Debugging Fortran code with Totalview

Reinhold Bader
Leibniz Supercomputing Centre

February 2009

Overview and Licensing

- **Totalview**
  - **software development tools for analyzing, debugging, and tuning the performance of programs**
  - **can be serial, parallel (MPI) or multithreaded (OpenMP)**
    - *Coarrays are not supported though*
  - **presently available: release 8.6.x**
- **Toolset:**
  - **graphical debugger (scriptable shell interface also av.)**
  - **memory debugging (add-on product Memoryscape)**
  - **replay engine (no need to restart program at error)**
- **Documentation:**
  - **at LRZ:**

http://www.lrz.de/services/software/programmierung/totalview

  - **Vendor web site:**

http://www.totalviewtech.com/

©2009 LRZ

Preparations: compiling the code

Compiler	switches for compilation	switches for linkage	Remarks
ifort / mpif90	`-g -O2 -check all -traceback`	`-traceback`	-check all activates extensive runtime error checking; for array bound checking only replaceA3 by -check bounds
icc / icpc / mpicc / mpiCC	`-g -O2 [-traceback]`		-traceback only useful for C programs linked into Fortran executables

- **Debug symbols:**
  - **-g for nearly all compilers**
  - **without, you will not get source window, only assembler**
- **optimization:**
  - **usually a good idea to add optimization**
  - **code may run very slowly otherwise**
  - **disadvantage: exact source locations may not be identified**
- **external libraries:**
  - **may not have debug symbols**

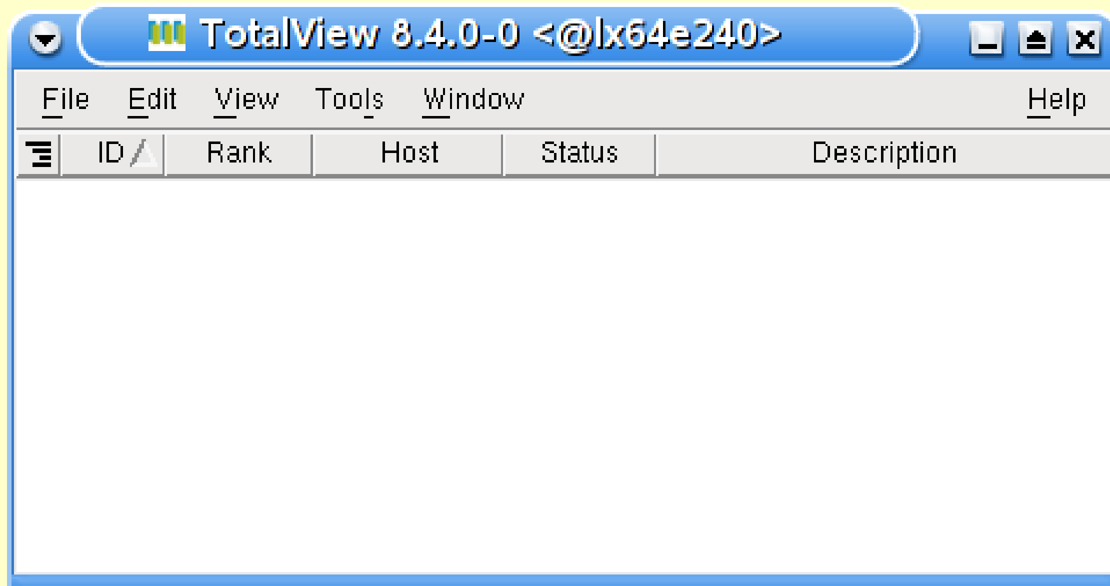
©2009 LRZ

Starting the debugger

- **Will use the GUI**
  - **check that X11 protocol works**
    - *DISPLAY variable and ~/.Xauthority entry*
  - **run the command**

```
totalview &
```

which will cause the following windows to appear:



main debugger window

- **File menu also offers „New program" entry**

... and the start-up window

New Program <@lx64a133>

Start a new process

Attach to process

Open a core file

Program | Arguments | Standard I/O | Parallel

Program: [ ] Browse...

On host: (local) Add Host...

Enable ReplayEngine
Record all program state while running. Roll back your program to any point in the past.

Enable memory debugging
Track dynamic memory allocations. Catch common errors, leaks, and show reports.

Halt on memory errors

OK Cancel Help

note the multiple tabs

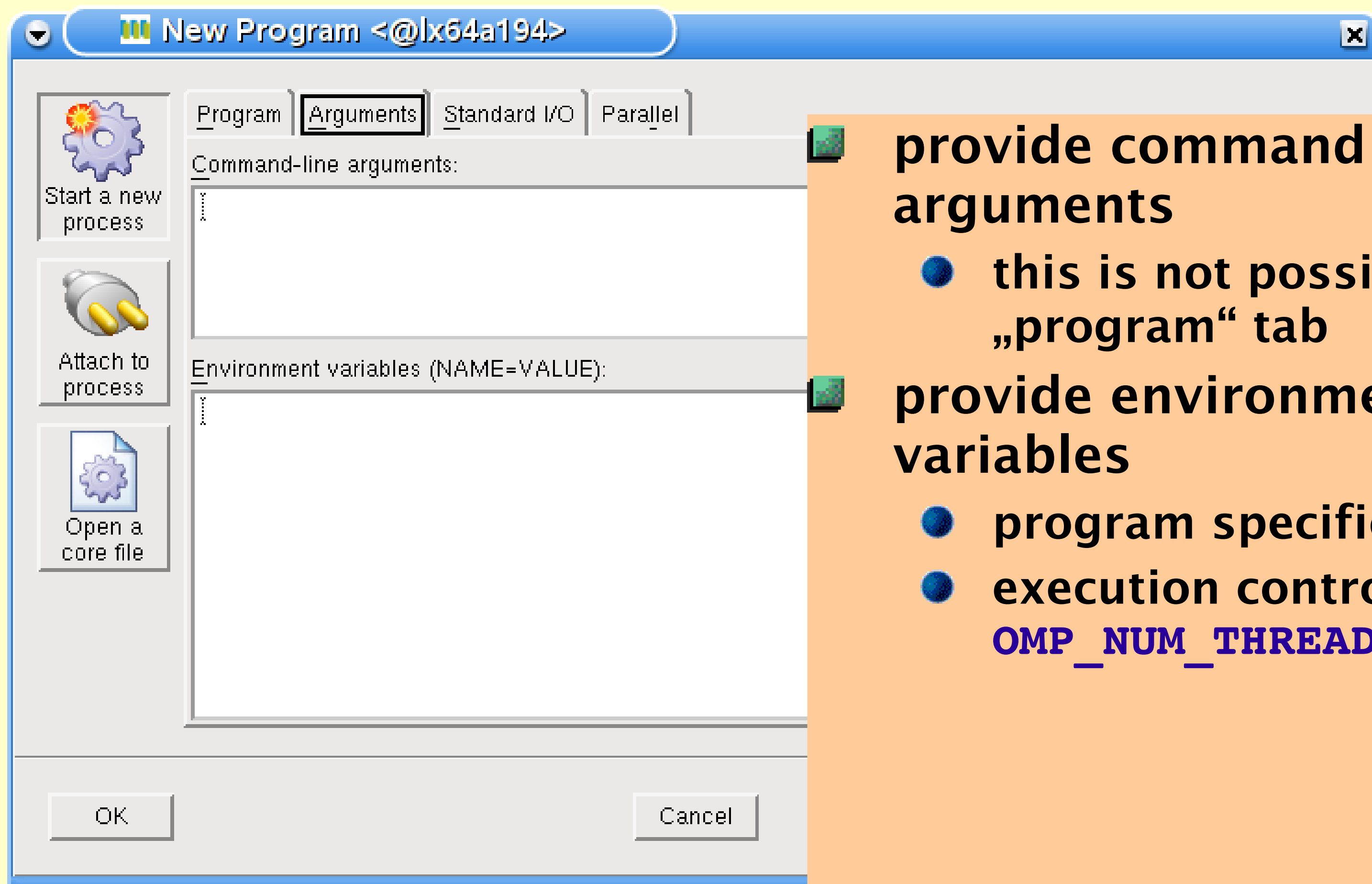
©2009 LRZ

Discussion of start-up window

- Selection of item (left) will change context
  - only discussing „start new process“ from now on
- „Program“ item:
  - type in program name, e. g. `./myprog.exe`
  - ... or use file browser to select executable
- „Host“ item:
  - can start on remote host
  - keep things simple and start on local host
- additional features:
  - need to activate explicitly
    - *memory debugging is always a good idea*
  - may not all be available on every platform

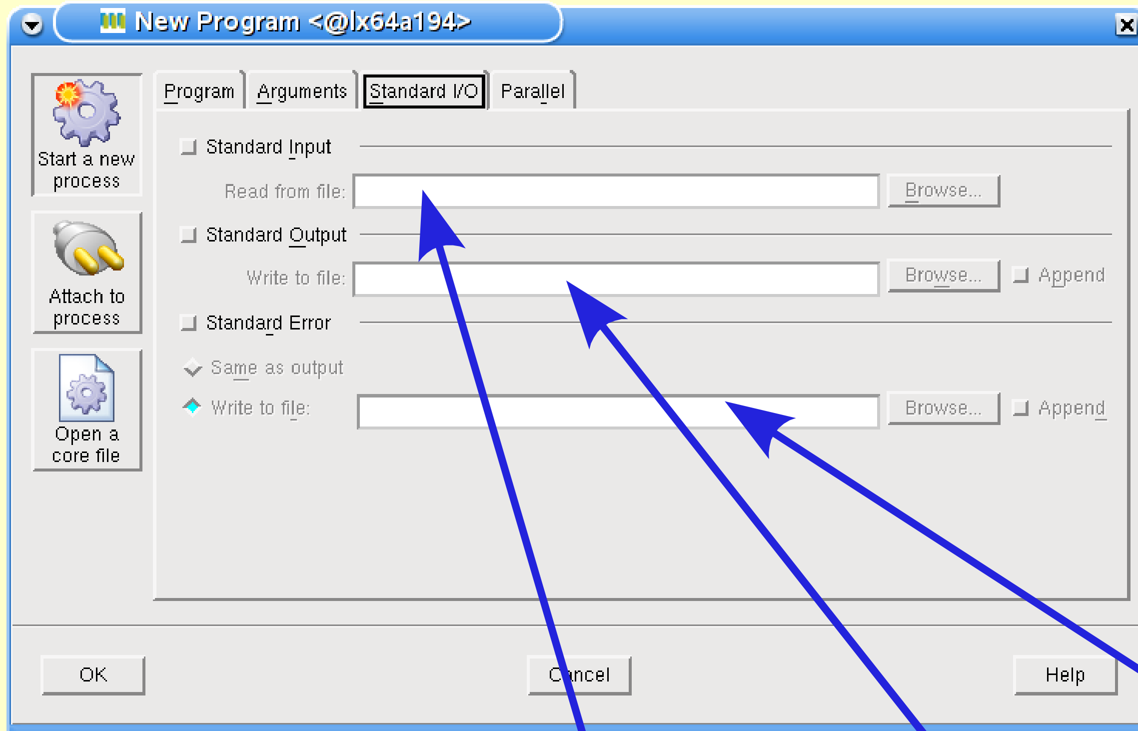
©2009 LRZ

The „arguments“ tab



- provide command line arguments
  - this is not possible via the „program“ tab
- provide environment variables
  - program specific
  - execution control e.g., `OMP_NUM_THREADS`

The „standard I/O“ tab



translate I/O items from command line

- need to mark required channels
- and specify files

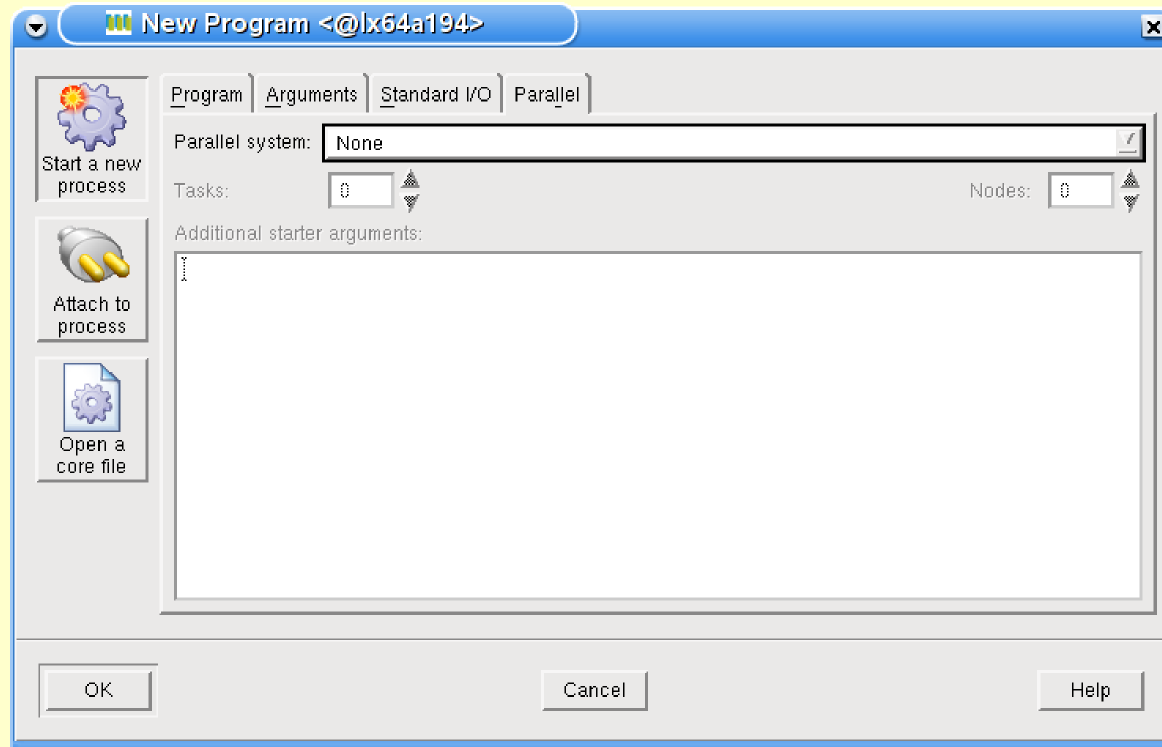
else:

- console is used

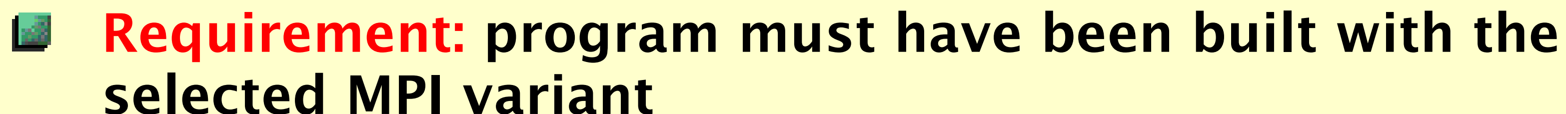
```
./myprog.exe < infile.dat > outfile.dat 2> errfile.dat
```

©2009 LRZ

The „parallel“ tab



- relevant for MPI programs
  - by default, „None“ is selected
- sensible choices on LRZ HPC systems:
  - MPT (= SGI MPI) on Altix
  - Intel MPI
- set number of tasks
  - will not start if 0



- **Requirement:** program must have been built with the selected MPI variant
  - load appropriate environment module on LRZ system

Example 1: simple serial program with bug

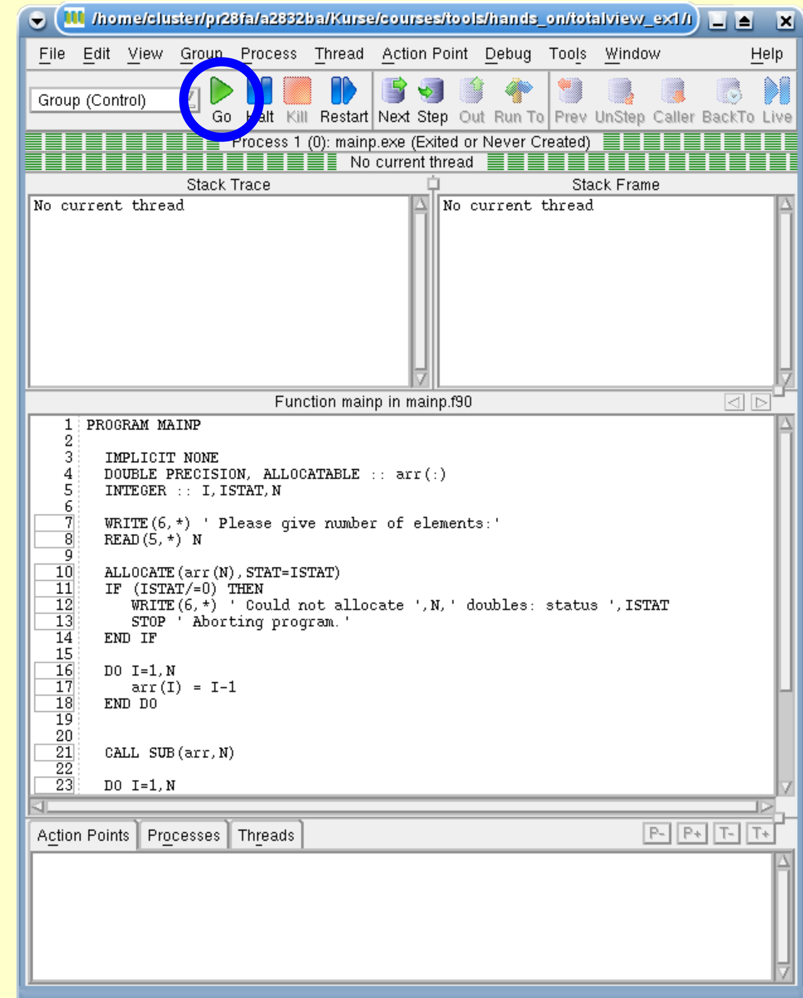
- **Step-by-step:**
  - build program by typing `make` (may need to suitably modify `Makefile`)
  - run program from console: `./mainp.exe`

```
Please give number of elements:
56
forrtl: severe (174): SIGSEGV, segmentation fault occurred
Image              PC                Routine            Line        Source
mainp.exe          0000000000402F30  sub_                  9        sub.f90
mainp.exe          0000000000402D98  MAIN__               21        mainp.f90
mainp.exe          0000000000402A22  Unknown         Unknown        Unknown
libc.so.6          00002AC2D96F5154  Unknown         Unknown        Unknown
mainp.exe          0000000000402969  Unknown         Unknown        Unknown
```

  - above output (red) from Intel compiler (traceback included) is in fact sufficient to diagnose and fix error
  - but still will go at it with the debugger

Starting the program

- **Procedure as described below:**
  - **only program name is needed**
  - **can specify prepared file `input.dat` as stdin**
  - **finally, press OK button in start-up window**
- **Will see source code of main program**
  - **unless compiled without debug symbols**
- **Press Go button**
  - **runs the program ...**

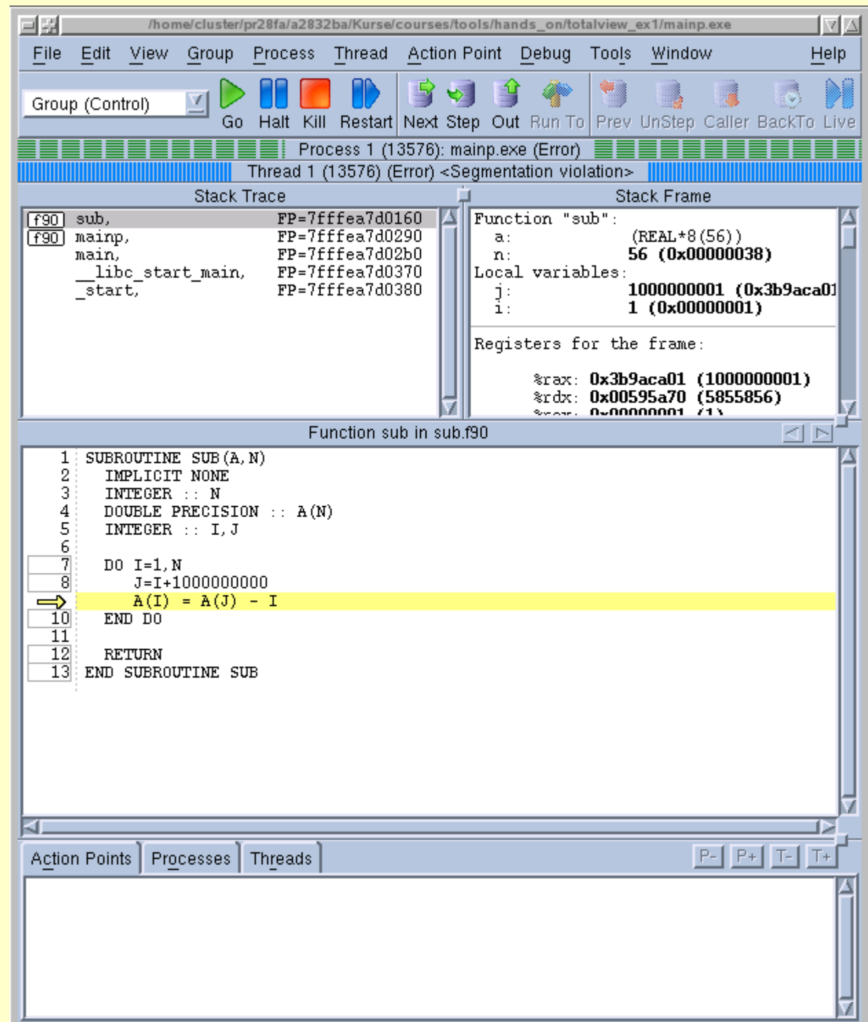


©2009 LRZ

Hitting the program error

Subwindows:

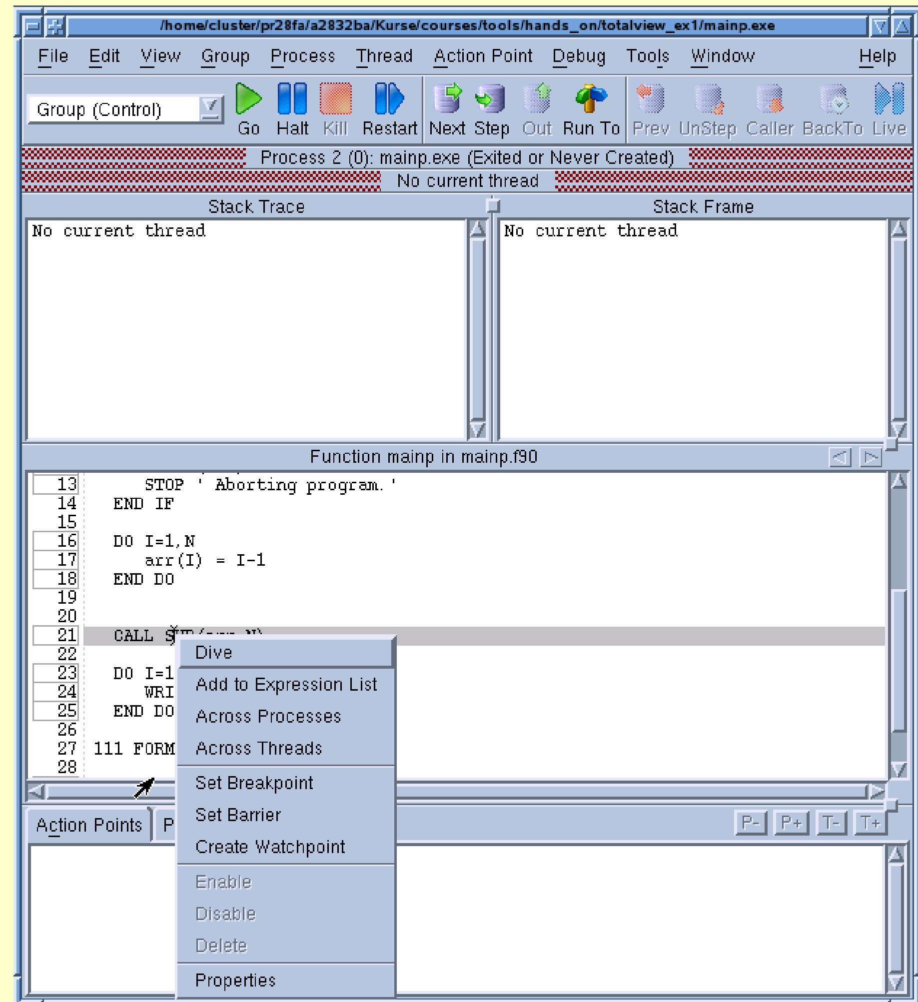
1. **stack trace**
   - call sequence
   - move to source file
2. **stack frame**
   - local variables and their values
3. **source window**
   - indicates presently executed statement
   - last statement executed if program crashed
4. **info tabs**
   - further information about processes etc.



©2009 LRZ

Controlling program execution: Breakpoints and Stepping (1)

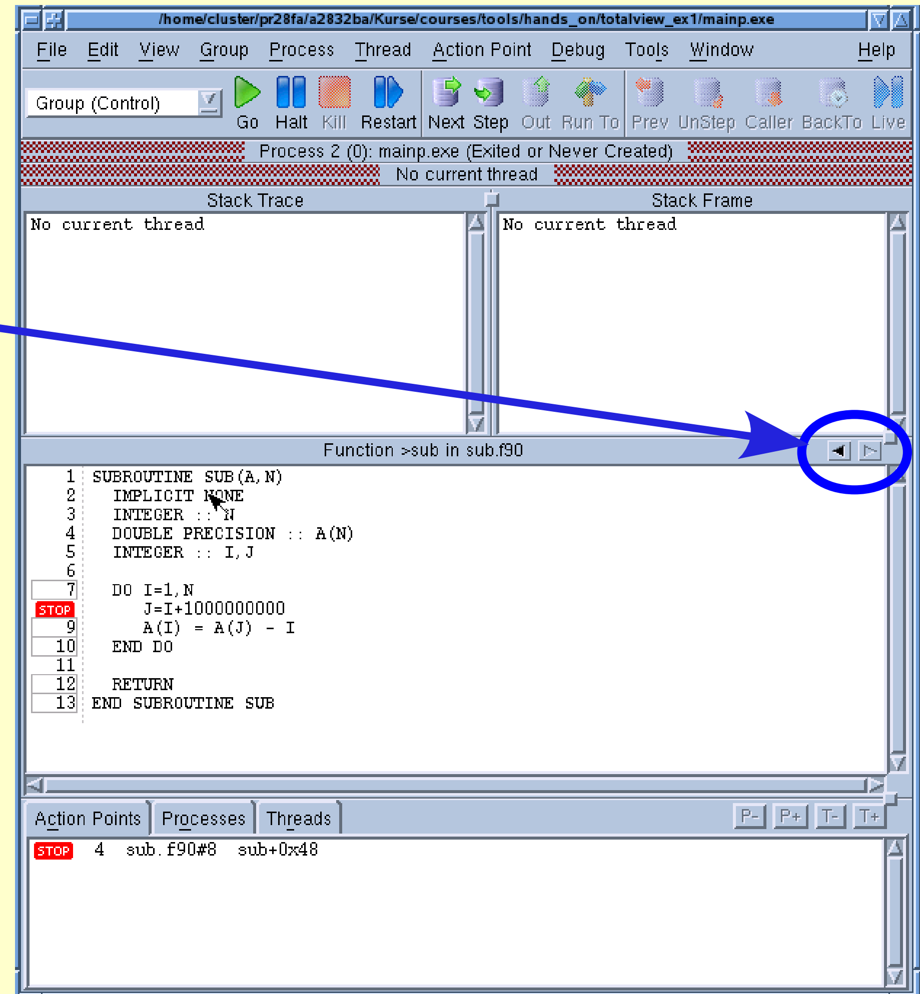
- **Non-local errors:**
  - **segmentation error may be unrelated to actual error in code**
- **Have a suspicion which code section has a problem**
  - **dive into subroutine via context menu (right click on `SUB()`)**



©2009 LRZ

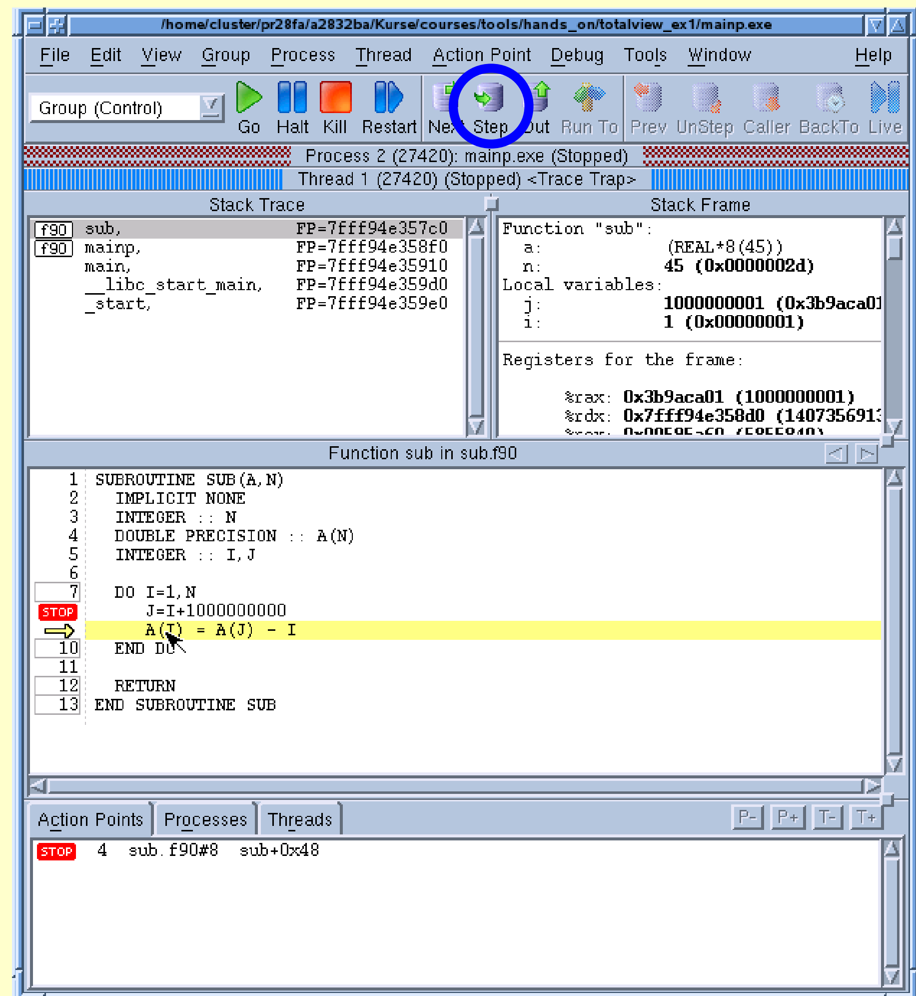
Controlling program execution: Breakpoints and Stepping (2)

- Now see source code of `SUB()`
  - even if program not started yet
  - navigation arrows
- Click on line number just before suspicious part of code
  - sets a breakpoint
  - stop sign in source line and in „Action Points“ Tab
- „Go“ →
  - will stop at every breakpoint



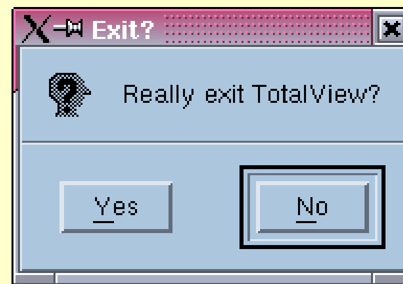
Controlling program execution: Breakpoints and Stepping (3)

- **Yellow Arrow indicates execution point**
  - **statement to be executed**
- **Step button**
  - **to next statement**
- **Next button**
  - **to next statement in presently executed function**
- **Out button**
  - **to next statement in calling function**



Quitting Totalview

- Choose File → Exit in any window



then confirm your choice

©2009 LRZ

Example 2: Global variables, watching

- Have a code with module variables

```
module mod_stuff
  implicit none
  integer :: my_global
  real, allocatable :: my_array(:)
contains
  subroutine modify(i, x)
    integer, intent(in) :: i
    real, intent(in) :: x
    my_array(i) = x
  end subroutine
end module
```

- Dive into variable from program **USE**ing the module
  - start up with replay engine and memory debugging enabled

myprog.exe

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

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

Process 1 (10566): myprog.exe (At Breakpoint 1) [M]

Thread 1 (10566) (At Breakpoint 1)

Stack Trace

```
f90 myprog,              FP=7fff329b08f0
    main,                FP=7fff329b0910
    __libc_start_main,   FP=7fff329b09d0
    _start,              FP=7fff329b09e0
```

Stack Frame

```
Function "myprog":
  No arguments.
Local variables:
  i:              849021432 (0x329b09f8)
  my_array:       (REAL*4,allocatable::
Registers for the frame:
  %rax: 0x00000000 (0)
  %rdx: 0x00000000 (0)
  %rcx: 0x00000000 (0)
```

Function myprog in myprog.f90

```
 1  program myprog
 2    use mod_stuff
 3    implicit none
 4    integer :: i
      allocate(my_array(10))
 6    call modify(5, 2.5)
 7    i = 15
 8    call modify(i, 4.5)
 9    deallocate(my_array)
10    call modify(5, 4.5)
11  end program
```

Action Points | Processes

1 myprog.f90#5

my_array - myprog.exe - 1.1

File Edit View Tools Window Help

1.1

Expression: my_array Address: 0x0057b5e0 [Sparse]

Slice: (:) Filter:

Actual Type: REAL*4,allocatable::(0:-1)

Type: REAL*4,allocatable::(:)

Error

(Unallocated)

©2009 LRZ

Global variables (2)

- Need to start up program
  - otherwise name unknown
  - use a breakpoint
  - dive into array via right click
  - after allocation: watch window automatically changes
- Replay Engine:
  - undoing buttons are now active
  - undo stepping

©2009 LRZ

Global variables (3)

- **Same applies to changing values**
  - **call of module procedure `modify()` does this here**

myprog.exe

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

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

Process 1 (10566): myprog.exe (Stopped) [M]

Thread 1 (10566) (Stopped) <Trace Trap>

Stack Trace

```
f90  myprog,              FP=7fff329b08f0
     main,                FP=7fff329b0910
     __libc_start_main,   FP=7fff329b09d0
     _start,              FP=7fff329b09e0
```

Stack Frame

```
Function "myprog":
  No arguments.
Local variables:
  i:          849021432 (0x329b09f8)
  my_array:   (REAL*4,allocatable::
Registers for the frame:
     %rax: 0x0045f690 (4585104)
     %rdx: 0x005850a4 (5787812)
     %rcx: 0xfffffffffffffffc (-4)
```

Function myprog in myprog.f90

```
 1  program myprog
 2    use mod_stuff
 3    implicit none
 4    integer :: i
STOP  allocate(my_array(10))
 6    call modify(5, 2.5)
 ⇨    i = 5
 8    call modify(i, 4.5)
 9    deallocate(my_array)
10    call modify(5, 4.5)
11  end program
```

Action Points | Processes

STOP 1 myprog.f90#5

my_array - myprog.exe - 1.1

File Edit View Tools Window Help

Expression: my_array Address: 0x0057b5e0 [Sparse]

Slice: (:) Filter:

Actual Type: REAL*4,allocatable::(10)

Type: REAL*4,allocatable::(:)

Field	Value
(1)	2.45891e+34
(2)	1.55796e-41 <denormalized>
(3)	2.45891e+34
(4)	1.55796e-41 <denormalized>
(5)	2.5
(6)	9.68233e-39 <denormalized>

©2009 LRZ

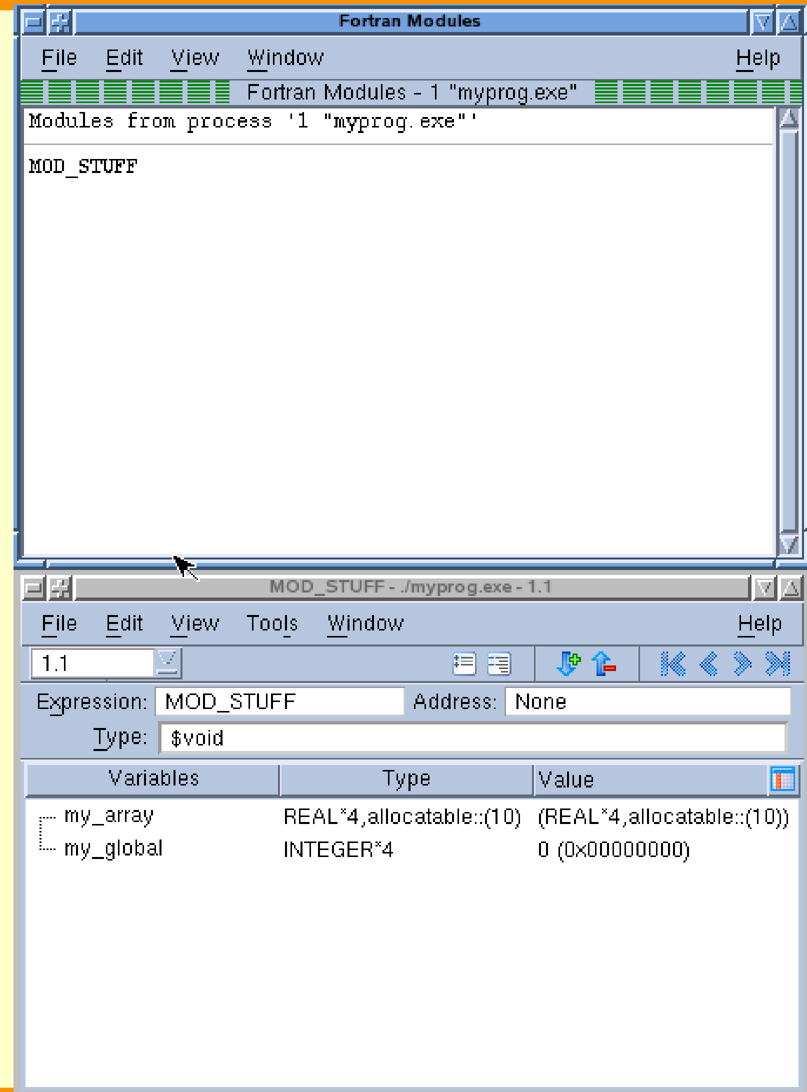
Changing variable values

- Second call to **`modify()`** was faulty
  - **`i=15`** out of bounds – TV actually did not notice ...
  - press **Prev** button to return to statement before second call
  - dive into variable **`i`**
  - change value to whatever you think is the correct one in the variable window
    - *can right click ⟶ choose „Change Value“*
    - *press return after changing value*
    - *TV 8.6.0 is* ***buggy*** *– need to deactivate Replay engine and re-start with suitable breakpoint*

```
 1  program myprog
 2    use mod_stuff
 3    implicit none
 4    integer :: i
 5    allocate(my_array(10))
 6    call modify(5, 2.5)
 7    i = 15
 8    call modify(i, 4.5)
 9    deallocate(my_array)
10    call modify(5, 4.5)
11  end program
```

Fortran specific tools

- **Selecting**
  Tools → Fortran Modules
  and then clicking on one of the appearing modules in the selection
  - overview of all global entities
  - again – after starting program!
- **Derived type debugging is also supported**
  - dive into type components



©2009 LRZ

Example 3: Display and visualize array data

- Have a program with a 2D array
  - dive into array → huge amounts of data
  - difficult to inspect
- Various filtering possibilities
  - select an array slice by editing the Slice field
  - e.g., `(1:220:10, 5)`
  - if only very few values are abnormal, this still may not help
  - revert to `(:,:)` and ...

aa - ./myprog.exe - 1.1

File Edit View Tools Window Help

1.1

Expression: aa Address: 0x0058c180
Slice: (:,:) Filter:
Type: REAL*8(223,223)

Field	Value
(1,1)	0.00999783334341645
(2,1)	0.00999183453409133
(3,1)	0.00998183903342722
(4,1)	0.00996785093023914
(5,1)	0.00994987581958188
(6,1)	0.00992792089126003
(7,1)	0.00990199492695218
(8,1)	0.00987210829669839
(9,1)	0.00983827295475228

aa - ./myprog.exe - 1.1

File Edit View Tools Window Help

1.1

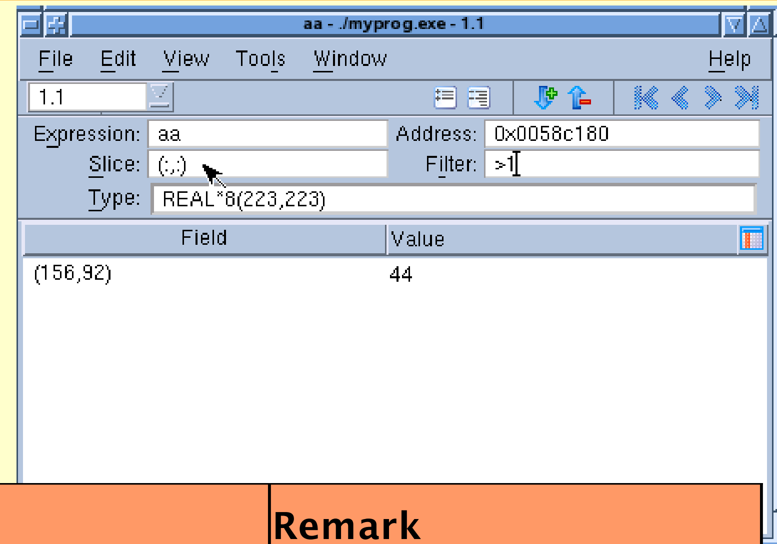
Expression: aa Address: 0x0058c180
Slice: (1:220:10,5) Filter:
Type: REAL*8(223,223)

Field	Value
(1,5)	0.0499691737700142
(11,5)	0.0487745438109176
(21,5)	0.0456354267070532
(31,5)	0.0406769691512952
(41,5)	0.0340968491990742
(51,5)	0.0261573954681158
(61,5)	0.0171751289242783
(71,5)	0.00750814418948334
(81,5)	-0.00245816656082723

©2009 LRZ

Applying filters

- Enter „>1“ into „**Filter**“ field
  - can make exceptional values stand out like sore thumb

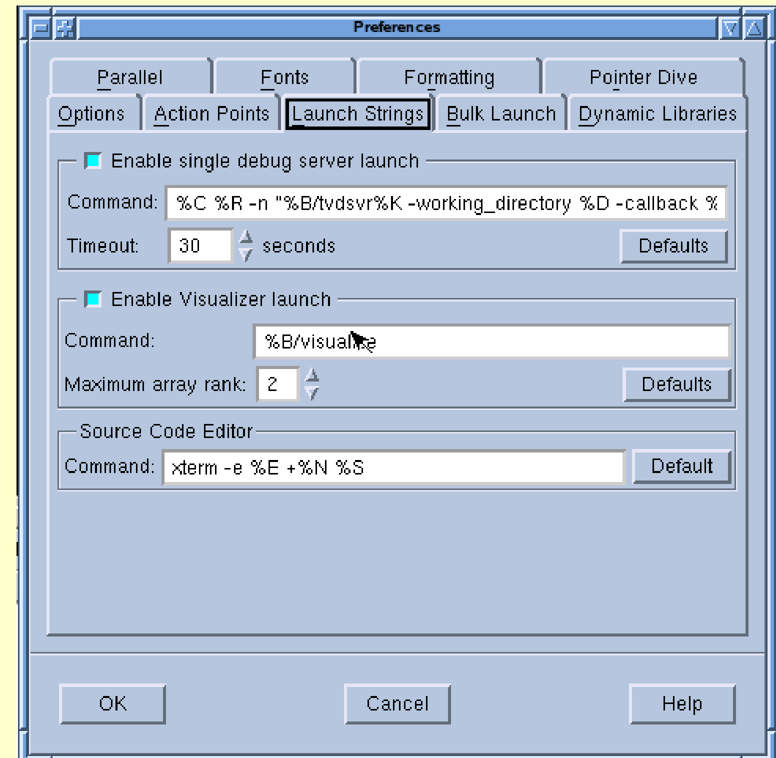


Filter expression	What will it do?	Remark
<=5.2	display elements smaller OR equal 5.2	
!=0	display elements NOT equal zero	
==$nan	display elements with value „Not A Number“	only == or != operation allowed
==$inf	display elements with value +/– Infinity	only == or != operation allowed
-4.3:5.6	display elements with values between –4.3 and 5.6, inclusive	
>-4.3:<5.6	display elements with values between –4.3 and 5.6, exclusive	
($value > 0 && $value < 50) !! ($value > 100)	display elements with value either between 0 and 50 or larger than 100	

©2009 LRZ

Array Visualizer – graphical display of data

- **Open File → Preferences Menu:**
  - **select „Launch Strings“ tab**
  - **check whether the „Enable Visualizer Launch“ button is active**
  - **may also need to change rank entry**
  - **default command should do fine**
    - *could attach own visualizer if so desired*



Visualizing

- Can e.g., visualize 2D slice of a 3D array
  - need to switch filtering off
  - select Tools → Visualize in variable window