Debugging with TotalView

Thomas Reichstein

reichstein@rz.rwth-aachen.de

Center for Computing and Communication
RWTH Aachen University

22.03.2010

Agenda

- 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

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

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!

Starting TotalView

Initialise the environment:

```
$ module load totalview
```

Startup:

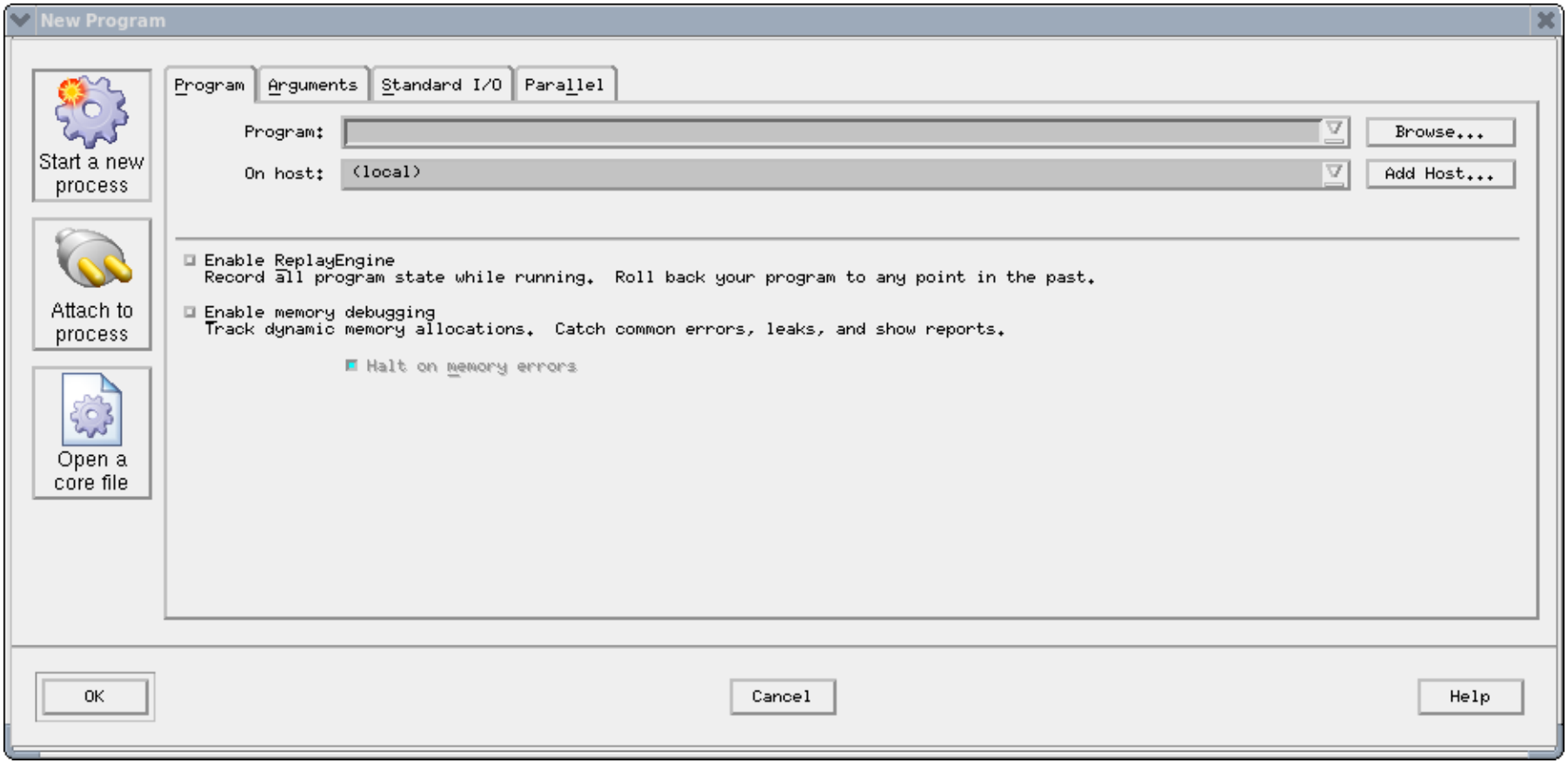
```
$ totalview
```

Main modes: start a new process
attach to a running process
load a core file

or load a binary called a.out directly:

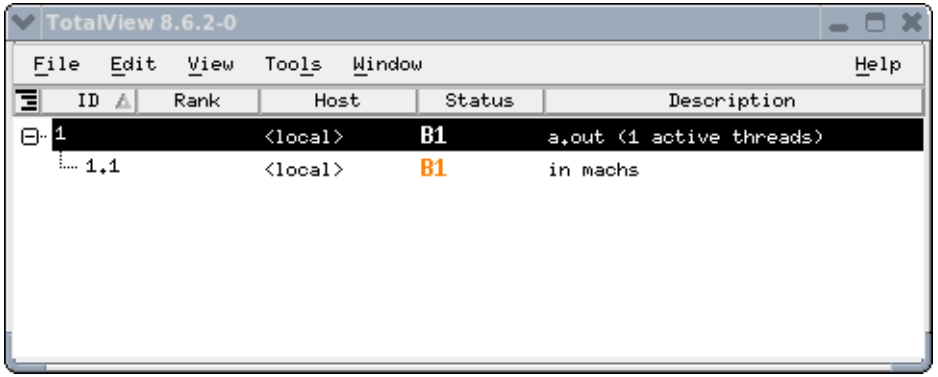
```
$ totalview a.out -a <options of a.out>
```

Options



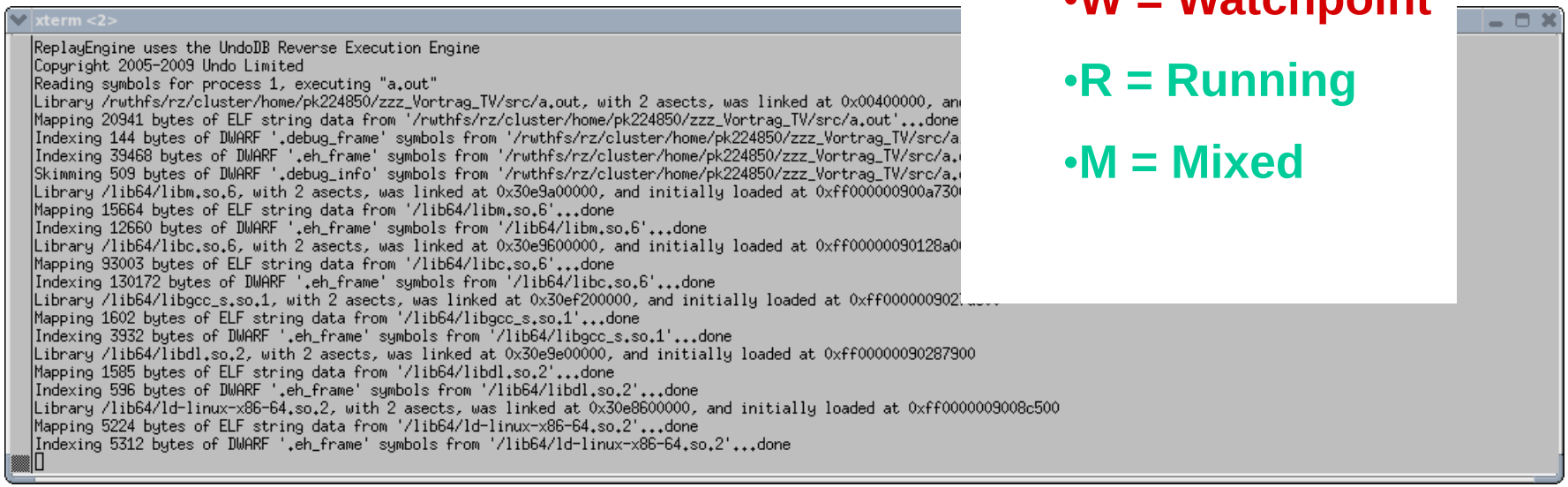
RWTH AACHEN UNIVERSITY Center for Computing and Communication

Root Window and console output

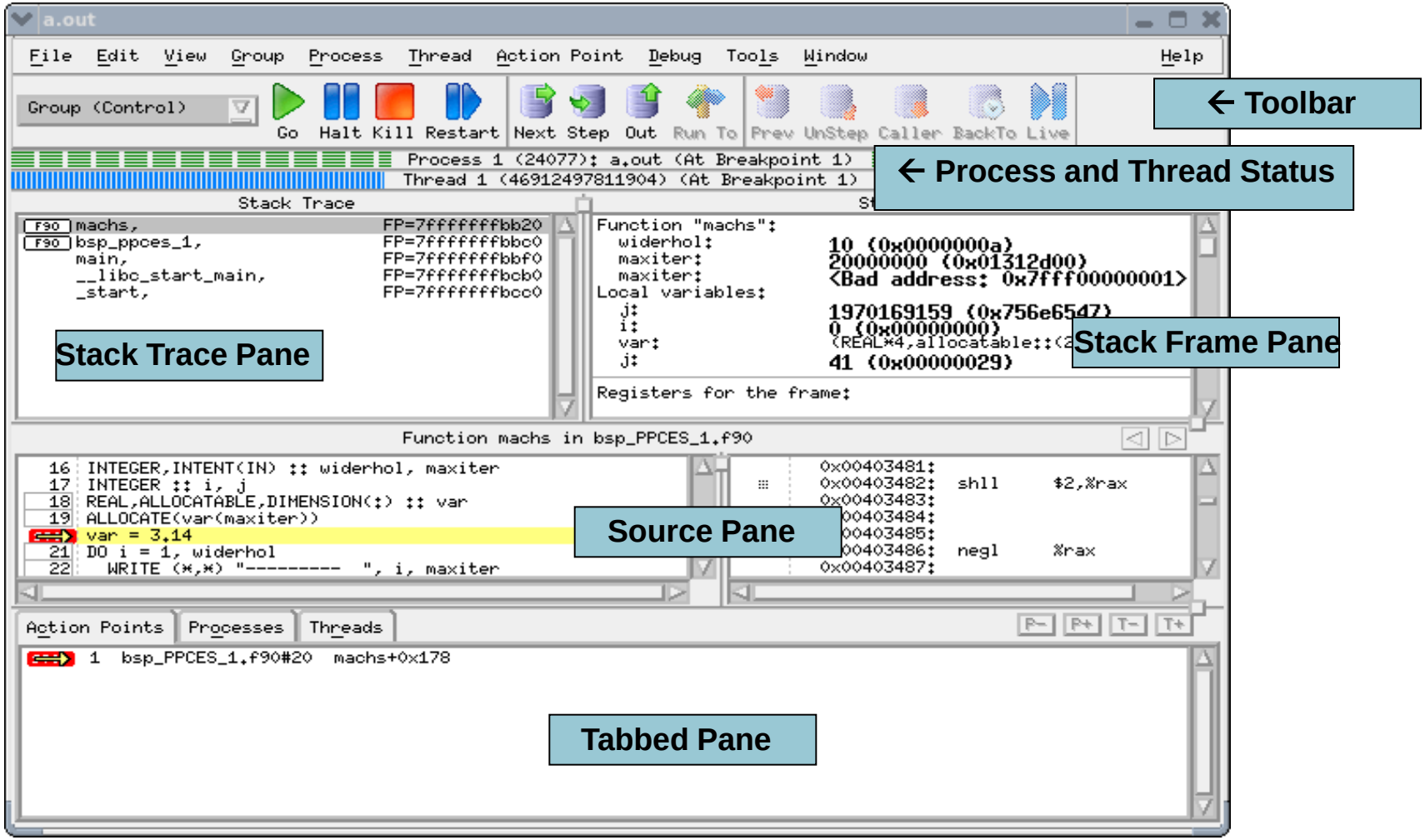


Status Info

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

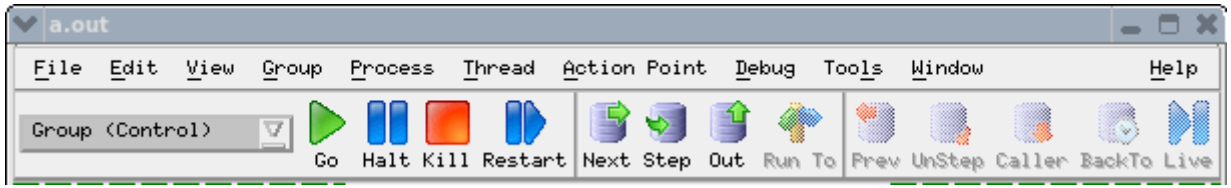


Process Window



RWTH AACHEN UNIVERSITY Center for Computing and Communication

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

Looking at Data

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

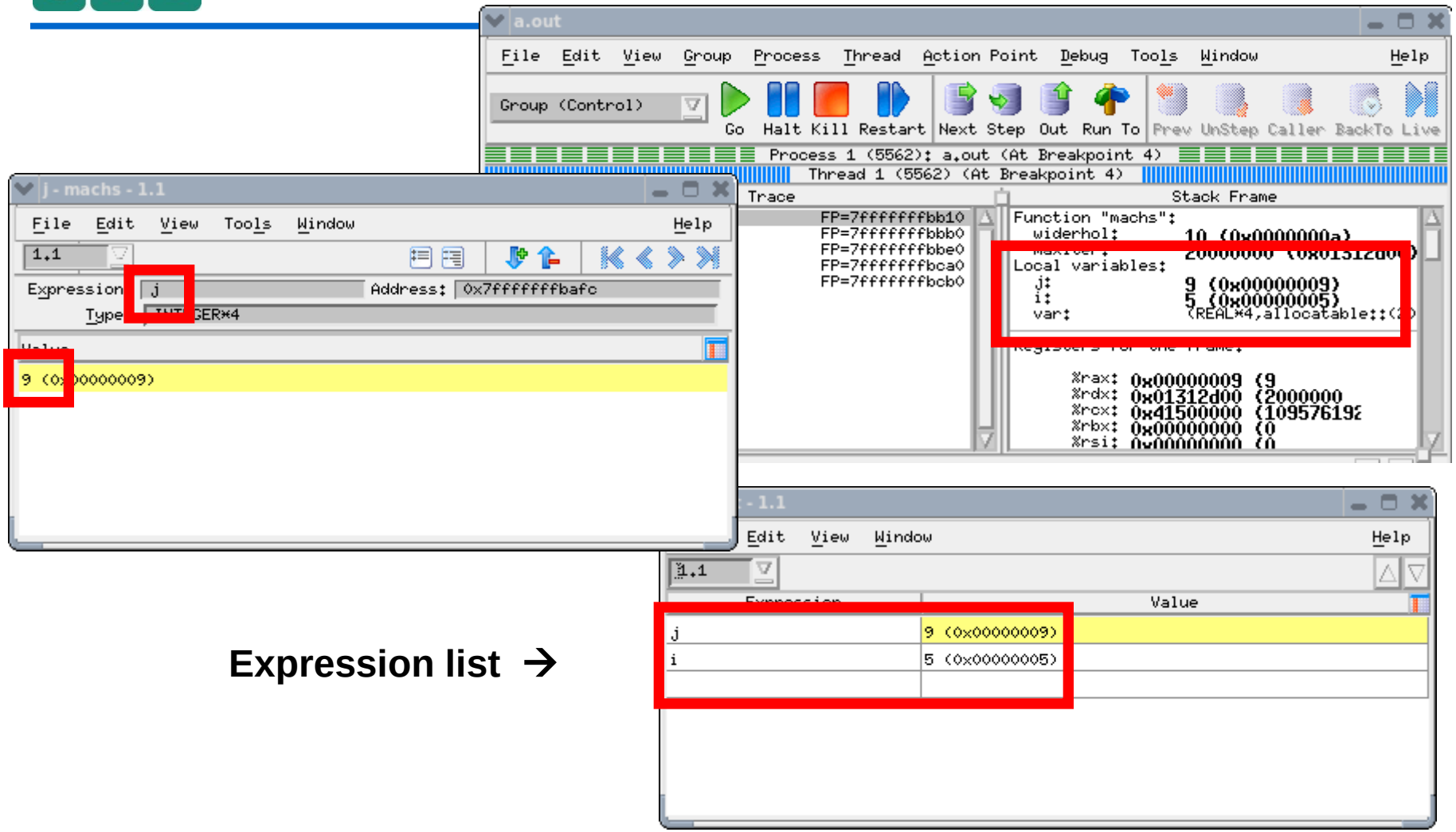
3.3 View → Lookup Variable

Monitor via expression list

4. right-click on a variable and then

“Add to Expression list”

Viewing Examples



Expression list →

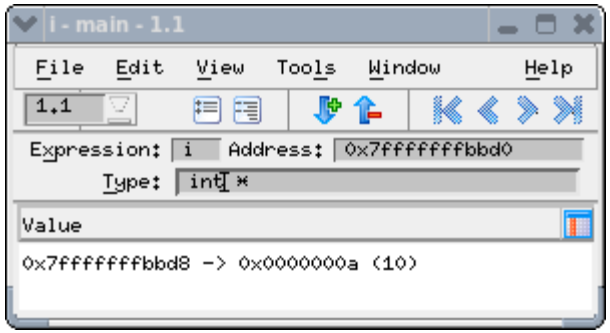
RWTH AACHEN UNIVERSITY Center for Computing and Communication

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

"Type Casting"

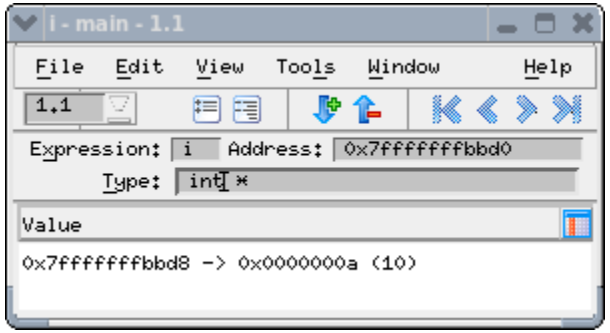
Edit the type of a variable

Type Casts Read from Right to Left

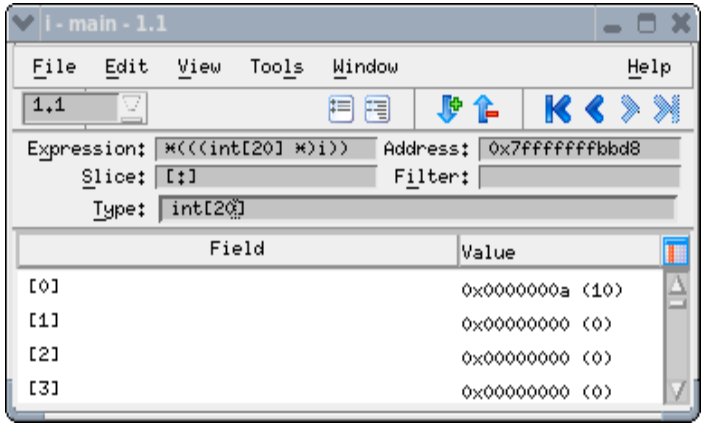
Examples:

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

int*[10] Array of 10 pointers to int



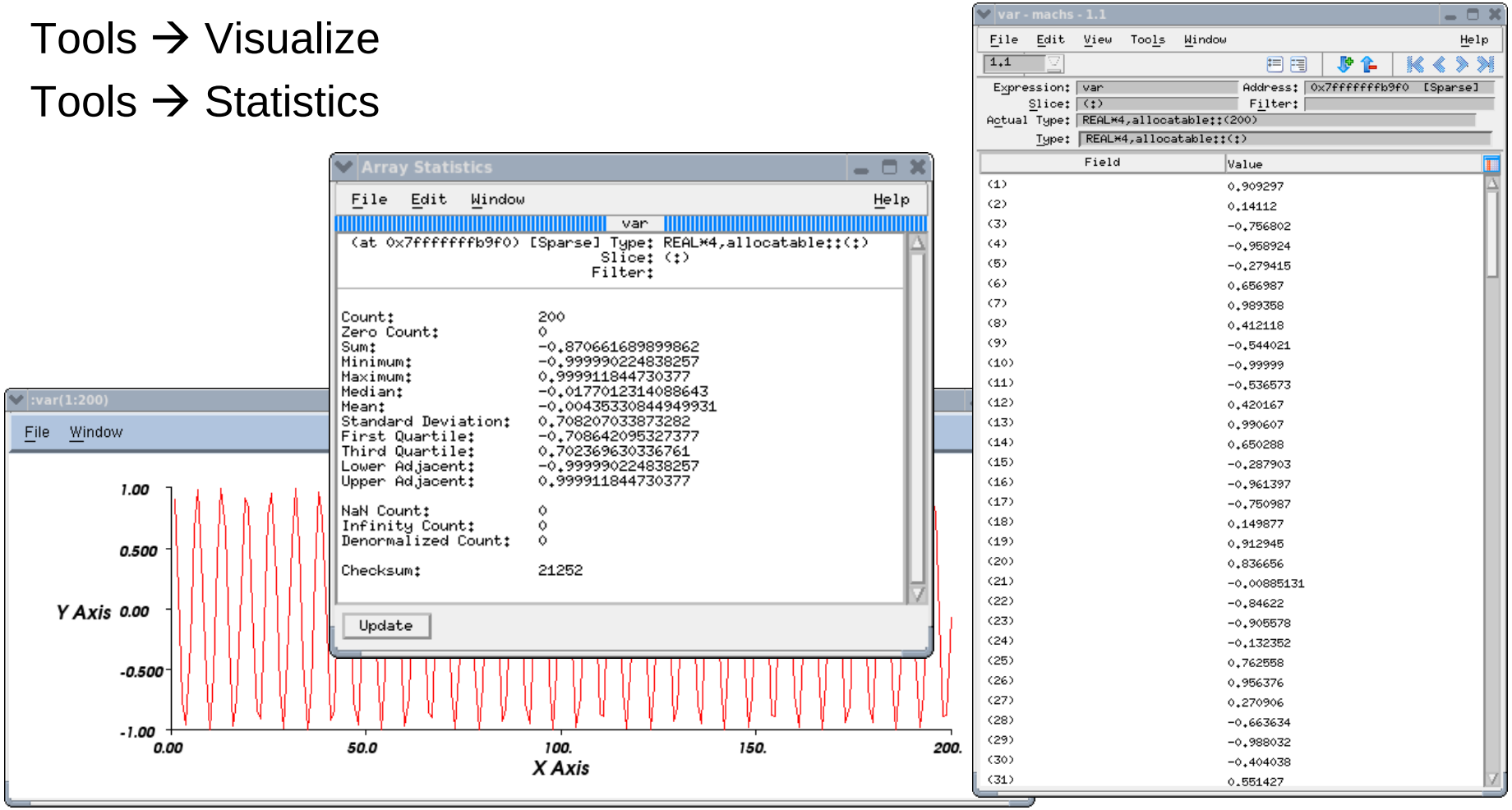
Type Casting



Array tools

Tools → Visualize

Tools → Statistics



RWTH AACHEN UNIVERSITY Center for Computing and Communication

Breakpoints

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

Agenda

- 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

OpenMP and TotalView

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!

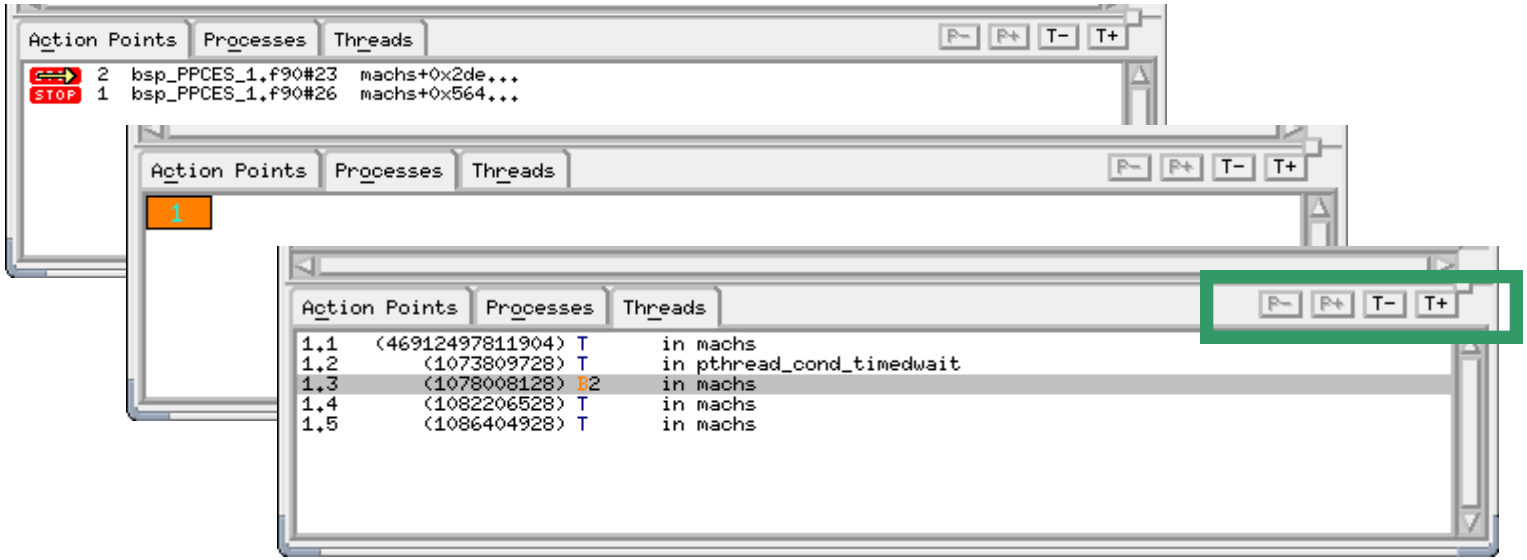
Nevertheless, TV is better than no TV

Start as usual

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

Note the Tabbed Pane on bottom of Main Window

Tabbed Pane



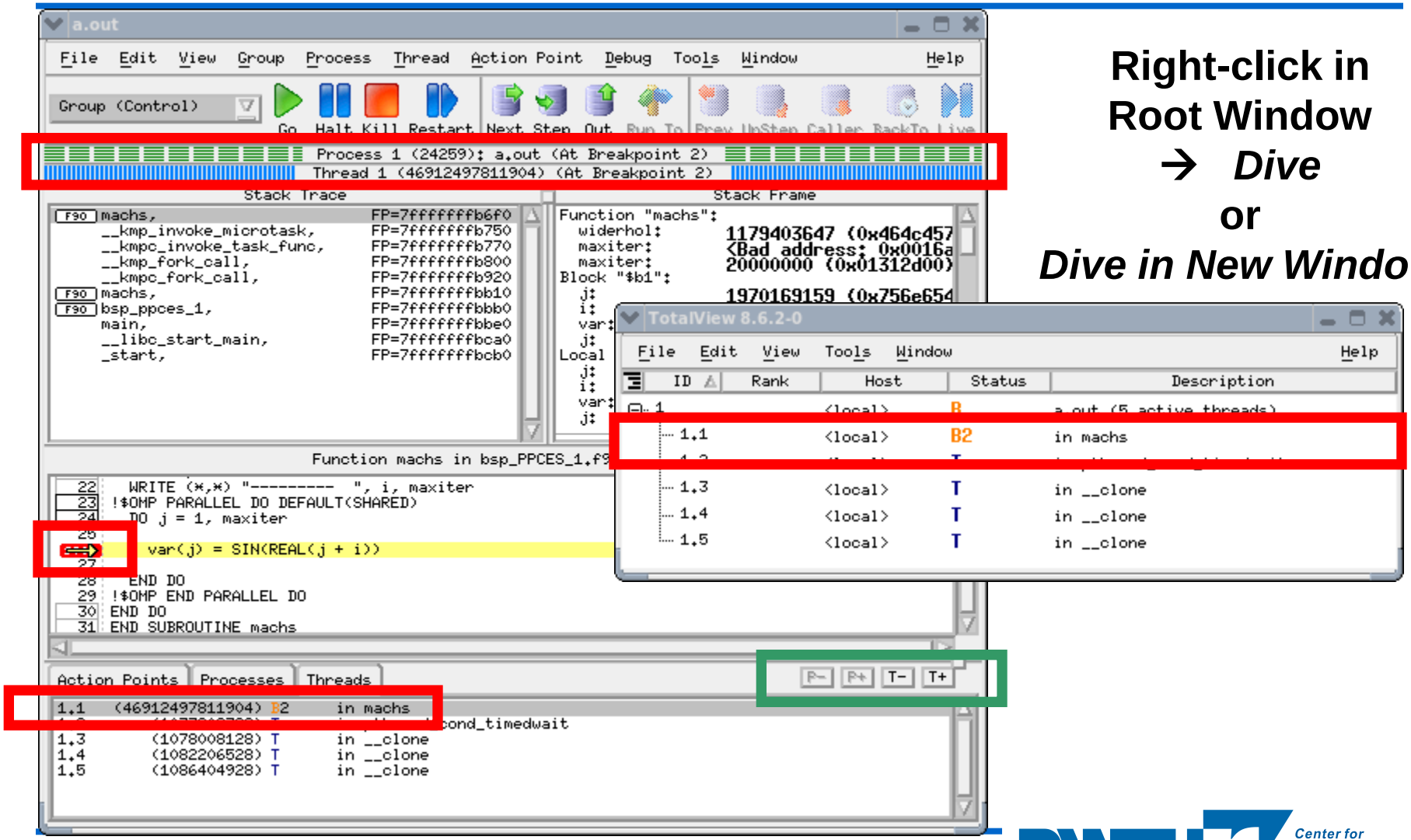
Action Points tab: e.g. Breakpoints

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

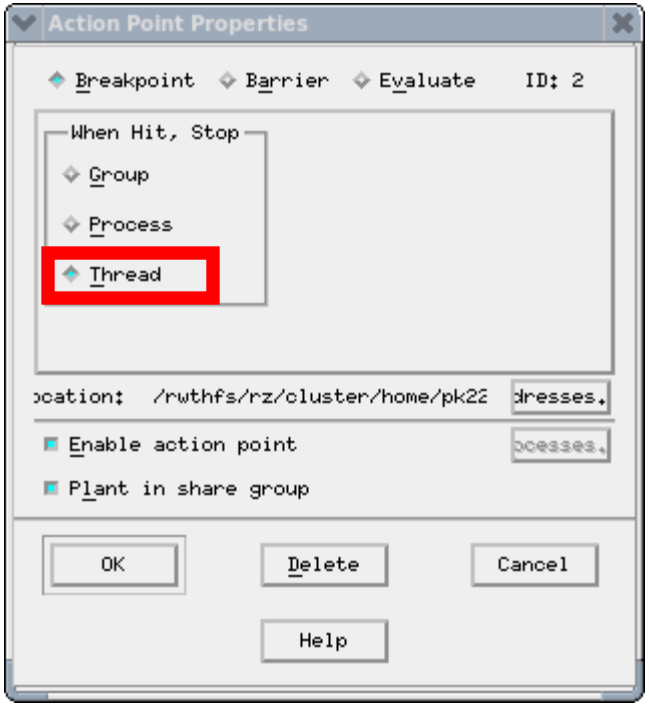
RWTH AACHEN UNIVERSITY Center for Computing and Communication

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)**



RWTH AACHEN UNIVERSITY Center for Computing and Communication

Looking at Variables across Threads

Let's see values of a variable in all threads

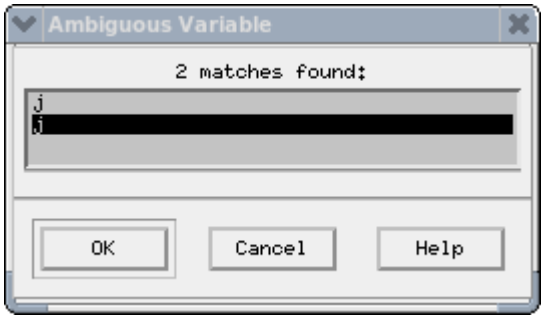
Right-click on a variable and *"Across Threads"*

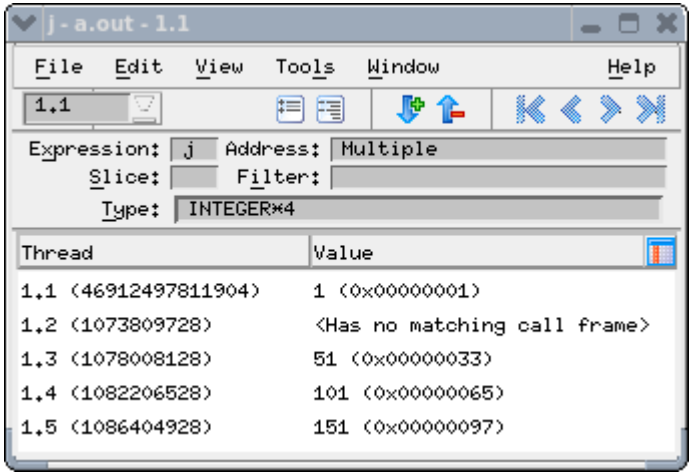
Shared vars are equal

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
```



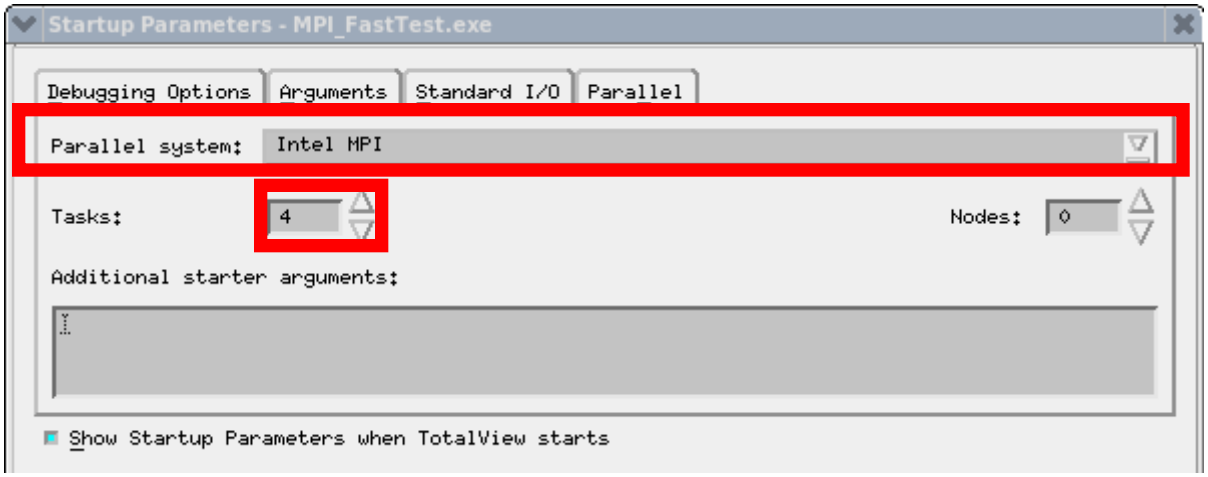


Agenda

- Why use a debugger?
- Before you start
- Starting TotalView
- Basic debugging
- Multithreaded (OpenMP) debugging
- MPI debugging
- Advanced
  - Memory debugging
  - Replay Engine

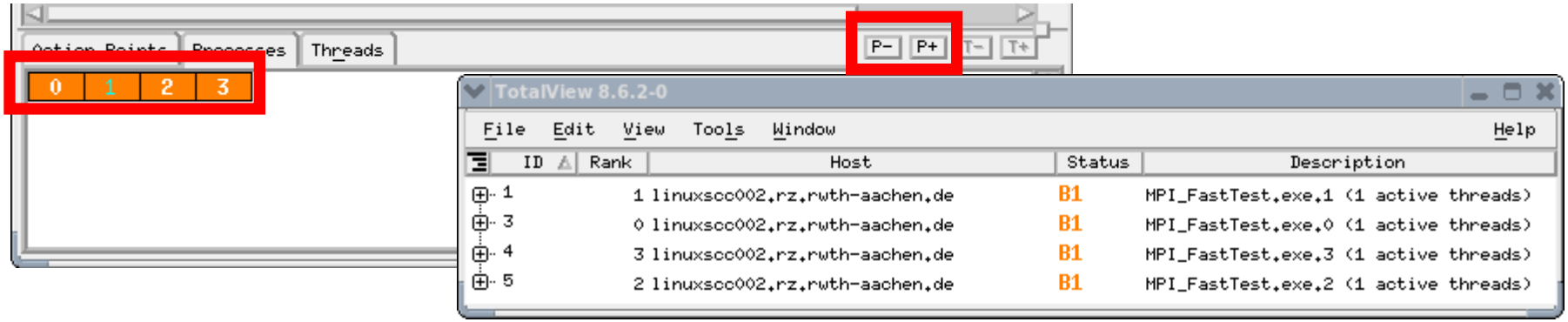
Starting MPI debugging

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



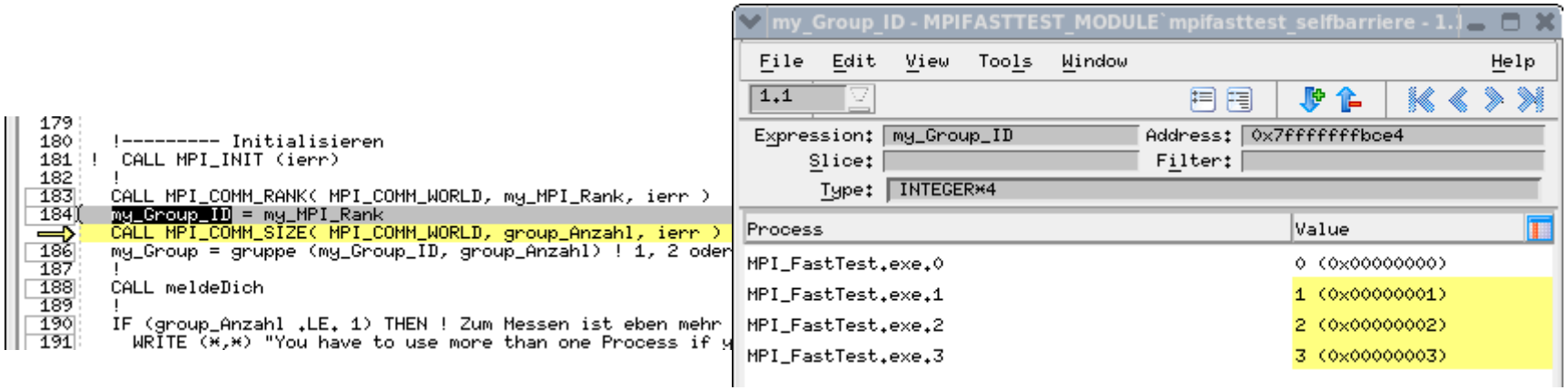
Classic Launch (the only way on BlueGene)
- e.g. Intel and Oracle MPI: *mpiexec -tv -np 16 MPI_FastTest.exe*
- Arguments depend on MPI vendor
- Attach/Attach to subset/Detach/Reattach possible
- Warning: don’t work if combinig Oracle MPI and Intel compiler

Navigating and Looking at Variables



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

Right-click on a variable and *"Across Processes"*

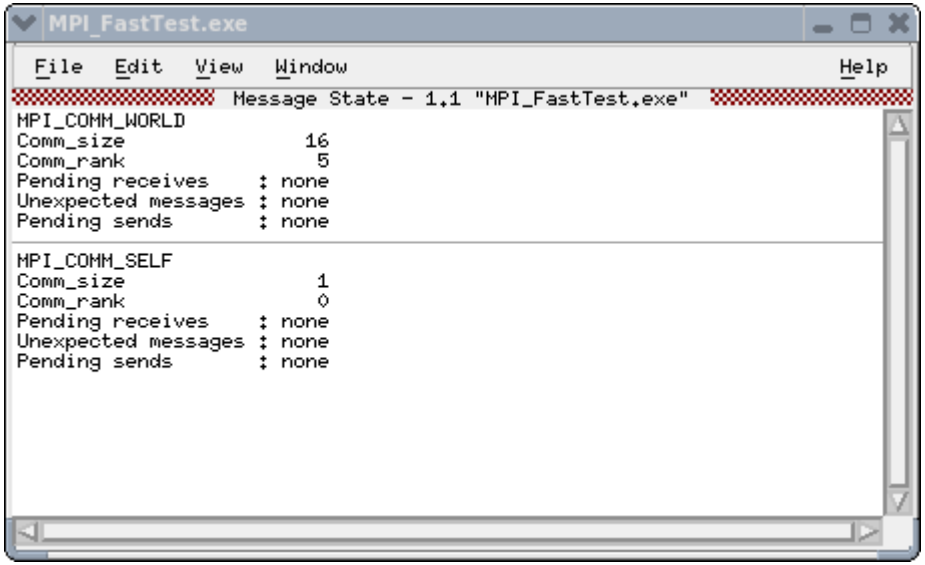


View MPI Message Queues

"Tools" → "Message Queue"

Provides information from the MPI layer

- Unexpected messages
- Pending Sends
- Pending Receives



Message Queue Graph

"Tools" → "Message Queue Graph"

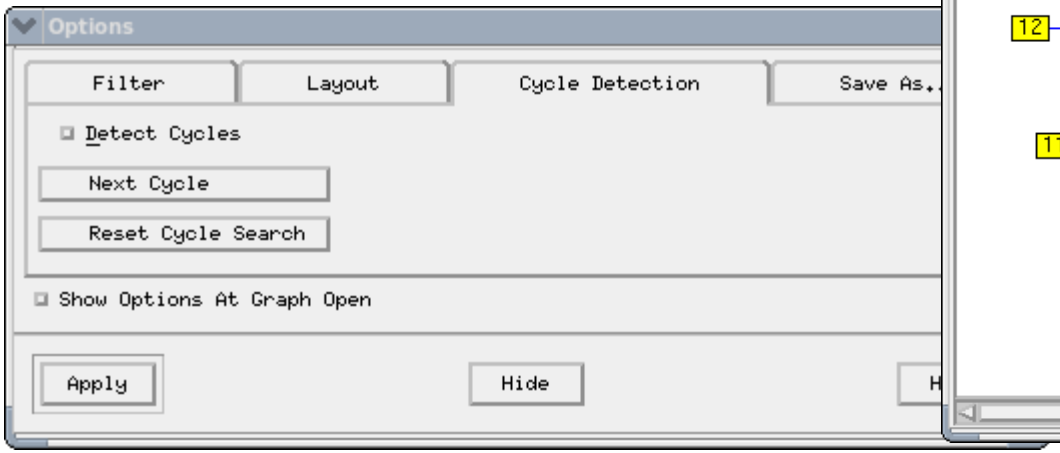
Hangs & Deadlocks

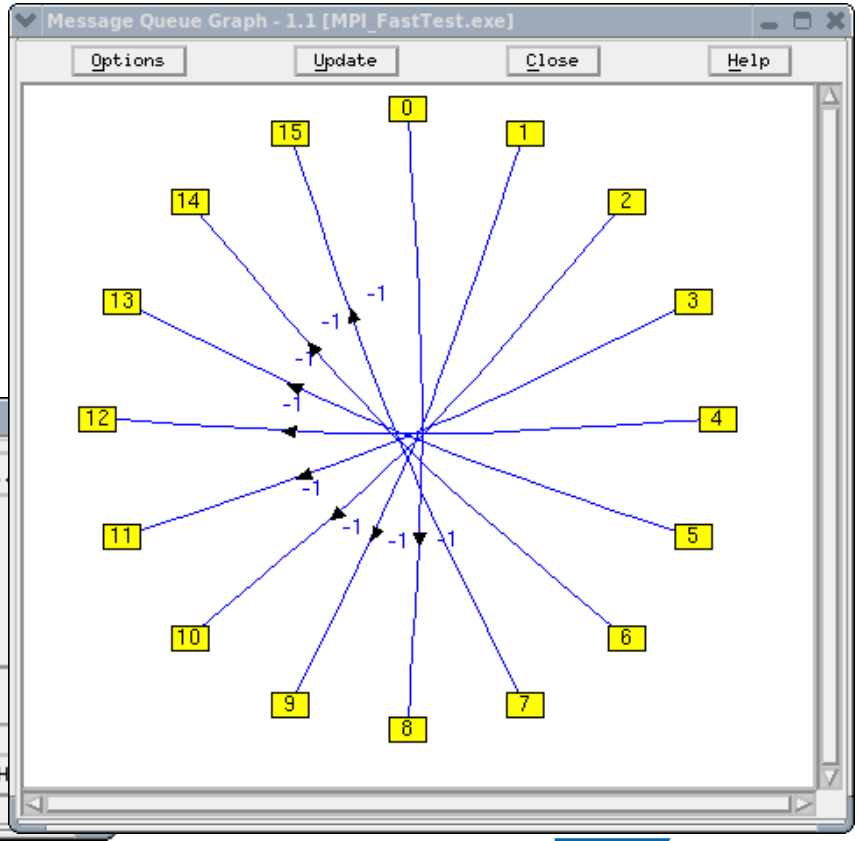
Pending Messages

Communication Patterns

Find deadlocks:

"Options" → „Cycle Detection"





RWTH AACHEN UNIVERSITY Center for Computing and Communication

Agenda

- 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 an 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

RWTH AACHEN UNIVERSITY Center for Computing and Communication

Summary

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 :-)

Online Information

TotalView Users Manual PDF Format
www.totalviewtech.com/pdf/other/TotalView_User_Guide.pdf

TotalView Reference Guide PDF Format
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?