



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)

# 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's going wrong?

# 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

**Run**

**Application:** /home/h9/hpclub70/Debugging/00/c/heatC-MPI Details

**Application:** /home/h9/hpclub70/Debugging/00/c/heatC-MPI Folder icon

**Arguments:** Text field

☐ **stdin file:** Text field Folder icon

**Working Directory:** Text field Folder icon

☒ **MPI:** 4 processes, bullx MPI Details

**Number of Processes:** 4 Up/Down arrows

☐ **Processes per Node:** 1 Up/Down arrows

**Implementation:** bullx MPI Change...

**mpirun arguments:** Text field

☐ **OpenMP** Details

☐ **CUDA** Details

☐ **Memory Debugging** Details...

☐ **Submit to Queue** Configure... Parameters...

**Environment Variables** Details

**Plugins** Details

Help Run Cancel

Start Program



# Allinea DDT: Main Window

The screenshot shows the Allinea DDT v3.2.1-27702 main window. The interface includes a menu bar (Session, Control, Search, View, Help), a toolbar with various debugging icons, and a status bar at the bottom. The central pane displays the source code of `heatC-MPI.c`, with line 429 (`MPI_Init(&argc, &argv);`) highlighted. To the left is a source file browser showing a tree of files. To the right is a variables pane showing the current values of `argc` (1) and `argv` (0x7fffffffa948). Below the source code is a stack pane showing the current call stack, with `main (heatC-MPI.c:429)` selected. At the bottom is an evaluation window for expressions and values. Callouts identify the following components:

- Process control: run, stop, stepping
- Source file browser
- Source view
- Process and thread selection
- Variables pane
- Output, Breakpoints, Watchpoints, Call stack
- Evaluation window

# Allinea DDT: Process Control & Stepping

The screenshot shows the Allinea DDT interface with several callouts explaining the controls:

- Run**: Points to the green play button in the toolbar.
- Pause**: Points to the yellow pause button in the toolbar.
- Step to next code line**: Points to the green arrow with a vertical line in the toolbar.
- Step over function calls**: Points to the green arrow with a bracket in the toolbar.
- Step out of current function**: Points to the green arrow with a large bracket in the toolbar.
- Right mouse button at source code line -> „Run to here“**: Points to the source code window.
- Select group / processes**: Points to the "All" group selection in the "Current Group" dropdown.
- Commands may affect whole group or single processes / threads**: Points to the "Focus on current" radio buttons (Group, Process, Thread).

The source code window displays the following code:

```
418 /*****  
419 * Main program time stepping loop.  
420 *****/  
int main(int argc, char* argv)
```

# Allinea DDT: Segmentation Fault

Processes 2 and 3 crashed

This is the line where the program crashed

Segmentation Fault!

Hit "Pause" to stop the program

Processes 2-3:  
Process stopped in heatTimestep (heatC-MPI.c:157) with signal SIGSEGV (Segmentation fault).  
Reason/Origin: address not mapped to object (attempt to access invalid address)  
Your program will probably be terminated if you continue.  
You can use the stack controls to see what the process was doing at the time.  
☒ Always show this window for signals  
Continue Pause

Variable Name	Value
dtheta	0
grid	0x7fffffffa7e8
grid->dx	1
grid->theta	0x74cdf0
	20
	1

Processes	Threads	Function
2	2	btlib_openib_async_thread
2	2	main (heatC-MPI.c:456)
2	2	heatTimestep (heatC-MPI.c:157)
2	2	service_thread_start

# 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 ...

DDT - Edit Breakpoint

Location:

- ☒ Line: File: /home/h9/hpclab70/Debugging/00/heatF-MPI.F90 Line Number: 189
- ☐ Function

Applies To:

- Process Group: All
- Process: All
- Thread: All

Hit Limits:

- Start on the n-th pass: 0
- Trigger every n-th pass: 1
- Stop after n hits: Never

☒ Condition: y==4

Language: Auto

OK Cancel

Processes	Threads	File	Line	Function	Condition
All	all	heatF-MPI.F90	189	heatconduction::heattimestep	y==4



# Allinea DDT: Breakpoints (2)

Processes 0 and 2 stopped at conditional breakpoint

Processes 0,2:

Process stopped at breakpoint in  
heatconduction::heattimestep  
(heatF-MPI.F90:189).

☒ Always show this window for user-defined breakpoints

Locals

Variable Name	Value
dt	0.0500000000
dtheta	0
dthetamax	100
err	0
grid	0
mymax	0
mympi	(rank = 0, ca
x	1
y	4

Processes	Threads	File	Line	Function	Condition	Start After	Trigger Ev
<input checked="" type="checkbox"/> All	all	heatF-MPI.F90	189	heatconduction::heattimestep	y==4	0	1

2 processes playing

# Allinea DDT: Array Visualization

**DDT - Multi-Dimensional Array Viewer**

Array Expression: `(*(grid)).theta[$i][$j]` Evaluate Cancel

Distributed Array Dimensions: None [How do I view distributed arrays?](#)

Range of  $i$ : From: 0 To: 21 Display: Rows

Range of  $j$ : From: 0 To: 21 Display: Columns

☒ Align Stack Frames ☐ Auto-update

☐ Only show if: See Examples

Data Table Statistics

[Goto](#) [Visualize](#) [Export](#) [Full Window](#)

		11	12	13	14
i 0	0	0	0	0	0
1	0	0.88681159560070588	1.3048188290809211	1.4239999999999999	1.
2	5908699	1.7179608286283654	1.9112312334248853	1.952	1.
3	6283654	1.9804744622314747	1.9900310562001513	1.968	1.
4	4248853	1.9900310562001513	1.8932952390670059	1.8560000000000001	1.
5	1.952	1.968	1.8560000000000001	2	1.

**DDT - Visualization**

File View Viewpoint

3D visualization of array data (Process 2).

☒ Process 2

Close

**Context Menu:**

- Add To Evaluations
- Add Watchpoint
- Edit Type/Language...
- Copy Value
- View As
- View Array**
- Compare Across Processes
- Compare Across Threads
- View Pointer Details
- Find Variable In Files
- ☐ Show variables from control statements
- ☐ Sort Members Alphabetically

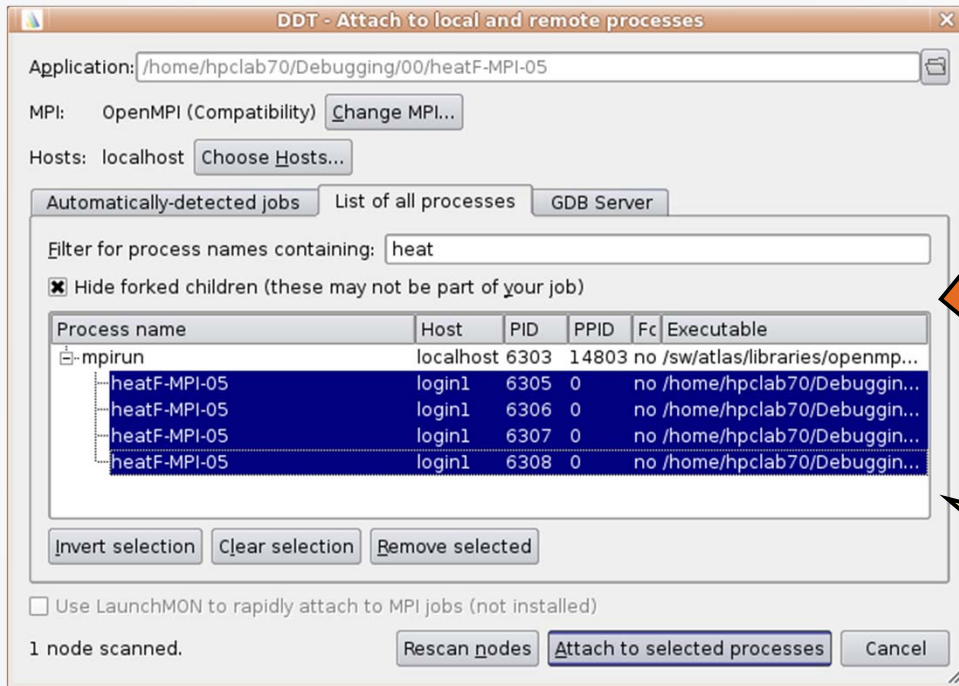
# Allinea DDT: Attach to running program

```
% 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
. . .
```

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

Intel Fortran only

Run program

Program crashes

```
-----
mpirun noticed that process rank 0 with PID 27934 on node login1
exited on signal 11 (Segmentation fault).
-----
```

Corefiles created

```
% 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
```

Analyze with DDT

# Allinea DDT: Core Files (2)

