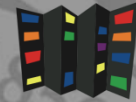


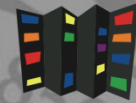


# Introduction to debugging

*Martin Čuma*  
*Center for High Performance*  
*Computing University of Utah*  
*m.cuma@utah.edu*



- Program errors
- Simple debugging
- Graphical debugging
- DDT and Totalview
- Intel tools
- Interpreted languages



- crashes
  - segmentation faults (bad memory access)
    - often writes core file – snapshot of memory at the time of the crash
  - wrong I/O (missing files)
  - hardware failures
- incorrect results
  - reasonable but incorrect results
  - NaNs – not a numbers – division by 0, ...

- write variables of interest into the stdout or file
- simplest but cumbersome
  - need to recompile and rerun
  - need to browse through potentially large output

```
Terminal - u0101881@p8:~/tests
root@p8:/uufs/chpc.utah.edu/sys/buildir/oc... u0101881@p8:~/tests
81 0.80500000000000005      272.32451601382030
82 0.81500000000000006      274.72803717231750
83 0.82500000000000007      277.10810411170019
84 0.83499999999999996      279.46489239846238
85 0.84499999999999997      281.79858064688352
86 0.85499999999999998      284.10935027759365
87 0.86499999999999999      286.39738528452926
88 0.87500000000000000      288.66287201019298
89 0.88500000000000001      290.90599892911797
90 0.89500000000000002      293.12695643942459
91 0.90500000000000003      295.32593666234624
92 0.91500000000000004      297.50313324959058
93 0.92500000000000004      299.65874119839486
94 0.93500000000000005      301.79295667412481
95 0.94500000000000006      303.90597684026102
96 0.95500000000000007      305.99799969561070
97 0.96499999999999997      308.06922391857796
98 0.97499999999999998      310.11984871832163
99 0.98499999999999999      312.15007369262662
100 0.99500000000000000      314.16009869231254
pi is approximately: 3.1416009869231254 Error is: 0.0000833333333323
Enter the number of intervals: (0 quits)
```



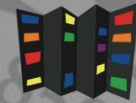
- text only, e.g. gdb, idb
- need to remember commands or their abbreviations
- need to know lines in the code (or have it opened in other window)
- useful for quick code checking on compute nodes and core dump analysis

```

Terminal - u0101881@p8:~/tests
root@p8:/uufs/chpc.utah.edu/sys/buildir/oc...  u0101881@p8:~/tests
Starting program: /uufs/chpc.utah.edu/common/home/u0101881/tests/pi3
warning: File "/opt/at9.0/lib64/power8/libthread_db-1.0.so" auto-loading has been declined by
your 'auto-load safe-path' set to "$debugdir:$datadir/auto-load".
To enable execution of this file add
  add-auto-load-safe-path /opt/at9.0/lib64/power8/libthread_db-1.0.so
line to your configuration file "/uufs/chpc.utah.edu/common/home/u0101881/.gdbinit".
To completely disable this security protection add
  set auto-load safe-path /
line to your configuration file "/uufs/chpc.utah.edu/common/home/u0101881/.gdbinit".
For more information about this security protection see the
"Auto-loading safe path" section in the GDB manual.  E.g., run from the shell:
  info "(gdb)Auto-loading safe path"
warning: Unable to find libthread_db matching inferior's thread library, thread debugging will
not be available.
Process          0 of          1 is alive
Enter the number of intervals: (0 quits)
100000

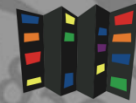
Breakpoint 1, MAIN__ () at pi3.f:50
50      if ( n .le. 0 ) goto 30
(gdb) p n
$1 = 100000
(gdb)

```



- have graphical user interface
- freeware or commercial
- DDD, GDBFrontend
- Eclipse CDT, VSCode
- Intel development tools
- Rogue Wave Totalview - commercial
- Allinea DDT - commercial

- The only real alternative for parallel or accelerator debugging
- Cost a lot of money (thousands of \$), but, worth it
- We had Totalview license (for historical reasons), 32 tokens enough for our needs (renewal ~\$1500/yr)
- In 2017 we switched to DDT which gave us competitive upgrade
- XSEDE/ACCESS systems have DDT



## 1. Compile binary with debugging information

- flag -g

```
gcc -g test.f -o test
```

## 2. Load module and run DDT

```
module load ddt
```

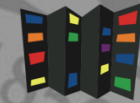
- DDT + executable

```
ddt ./executable
```

- DDT + core file

```
ddt executable core_file
```

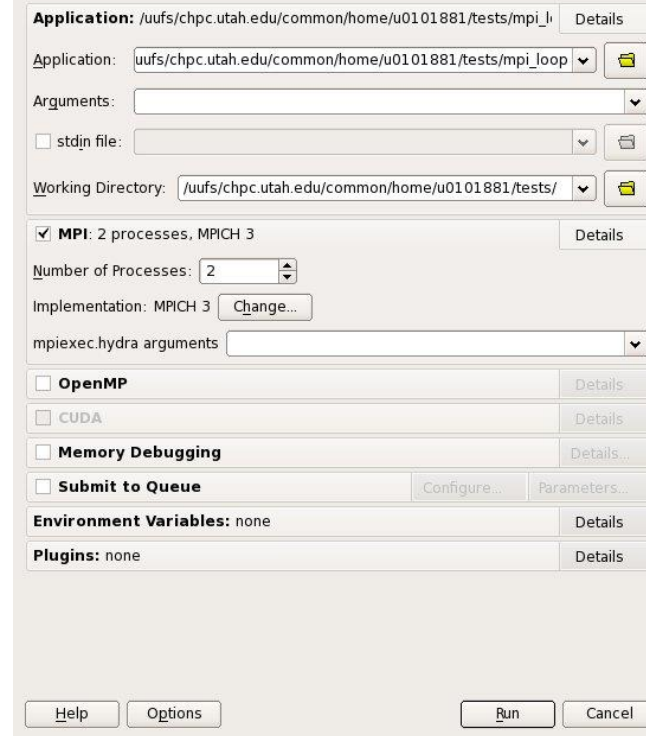




- run DDT and attach the executable
  - start DDT
  - fill in information in the Run dialog
  - click Run button to start the program
- run DDT and attach running program
  - start DDT
  - click “Attach to an already running program”
  - choose process and click “Attach to” button

### 3. DDT operation

- left mouse button - select
- right mouse button – context sensitive menu



The screenshot shows the DDT Run dialog box with the following configuration:

- Application:** /uufs/chpc.utah.edu/common/home/u0101881/tests/mpi\_l (Details)
- Application:** uufs/chpc.utah.edu/common/home/u0101881/tests/mpi\_loop (icon)
- Arguments:** (empty field)
- stdjn file:** (empty field) (icon)
- Working Directory:** /uufs/chpc.utah.edu/common/home/u0101881/tests/ (icon)
- MPI: 2 processes, MPICH 3** (Details)
- Number of Processes:** 2 (spin box)
- Implementation:** MPICH 3 (Change...)
- mpiexec.hydra arguments:** (empty field)
- OpenMP** (Details)
- CUDA** (Details)
- Memory Debugging** (Details...)
- Submit to Queue** (Configure... Parameters...)
- Environment Variables:** none (Details)
- Plugins:** none (Details)

Buttons at the bottom: Help, Options, Run, Cancel



Array Expression:

Distributed Array Dimensions:  [How do I view distributed arrays?](#)

Staggered Array [What does this do?](#)  Align Stack Frames  Auto-update

Range of \$i

From:  **Array viewer window**

To:

Display:

Only show if:  [See Examples](#)

Data Table  Statistics

i	0	0
1	0	0
2	0	0
3	0	0
4	0	0
5	0	0
6	0	0
7	0	0
8	0	0
9	0	0
10	0	0
11	0	0
12	0	0

Expression:

Processes in current group (All, 2 procs)  Align stack frames

Limit comparison to  significant figures

Only show if:  [See Examples](#)

Use as MPI Rank

Values	Process(es)
0	0
1	1

Statistics

Count: 2  
 Not shown: 0  
 Errors: 0  
 Aggregate: 0  
 Numerical: 2  
 Sum: 1  
 Minimum: 0  
 Maximum: 1  
 Range: 1  
 Mean: 0.5  
 Variance: 0.5  
 Std. Dev.: 0.707106781  
 -nan: 0  
 -inf: 0  
 -inf: 0  
 <0: 0  
 =0: 1  
 >0: 1

File Edit View Control Tools Window Help

Current Group: All Focus on current:  Group  Process  Thread  Step Threads Together

All

Create Group

Project Files

Search (Ctrl+K)

Application Code

Sources

External Code

main(int argc, char \*argv)

mpi\_loop.c

```

15 MPI_Init(&argc, &argv);
16
17 MPI_Comm_size(MPI_COMM_WORLD, &nprocs);
18 MPI_Comm_rank(MPI_COMM_WORLD, &rank);
19
20
21 MPI_Barrier(MPI_COMM_WORLD);
22 printf("This is rank %d of %d reporting for duty\n",
23        rank, nprocs);
24 MPI_Barrier(MPI_COMM_WORLD);
25
26 //free(k);
27
28 //fp = fopen("myfile", "r");
29 //fscanf(fp, "%d\n", i);
30
31 for (i=rank; i < ARRSIZE+100; i+=nprocs)
32 {
33     j[i] = i;
34     j[i] = j[i]+i;
35 }
36
37 MPI_Reduce(k, j, ARRSIZE, MPI_INT, MPI_SUM, 0, MPI_COMM_WORLD);
38
39 if (rank==0) printf("Last element: %d\n", j[ARRSIZE-1]);
40 MPI_Finalize();
41 return EXIT_SUCCESS;
42
43
  
```

**Main window**

Input/Output Breakpoints Watchpoints Stacks Tracepoints Tracepoint Output Logbook

Input/Output

```

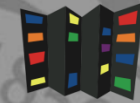
[1506547499.888709] [frisco8:9623 :0] sys.c:744 MXM WARN Conflicting CPU frequencies detected, using: 3244.69
This is rank 0 of 2 reporting for duty
This is rank 1 of 2 reporting for duty
  
```

Note: Allinea DDT can only send input to the mpiexec.hydra process with this MPI implementation

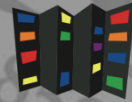
Type here ('Enter' to send):

**Process window**

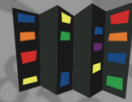
Rank	Host	PID	Main Thread
0	frisco8	9622	LWP 9622
1	frisco8	9623	LWP 9623



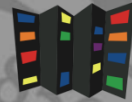
- Data examination
  - view data in the variable windows
  - change the values of variables
  - modify display of the variables
  - visualize data
- Action points
  - breakpoints and barriers (static or conditional)
  - watchpoints and tracepoints
  - evaluation of expressions



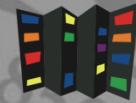
- Automatic attachment of child processes
- Create process groups
- Share breakpoints among processes
- Process barrier breakpoints
- Process group single-stepping
- View variables across procs/threads
- Display MPI message queue state



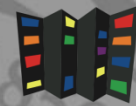
- Load up an existing program
  - DDT windows
  - step through the code
  - place breakpoints
  - examine variables
- Load a core file
  - examine the crash point



- Stack trace – procedure hierarchy - left
- Stack frame – variables display - right
- Source code – code + process navigation - center
- Threads list – in case of multithreaded application - top
- Action points – list of breakpoints, barriers,... - bottom

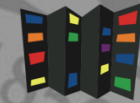


- Breakpoints and barriers
  - toggle location with left mouse (shift for barrier)
  - right-click – Properties for options
  - option conditional breakpoints
- Watchpoints
  - watch for change in a memory location
- Tracepoints
  - write screen output whenever line is executed

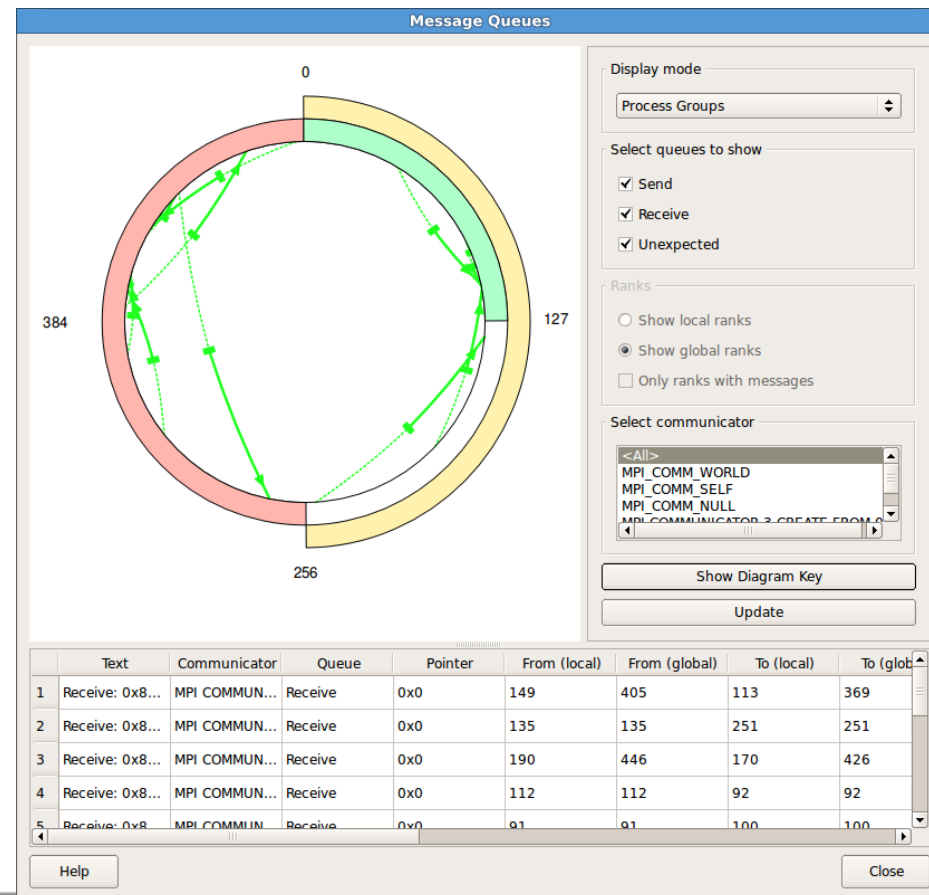


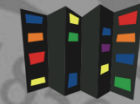
- Scalars in variable view on the right of the main window
- Arrays in Array viewer
  - right click on a variable, choose Array viewer
  - adjust array dimensions, hit evaluate
  - filter array values, e.g.  $\$value > 0$
- variable visualization, statistics,...





- Process/thread groups
- Cross-process/thread data comparison
- Message queue state graph and display

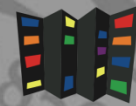




- Enable in the initial Run dialog
  - Memory usage
  - Memory leak detection
  - Exceeding array bounds
  - Dangling pointers

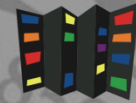


- We license the GPU add-on to DDT
- Works well with CUDA
- With OpenACC, the debugger sees what the compiler generates so stepping through code may not correspond to actual source lines
  - Use breakpoints and examine data at the breakpoints



- Can debug Python modules that are written in C/C++ or Fortran
- Or mixed C/C++, Fortran and Python code
- No stepping, breakpoints, variable view of the Python code
- Run as:  

```
ddt python3 %allinea_python_debug%  
mycode.py
```



- DDT webpage

<https://www.linaroforge.com/linaroDdt/>

- Setting up DDT

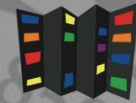
Clusters: `module load ddt`

Some group desktops: `inquire` at CHPC

- Documentation

<https://docs.linaroforge.com/23.0.4/html/forge/ddt/index.html>

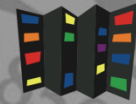
<https://www.chpc.utah.edu/documentation/software/debugging.php#d>



- compilers check for syntax errors
  - some compiler flags help too (-C)
- memory checking tools - many errors are due to bad memory management
  - mtrace – part of GNU tools
  - valgrind – more functionality, but still easy to use

- Free non commercial use
- Tools for all stages of development
  - Compilers and libraries
  - Verification tools
  - Profilers
- More info

<https://www.intel.com/content/www/us/en/developer/tools/oneapi/hpc-toolkit.html>



- Thread checking
  - Data races and deadlocks
- Memory checker
  - Like leaks or corruption
  - Good alternative to Totalview MemoryScape or DDT
- Standalone or GUI integration
- More info

<https://www.intel.com/content/www/us/en/developer/tools/oneapi/inspector.html>



- Source the environment

```
module load inspector
```

- Compile with `-tcheck -g`

```
ifort -openmp -tcheck -g trap.f
```

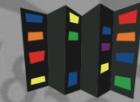
- Run `tcheck`

```
inspxe-gui – graphical user interface
```

```
inspxe-cl – command line
```

- Tutorial

<https://www.intel.com/content/www/us/en/developer/articles/training/inspector-tutorials.html>



- MPI profiler and correctness checker
- Detects violations of MPI standard and errors in execution environment
- To use correctness checker

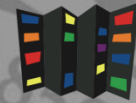
```
module load intel-oneapi-compilers intel-oneapi-mpi  
intel-oneapi-itac  
setenv VT_CHECK_TRACING 0  
mpirun -check-mpi -n 4 ./myApp
```

- ITAC documentation

<https://www.intel.com/content/www/us/en/developer/tools/oneapi/trace-analyzer.html>



- Matlab
  - Debugger built in the editor
- Python
  - gdb like debugging built in (pdb module)
  - Many different third party IDEs (Spyder)
  - PyCharm, but beware of larger memory/CPU use if debugging big data
- R
  - gdb like debugging built in (traceback(), browse(), debug())
  - Rstudio IDE includes graphical debugger



- Terminal debuggers
- Compiler vendor debuggers
- DDT/Totalview for graphical debugging
- Code checkers and memory checkers
- Inspector for thread and memory debugging
- ITAC MPI checker
- Interpreted languages debugging