ID	Rank	Host	Status	Description
1		0	134.61.220.74	R	fix_me.exe.0 (3 active threads)
3		3	134.61.220.74	R	fix_me.exe.3 (3 active threads)
4		1	134.61.220.74	R	fix_me.exe.1 (3 active threads)
5		2	134.61.220.74	R	fix_me.exe.2 (3 active threads)

## ■ Press the “HALT” button



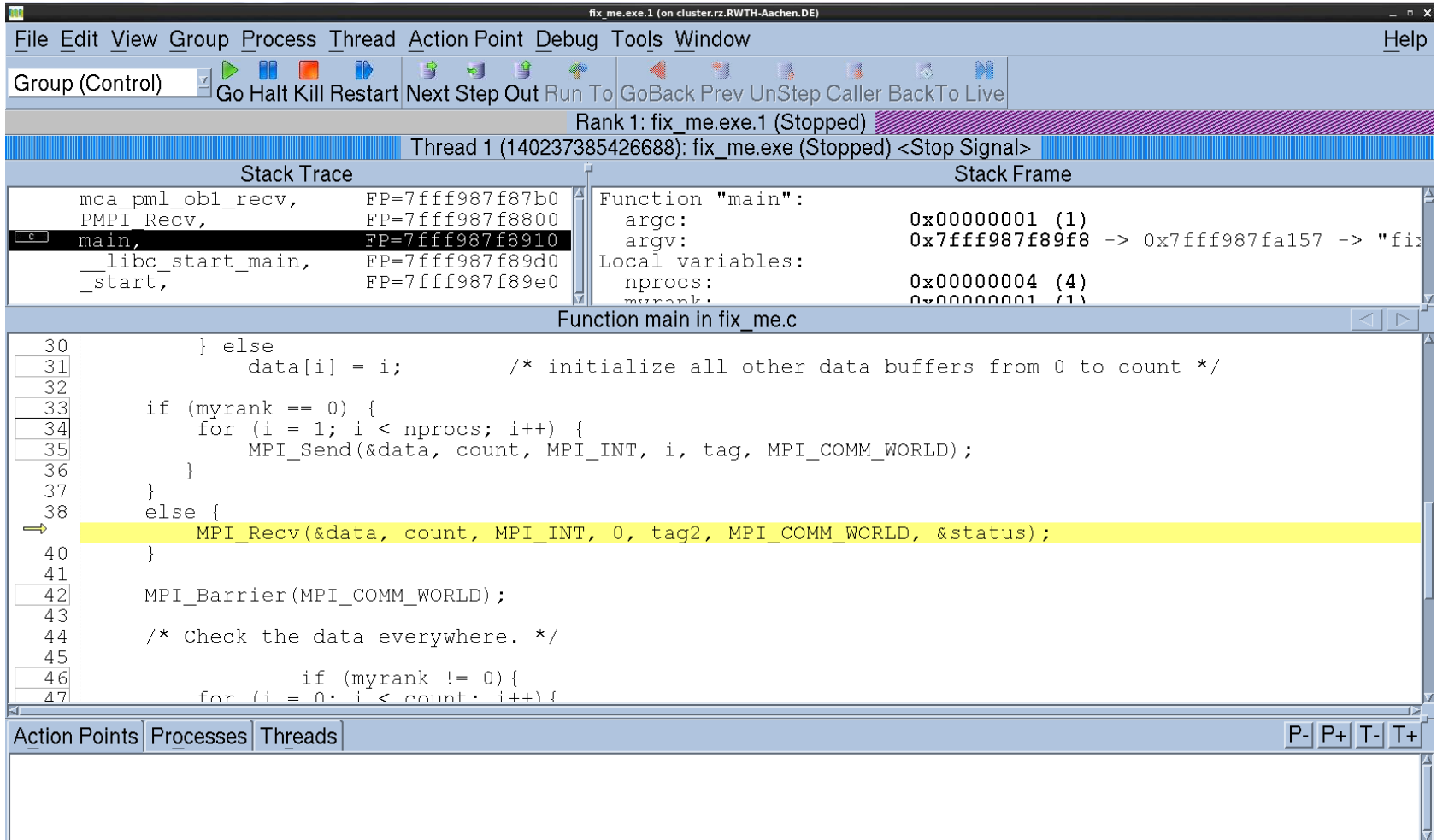
The screenshot shows the TotalView debugger interface for a process named 'fix\_me.exe.0' on a cluster at RWTH Aachen University. The process is in a 'Stopped' state. The 'Stack Trace' pane on the left shows the call stack, with the current frame being 'PMPI\_Barrier' in 'libmpi.so.1'. The 'Registers for the frame' pane on the right shows the values of various registers, including %rax (0x945e0430), %rdx (0x00015a05), %rcx (0x00000000), %rbx (0x00000003), %rsi (0x00000000), %rdi (0x00000002), and %rbp (0x7fff59ead750). The 'Function PMPI\_Barrier in libmpi.so.1' pane at the bottom shows the assembly code for the function, with the instruction 'call \*376(%r12)' highlighted in yellow. The interface includes a menu bar (File, Edit, View, Group, Process, Thread, Action Point, Debug, Tools, Window), a toolbar with various debugging actions (Go, Halt, Kill, Restart, Next Step, Out, Run To, Go Back, Prev, UnStep, Caller, Back To, Live), and a status bar at the bottom with 'Action Points', 'Processes', 'Threads', and navigation buttons (P-, P+, T-, T+).

## Rank 0 is waiting in barrier



The screenshot shows the TotalView debugger interface for the application `fix_me.exe.0` on a cluster at RWTH Aachen University. The process is stopped at Rank 0. The main window displays the source code for `main` in `fix_me.c`, with the `MPI_Barrier(MPI_COMM_WORLD);` line highlighted in yellow. The current execution point is at line 41. The stack trace on the left shows the call stack, with `main` at the top. The stack frame on the right shows the function `main` with arguments `argc: 0x00000001 (1)` and `argv: 0x7fff59ead938`. The local variables section shows `nprocs: 0x00000004 (4)` and `myrank: 0x00000000 (0)`. The bottom of the window shows the 'Action Points' tab with 'Processes' and 'Threads' sub-tabs.

## Rank 1 still waits for data



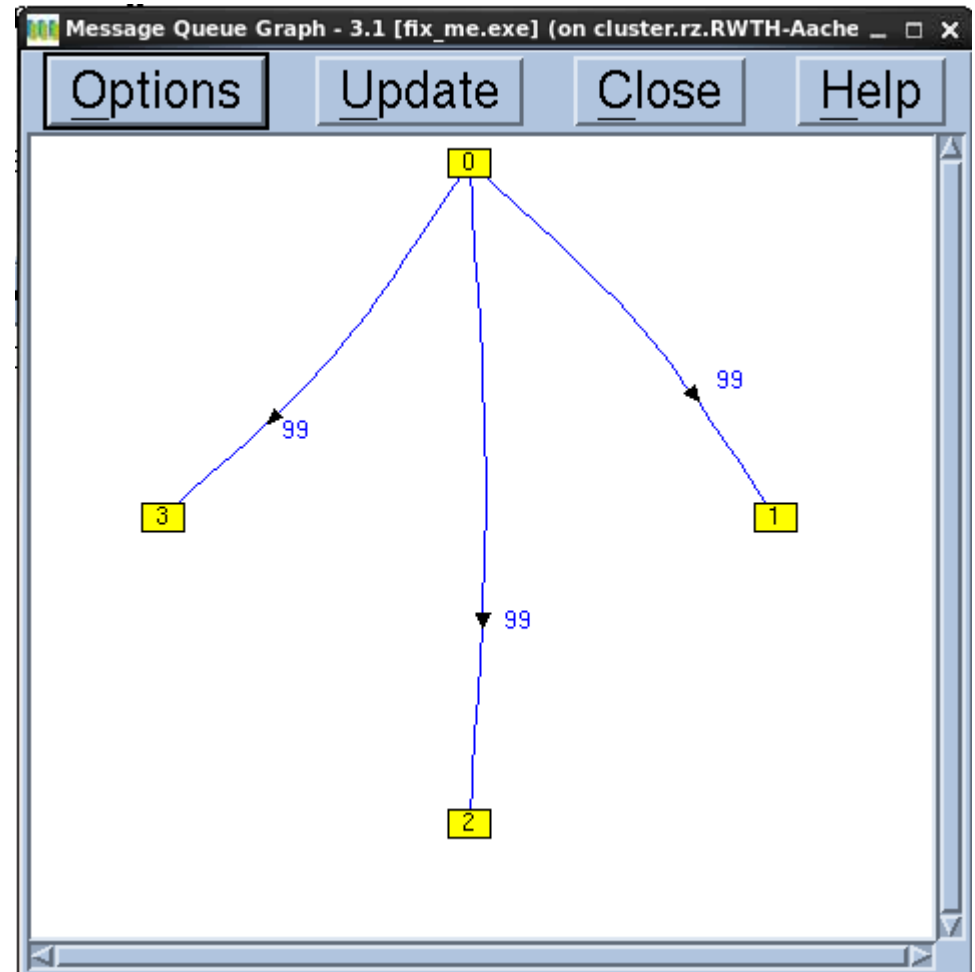
The screenshot shows the TotalView debugger interface for a process named 'fix\_me.exe.1'. The status bar indicates 'Rank 1: fix\_me.exe.1 (Stopped)'. The 'Stack Trace' pane shows the current call stack, with 'main' at the top. The 'Stack Frame' pane shows the function 'main' with arguments: 'argc: 0x00000001 (1)', 'argv: 0x7fff987f89f8 -> 0x7fff987fa157 -> "fix"', and local variables: 'nprocs: 0x00000004 (4)', 'myrank: 0x00000001 (1)'. The source code pane shows the following code:

```
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 error: rank %d, data[%d] = %d\n", myrank, j, data[j]);
48                 }
49             }
50         }
51     }
52 }
```

The line `MPI_Recv(&data, count, MPI_INT, 0, tag2, MPI_COMM_WORLD, &status);` is highlighted in yellow, indicating the current execution point. The 'Action Points' pane at the bottom is empty.

## ■ Message Queue Graph

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





## ■ 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"



“Tools” → “Message Queue Graph”

Hangs & Deadlocks

Pending Messages

Communication Patterns

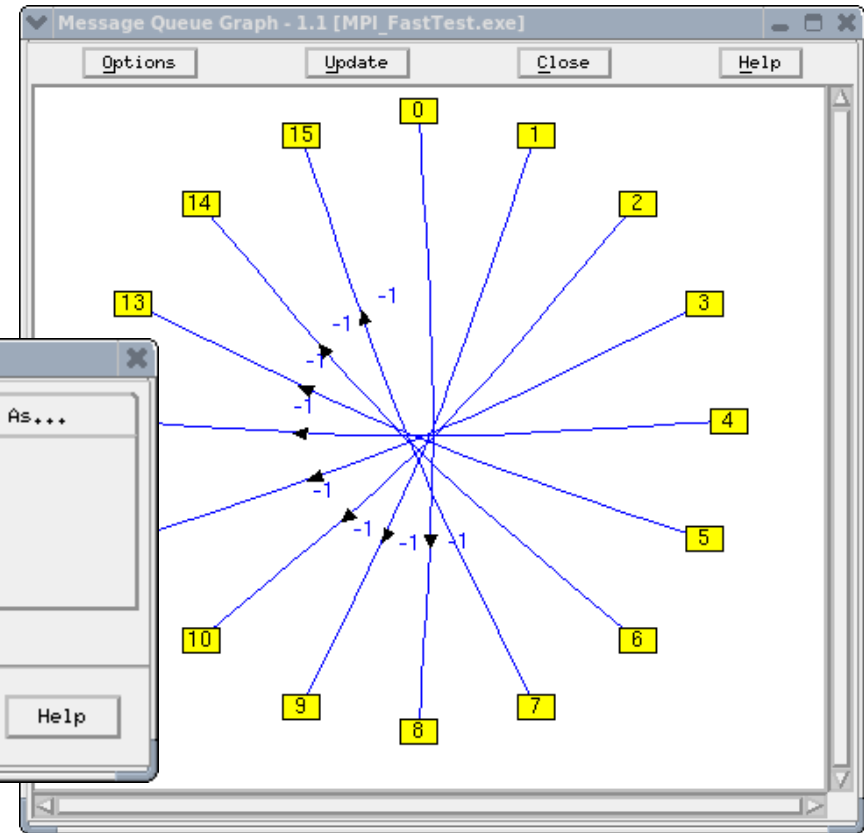
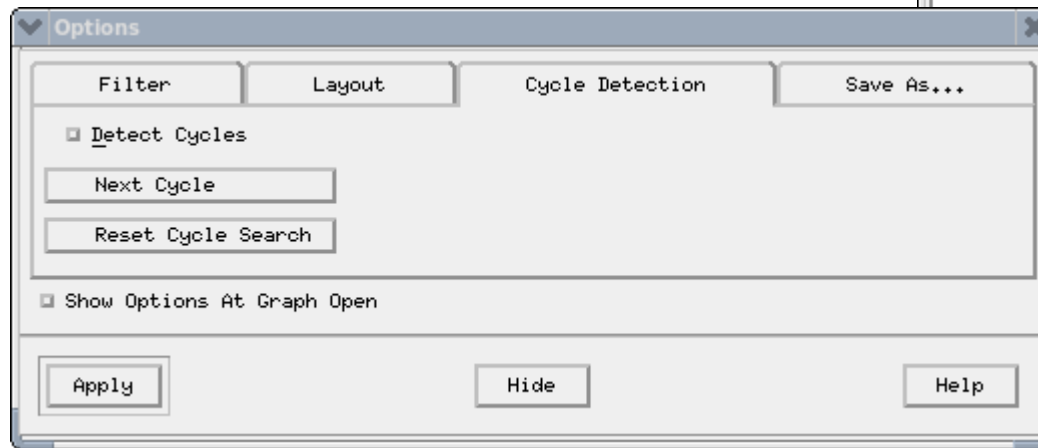
Find deadlocks:

“Options” → „Cycle Detection“

Green: Pending Sends

Blue: Pending Receives

Red: Unexpected Messages



Thank you for your attention!

