TECHNISCHE UNIVERSITÄT DRESDEN

Center for Information Services and High Performance Computing (ZIH)

Parallel Debugging with Allinea DDT

Parallel Programming Course, Dresden, 8.- 12. February 2016

Matthias Lieber (matthias.lieber@tu-dresden.de)

ZIH
Center for Information Services & High Performance Computing

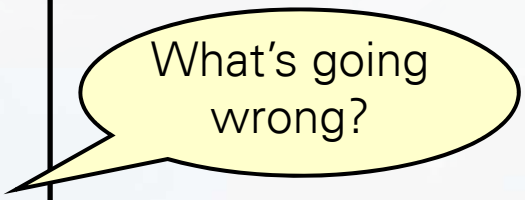
Why using a Debugger?

- Your program…
  - terminates abnormally
  - produces wrong results
  - shows incomprehensible behavior

➔ **You want to know what your program is (really) doing**

- Typical example: your program crashes with a segmentation fault

```
% icc myprog.c –o myprog
% ./myprog
Segmentation fault
%
```



What can a Debugger do?

- **Observe a running program:**
  - Print variables (scalars, arrays, structures / derived types, classes)
  - Inform about current source code line and function (function call stack)

- **Control the program execution:**
  - Stop the program at a specific source code line (**Breakpoints**)
  - Stop the program by evaluating variable expressions (**Conditional Breakpoints** and **Watchpoints**)
  - Stop the program before terminating abnormally
  - Execute the program line-by-line (**Stepping**)

Typical Usage of a Debugger

- Compile the program with the **`-g`** compiler flag
  - **`gcc -g myprog.c -o myprog`**
- Run the program under control of the debugger:
  - **`ddt ./myprog`**
  - Locate the position of the problem and examine variables
  - Find out why the program shows unexpected behavior
- Edit the source code, modify parameters, etc.
- Repeat until problem is solved

Debugger Operation Modes

- **Start program under debugger control**
  - Most common way to use a debugger
  - Not useful if you want to observe what the program does after a long runtime
- **Attach to an already running program**
  - Program was not started under debugger
  - Useful if program has been running for a long time
- **Core files** / core dumps
  - Core files are memory state of a crashed program written to disk
  - Only static analysis of program's data after termination
  - Useful if you don't expect a crash or don't want to wait until a crash happens (probably after long runtime)

Before you start using a Debugger…

- **Use compiler's check capabilities** like **`-Wall`** etc.
  - Read compiler's manual: **`man {gcc|ifort|pgf90|...}`**
  - Intel C: **`-Wall -Wp64 -Wuninitialized -strict-ansi`**
  - Intel Fortran: **`-warn all -std95 -C -fpe0 -traceback`**

- **Always compile your application with the –g flag**, especially during developing and testing
  - Adds symbolic debug info to binary, no performance impact

- Optimizations often interfere with debugging (e.g. functions or variables of interest are "optimized away")
  - If necessary, **compile with –O0 to disable optimizations**

Allinea DDT (Distributed Debugging Tool)

- Commercial debugging tool by Allinea
- C, C++, Fortran
- Parallel Support: pThreads, OpenMP, MPI, PGAS languages, CUDA, OpenACC, Xeon Phi
- Available for all common HPC platforms
- Intuitive graphical user interface
- Advanced features:
  - Visualization of array contents
  - Memory debugging
  - Modify variables
- More info: http://www.allinea.com

Allinea DDT: MPI Program Start

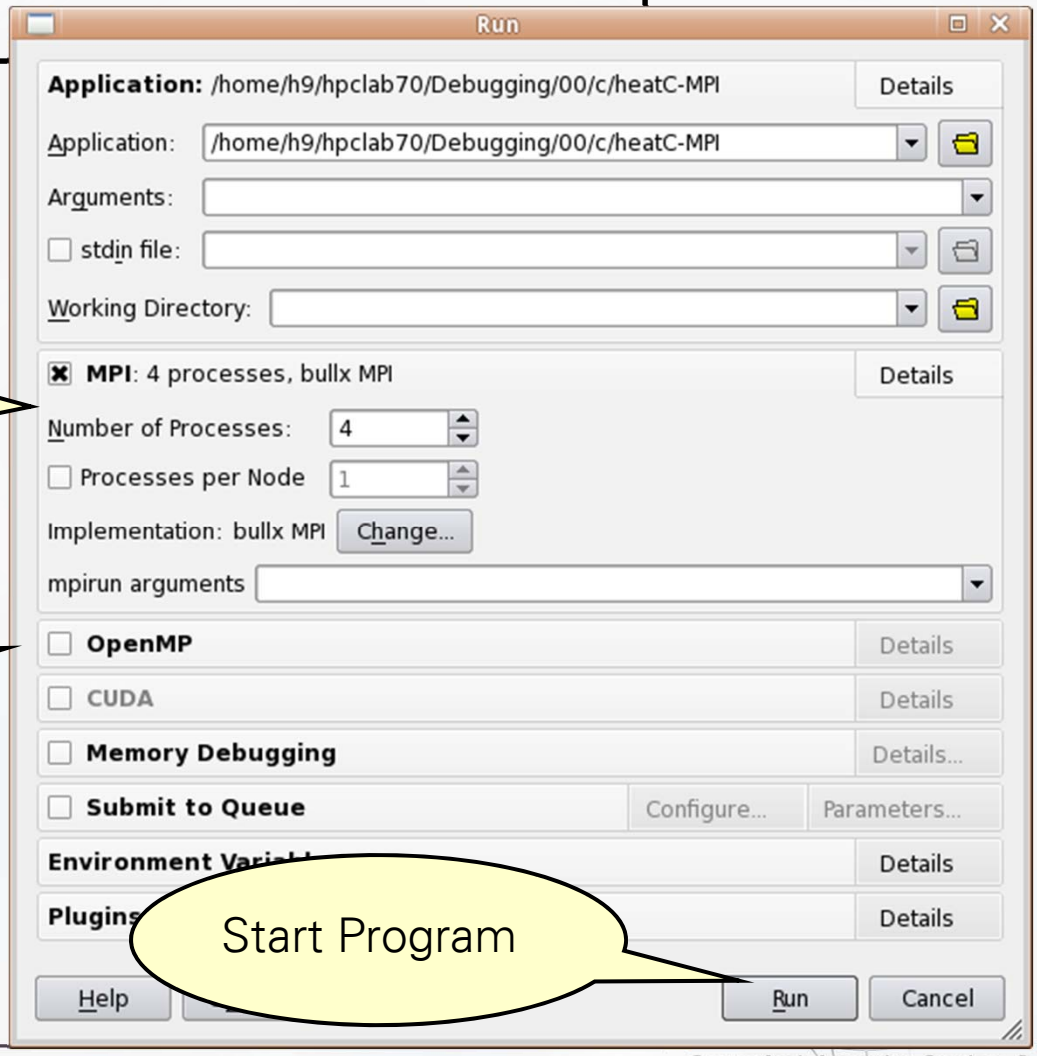
```
% mpicc -g -O0 heatC-MPI.c -o heatC-MPI
% ddt ./heatC-MPI
```

Compile with Debugging

Start DDT

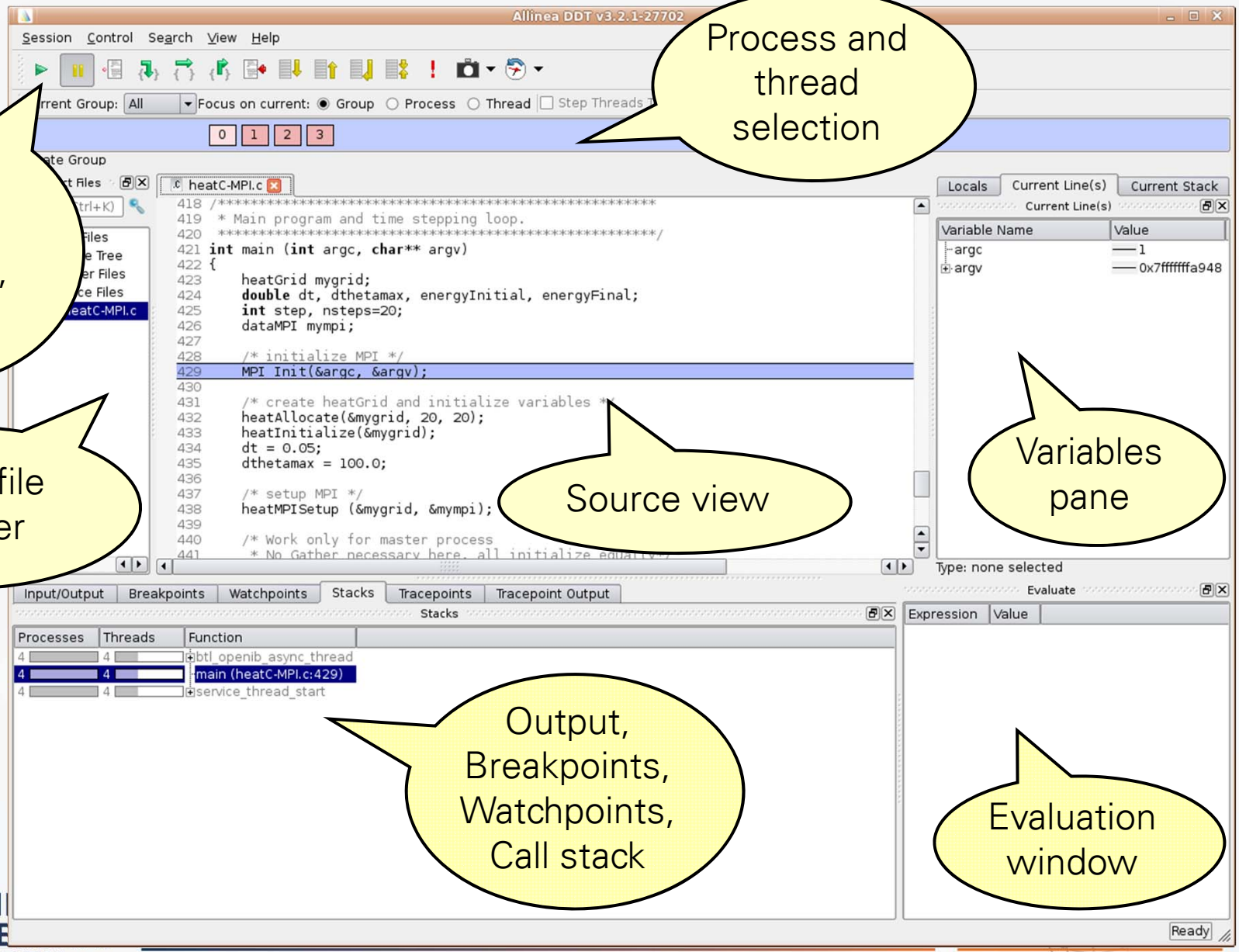
Set MPI implementation and number of MPI processes

… and / or number of OpenMP threads



Start Program

Allinea DDT: Main Window



Process and thread selection

Process control: run, stop, stepping

Source file browser

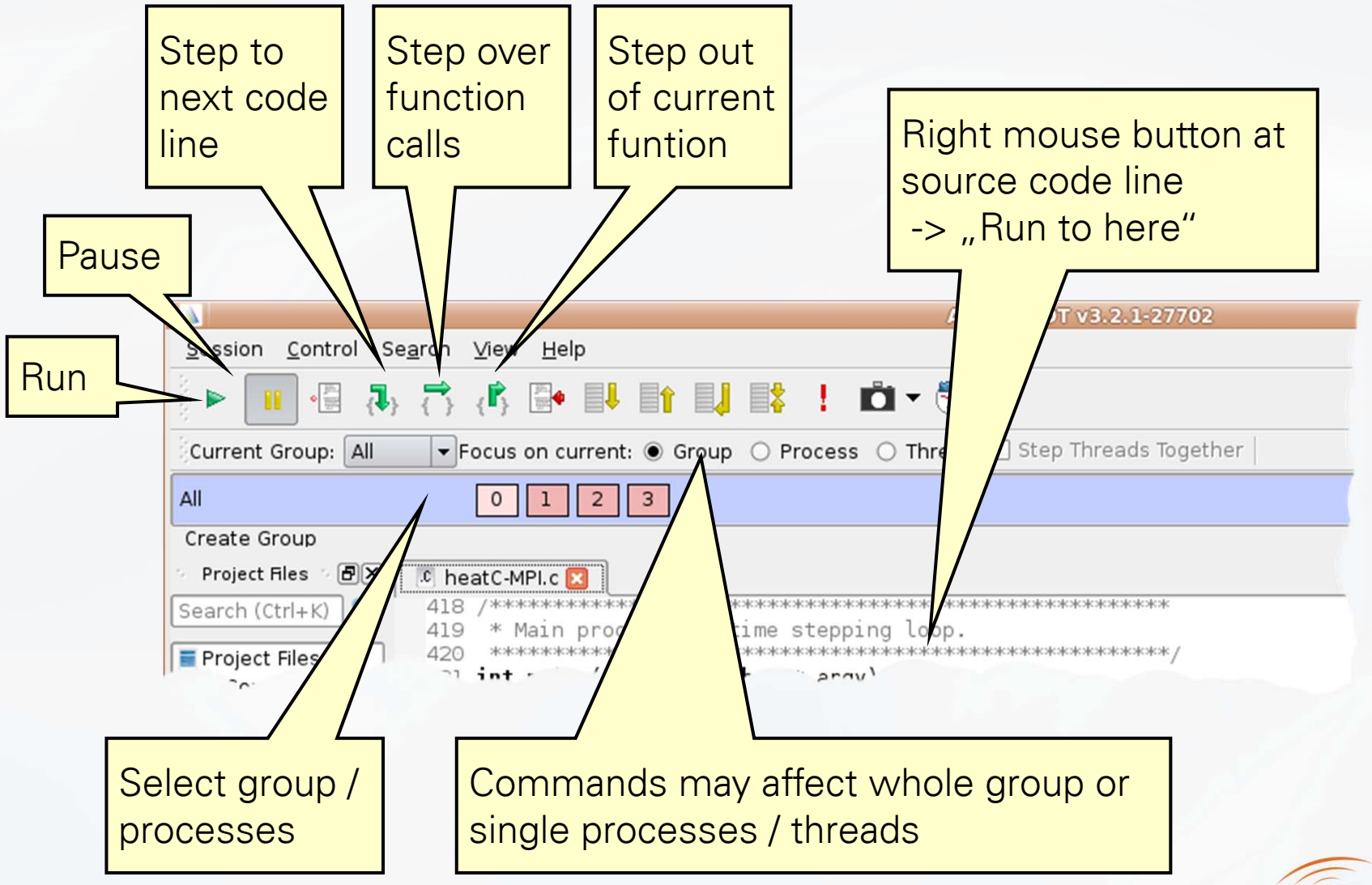
Source view

Variables pane

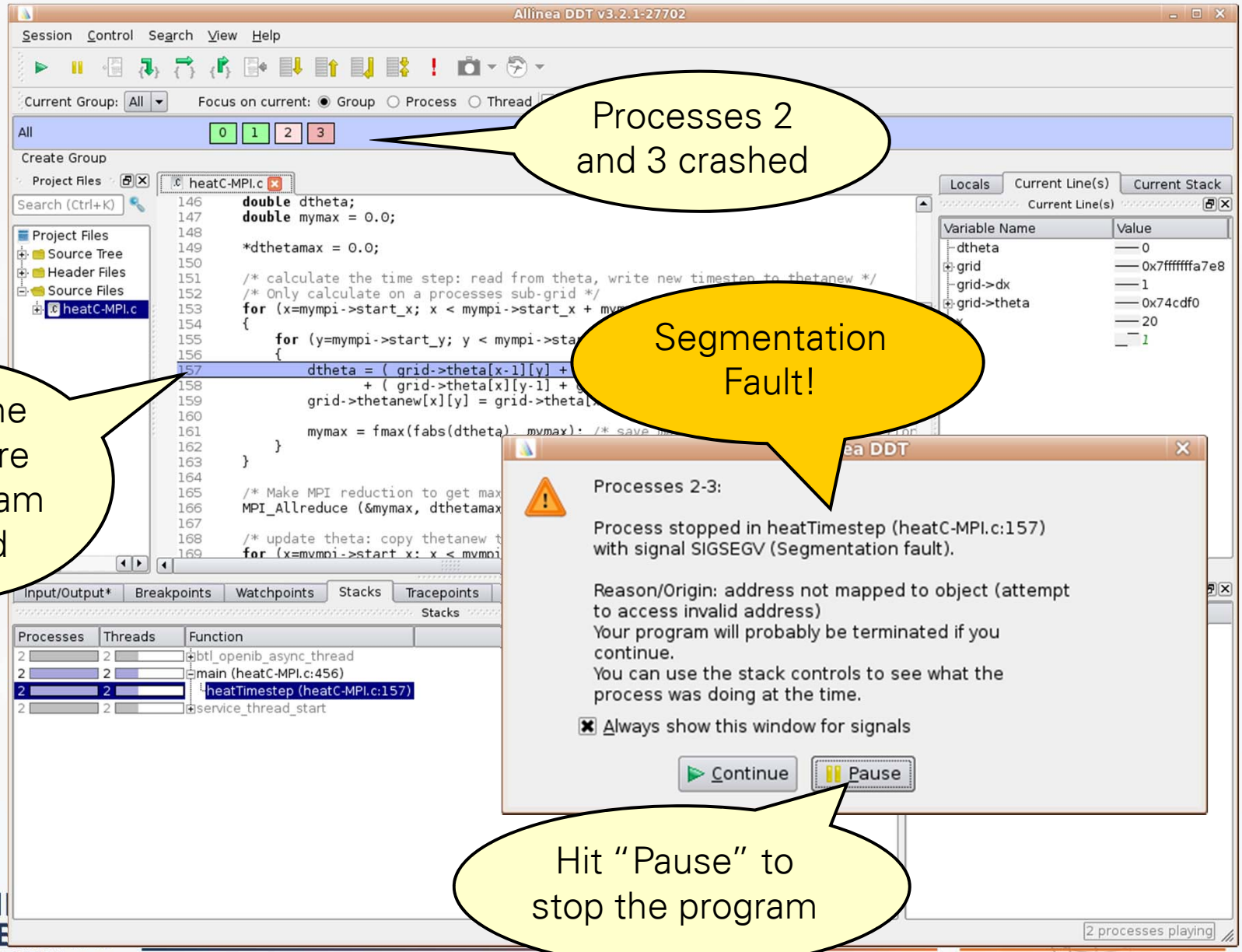
Output, Breakpoints, Watchpoints, Call stack

Evaluation window

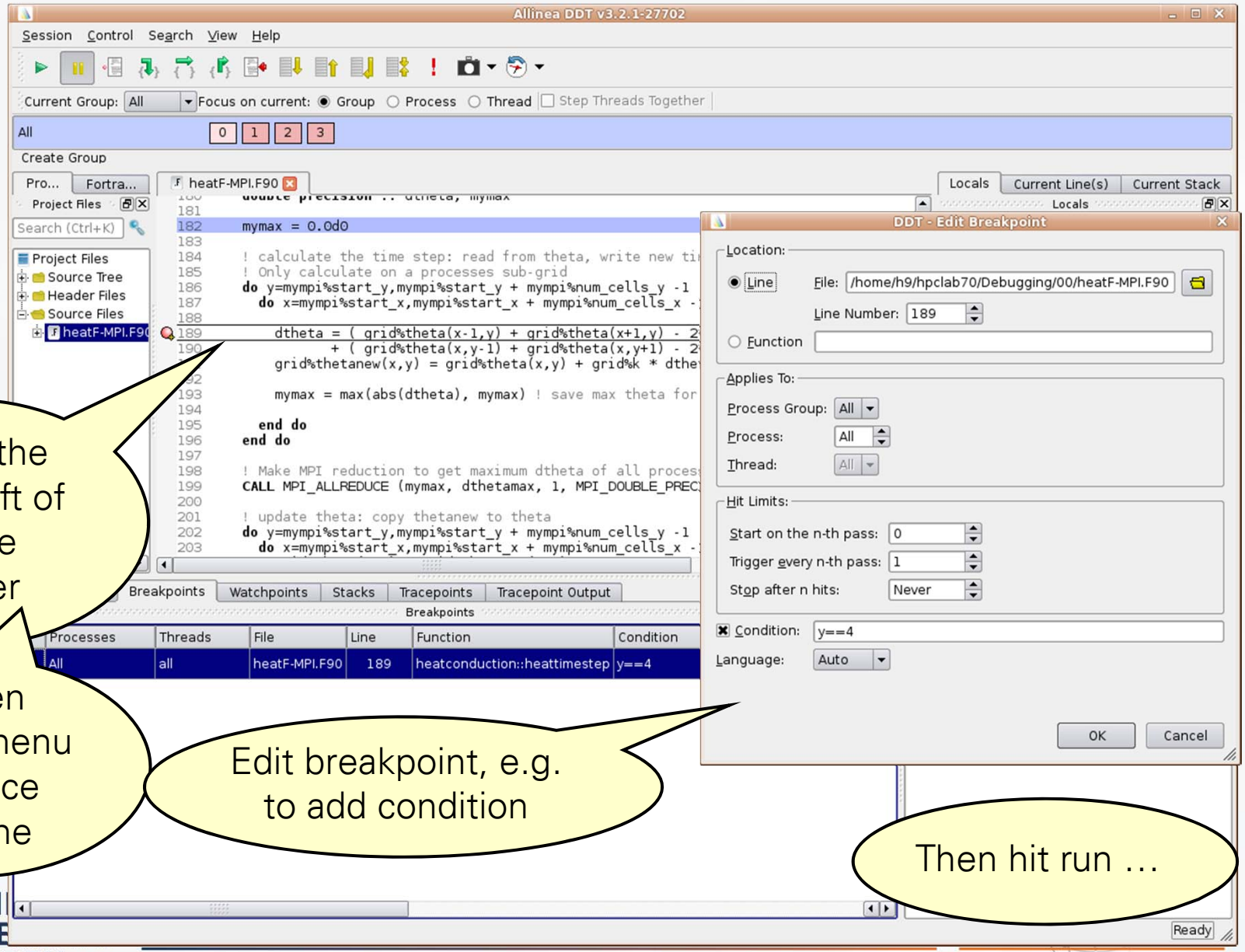
Allinea DDT: Process Control & Stepping



Allinea DDT: Segmentation Fault



Allinea DDT: Breakpoints (1)



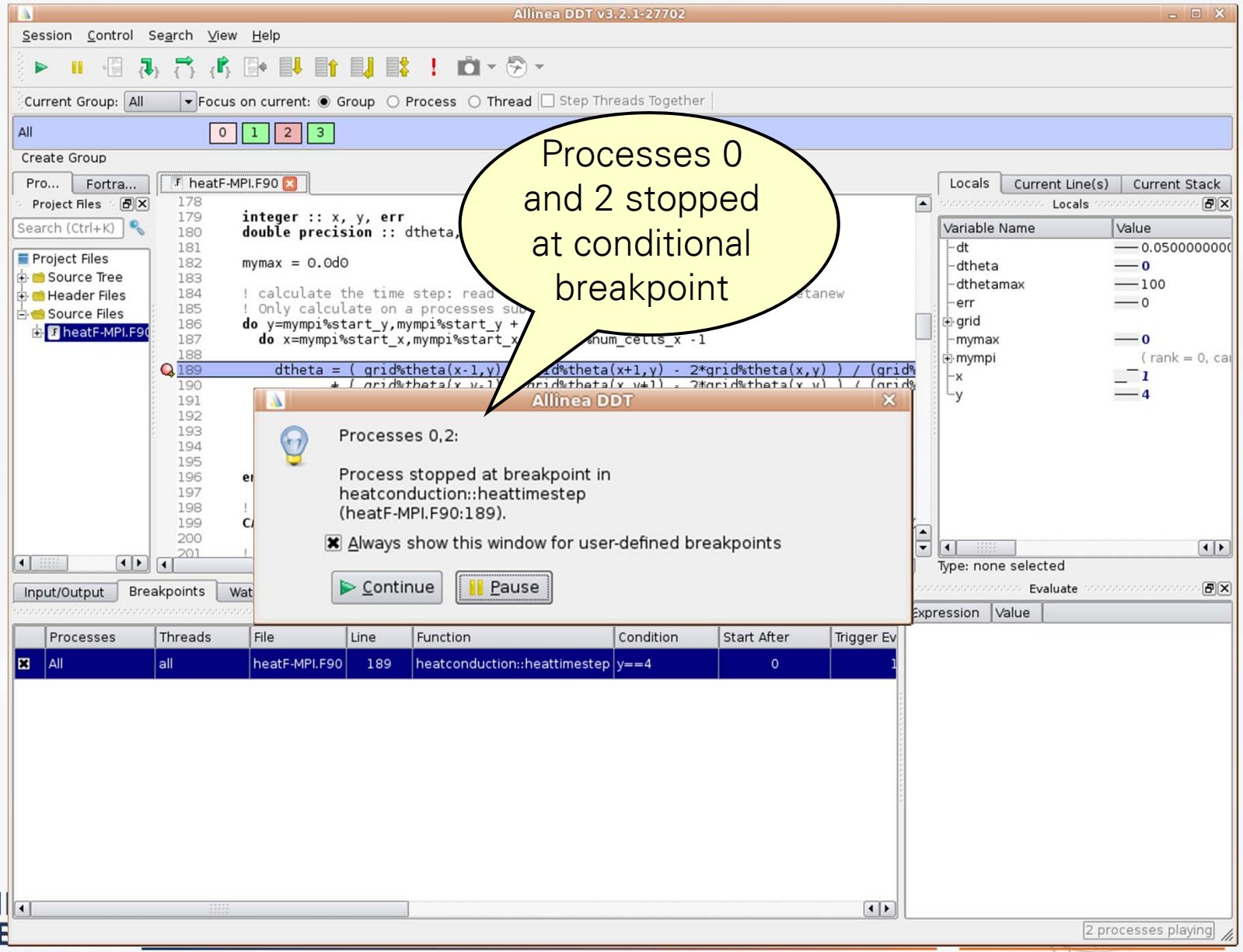
Click to the margin left of the line number

Or open context menu on source code line

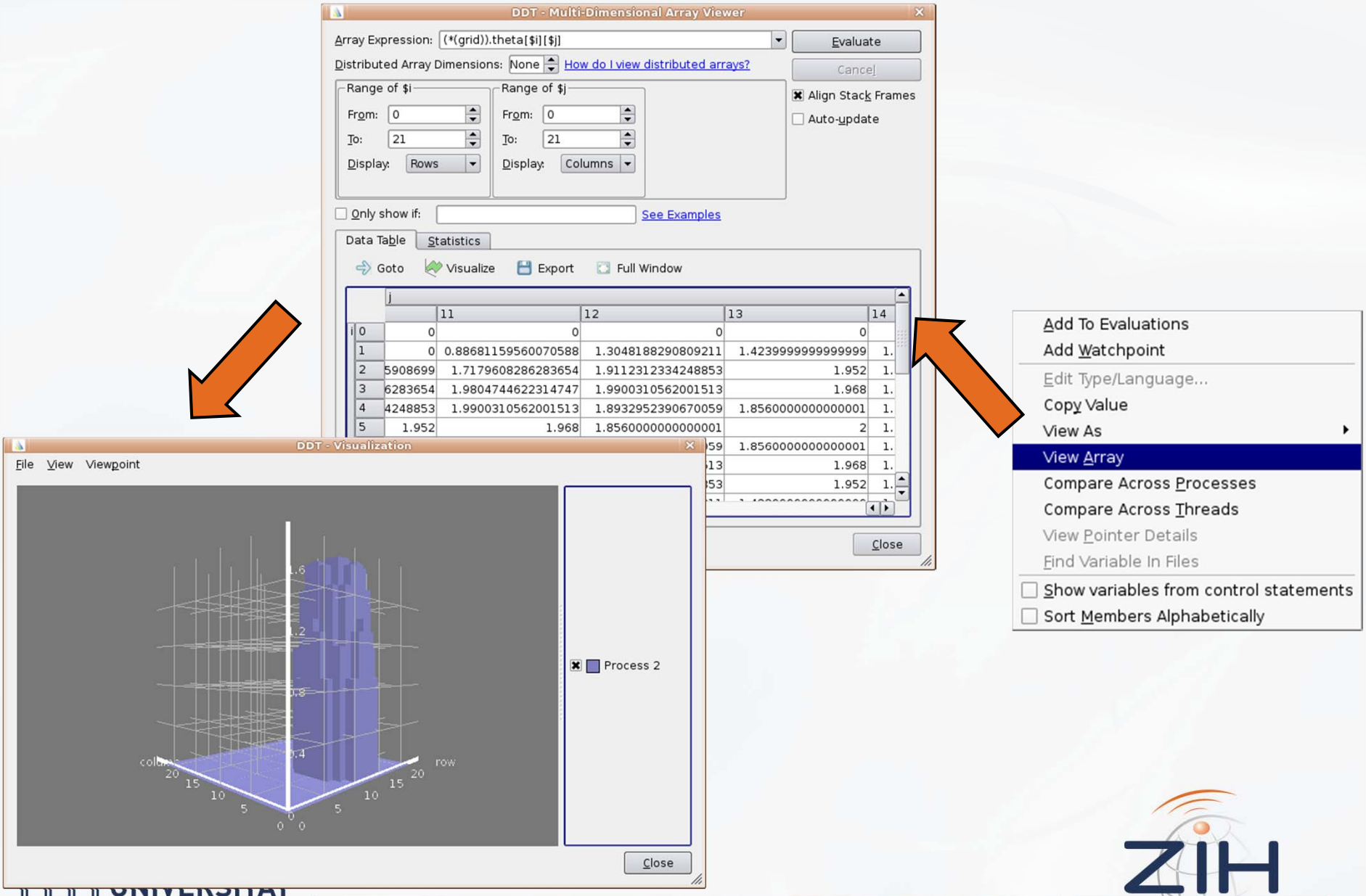
Edit breakpoint, e.g. to add condition

Then hit run …

Allinea DDT: Breakpoints (2)



Allinea DDT: Array Visualization



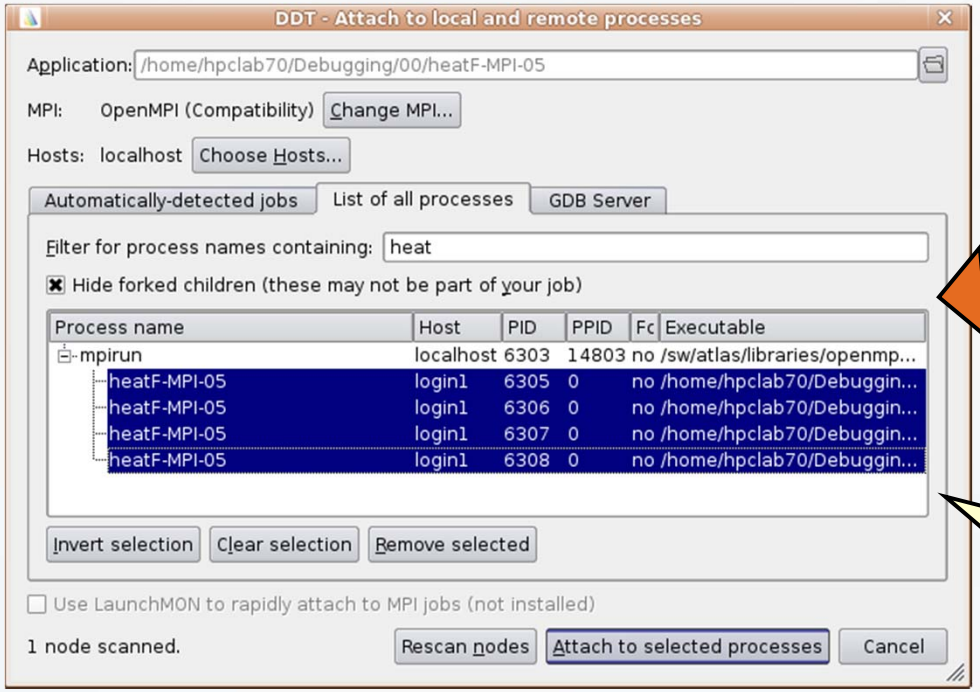
Allinea DDT: Attach to running program

```
% mpif90 -g heatF-MPI-05.F90 -o heatF-MPI-05
% mpirun -np 4 ./heatF-MPI-05
. . .
```

Programs runs – you want to know what it is doing?

Start DDT in a 2nd terminal:

```
% ddt
```



Select your program's processes

Attach

Allinea DDT: Core Files (1)

```
% mpif90 -g heatF-MPI-01.F90 -o heatF-MPI-01
% ulimit -c
1
% ulimit -Sc 100000
% export decfort_dump_flag=yes
% mpirun -np 2 ./heatF-MPI-01
. . .

-----------------------------------------
mpirun noticed that process rank 0 with PID 27934 on node login1
exited on signal 11 (Segmentation fault).
-----------------------------------------
% ls -l *.core
-rw------- 1 hpclab70 zih-hpclab 76M 10. Feb 11:03 login1.27934.core
-rw------- 1 hpclab70 zih-hpclab 76M 10. Feb 11:03 login1.27935.core
% ddt
```

Check core file size limit (reports kB) and increase if required (sets to 100 MB)

Intel Fortran only

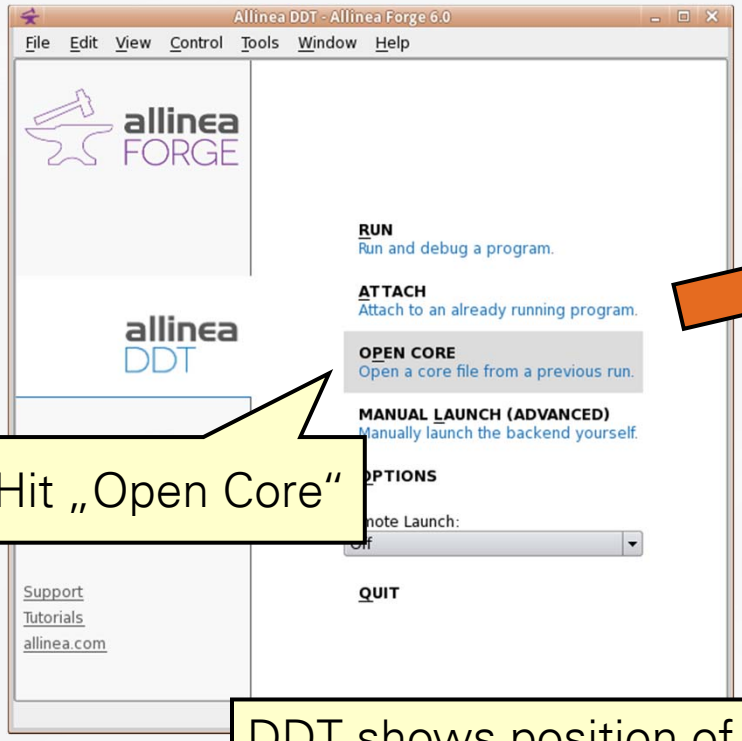
Run program

Program crashes

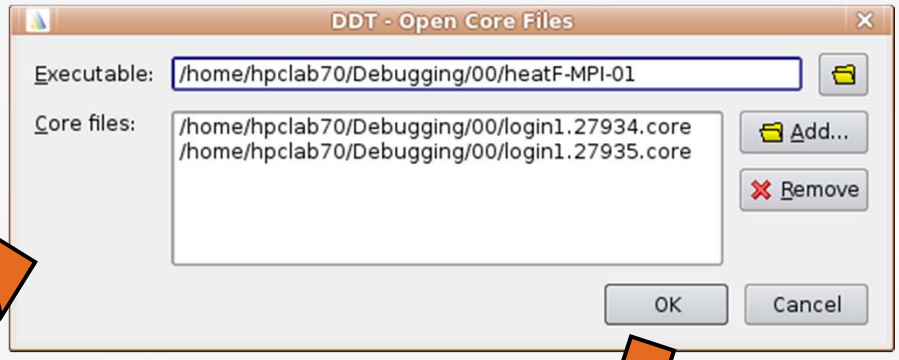
Corefiles created

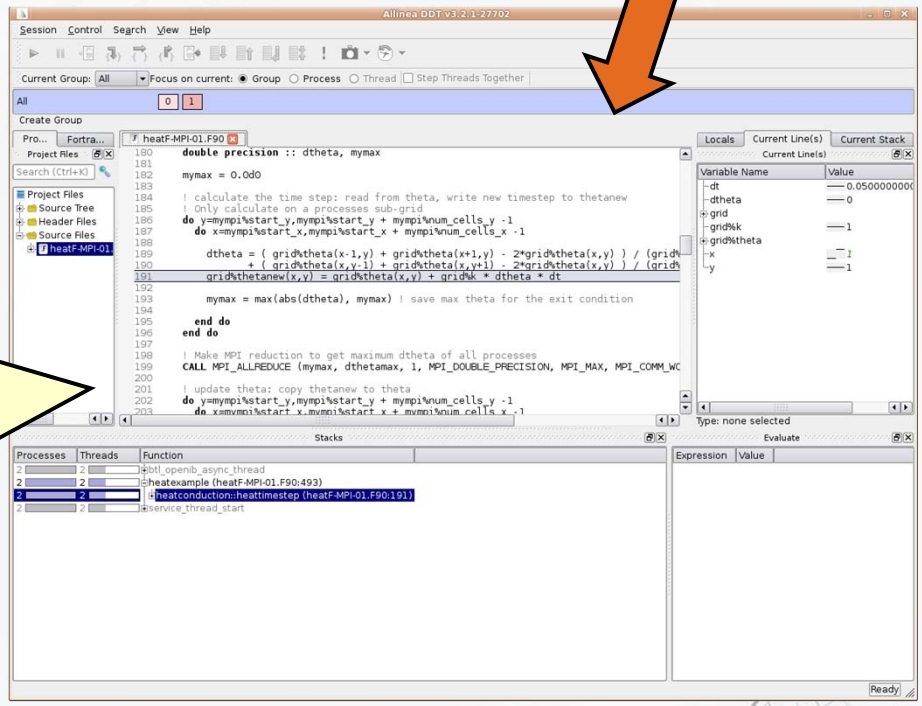
Analyze with DDT

Allinea DDT: Core Files (2)



Hit „Open Core“





DDT shows position of crash in source code

DDT shows variables at the time of the crash

But no stepping possible!