

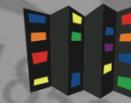


Introduction to debugging

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



- Program errors
- Simple debugging
- Graphical debugging
- Totalview
- Intel tools



Program errors

- 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, ...

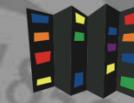


- write variables of interest into the stdout or file
- simplest but cumbersome
 - need to recompile and rerun
 - need to browse through potentially large output

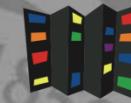


Terminal debuggers

- 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) to
- useful for quick code checking on compute nodes and core dump analysis



- have graphical user interface
- freeware or commercial
- Eclipse CDT - free
- PGI's pdbg – part of PGI compiler suite
- Intel development tools
- Rogue Wave Totalview - commercial
- Allinea DDT - commercial



Totalview in a nutshell

- source and machine level debugger
- command line and graphic interface
- serial and parallel debugging support
- supports remote debugging
- supports memory debugging
- allows stepping back (Replay Engine)
- supports CUDA debugging
- runs on variety of platforms



How to use Totalview

1. Compile binary with debugging information

- flag -g

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

2. Load module and run Totalview

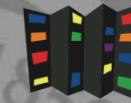
```
module load totalview
```

- TV + executable

```
totalview executable
```

- TV + core file

```
totalview executable core_file
```



How to use Totalview

- run TV and attach the executable
 - start TV
 - Start a Debugging Session window
 - choose an existing program or define a new one
- run TV and attach running program
 - start TV
 - pick “A running program (attach)”
 - choose process ID and executable file name

3. Totalview operation

- left mouse button - select
- right mouse button - menu
- left mouse button double click - dive

Totalview windows



Etnus TotalView 5.0

Attached **Unattached** Groups Log

```

1001795 T mpirun (in __select)
1003847 B4 mpirun<trapp_r>.0 (in main)
1/1003847 B4   in main
1004955 B4 mpirun<trapp_r>.1 (in main)
1/1004955 B4   in main
1004633 B4 mpirun<trapp_r>.2 (in main)
1/1004633 B4   in main
1000345 B4 mpirun<trapp_r>.3 (in main)
1/1000345 B4   in main

```

main

File Edit View Tools Window Help

my_rank (Laminated)

(at **0x7fff2ea4**) Type: int
Filter:

Process	Value
mpirun<trapp_r>.0	0x00000000 (0)
mpirun<trapp_r>.1	0x00000001 (1)
mpirun<trapp_r>.2	0x00000002 (2)
mpirun<trapp_r>.3	0x00000003 (3)

mpirun<trapp_r>.0

File Edit View Group Process Thread Action Point Tools Window Help

Group Control Go Halt Next Step Out Run To Next! Step!

Stack Trace Stack Frame

Process 1003847: mpirun<trapp_r>.0 (At Breakpoint 4)
Thread 1003847.1: mpirun<trapp_r>.0 (At Breakpoint 4)

```

Function "main":
    main,
    __start,
    FP=7fff2f20
    FP=7fff2f30
    argc: 0x00000001 (1)
    argv: 0x7fff2f34 -> 0x7fff30
    Local variables:
        p: 0x00000000 (0)
        tag: 0x00000000 (0)
        my_rank: 0x7fff2fc0 (2147430336)
        status: (Compound Object)
        n: 0x00000001 (1)
        i: 0x00000000 (0)
        local_n: 0x0fb70250 (263651920)
        h: 0x0fb70250

```

Function main in trapp.c

```

33     MPI_Send(&b, 1, MPI_FLOAT, i, tag, MPI_COMM_WORLD);
34     tag = 2;
35     MPI_Send(&n, 1, MPI_INT, i, tag, MPI_COMM_WORLD);
36
37 }
38 else
39 {
40     tag = 0;
41     MPI_Recv(&a, 1, MPI_FLOAT, 0, tag, MPI_COMM_WORLD, &status);
42     tag = 1;
43     MPI_Recv(&b, 1, MPI_FLOAT, 0, tag, MPI_COMM_WORLD, &status);
44     tag = 2;
45     MPI_Recv(&n, 1, MPI_INT, 0, tag, MPI_COMM_WORLD, &status);
46
47
48 STOP
49 h = (b-a)/n;
50 local_n = n/p;
51
52 local_a = a + my_rank*h*local_n;
53 local_b = local_a + h*local_n;
54
55 printf("%d %f %f %d\n", my_rank, local_a, local_b, local_n);

```

Thread (1)

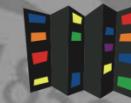
1/1003847 B4 in main

Action Points

- STOP 4 line 16 at main+0x24 in "trapp.c"
- BARR 7 line 27 at main+0x80 in "trapp.c"
- STOP 8 line 49 at main+0x1e8 in "trapp.c"



- 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
 - evaluation of expressions



Multiprocess debugging

- 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



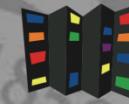
Basic operation example

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

Process view window

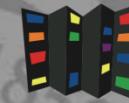


- Stack trace – procedure hierarchy
- Stack frame – variables display
- Source code – code + process navigation
- Threads list – in case of multithreaded application
- Action points – list of breakpoints, barriers, ...



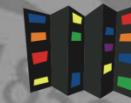
Running the debugger

- Menu Go/Halt/Next/Step/Hold or shortcuts
- Possible actions (thread,process/group):
 - go (g/G)
 - halt (h/H)
 - step (source line) (s/S)
 - step (instruction) (i/I)
 - next (source line) (n/N)
 - next (instruction) (x/X)
 - run (to selection) (r/R)
 - return (out of function) (o/O)



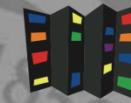
Action points

- Breakpoints and barriers
 - toggle location with left mouse (shift for barrier)
 - right-click – Properties for options
- Evaluation points
 - set conditional breakpoints
 - conditionally patch out code
- Watchpoints
 - watch for change in a memory location

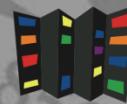


Data examination

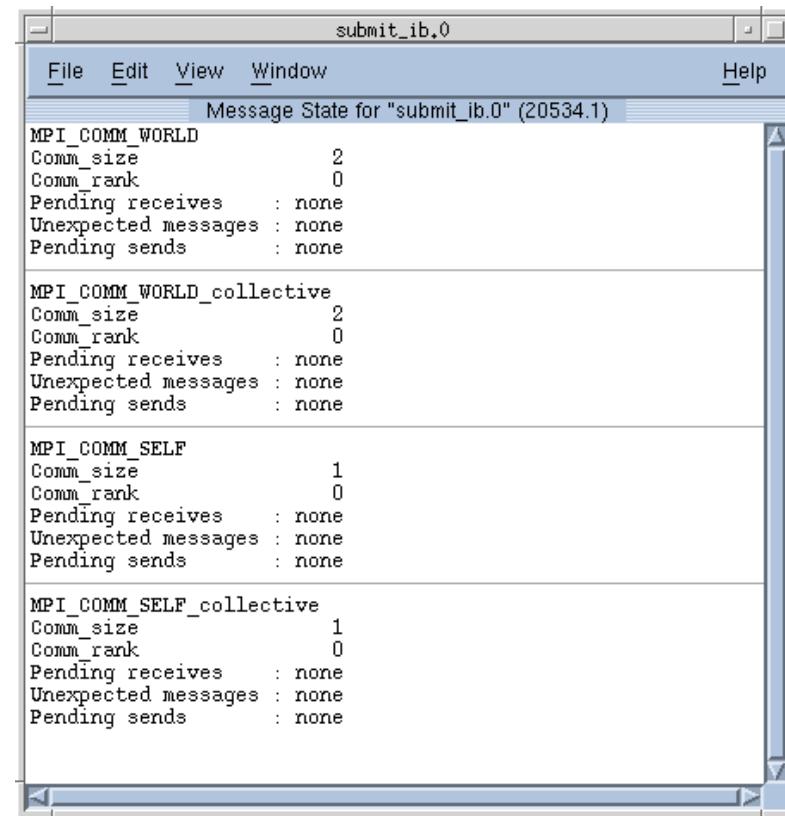
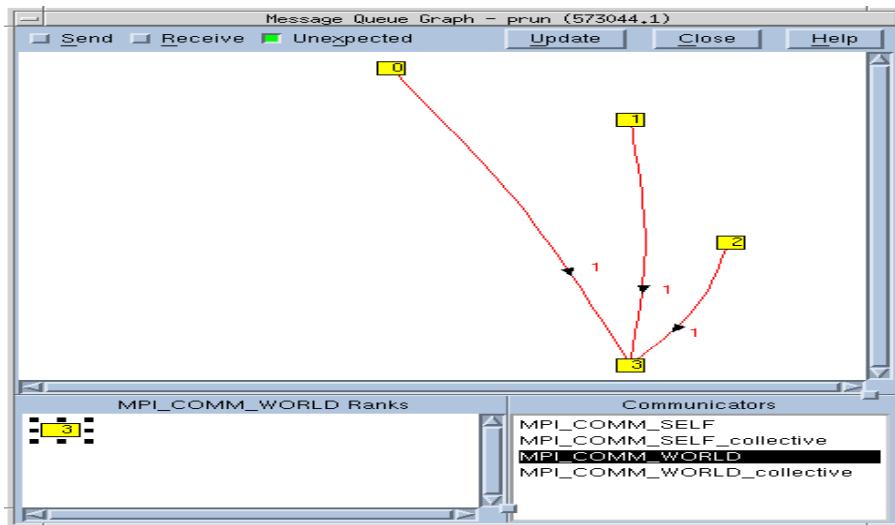
- Variable view
 - dive (right mouse) on any variable
 - change data type
 - select an array slice, e.g. (3:3,:)
 - filter array values, e.g. .ne. 0
- Variable visualization
- menu Visualize – only up to 2D arrays



- TV automatically attaches all threads
- put breakpoint to OpenMP parallel section to debug threads
- variable lamination - show values from all threads in one window – does not always work
- barrier points – shift-left click
- ambiguous action points – select all

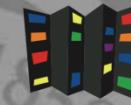


- Process synchronization – program groups
- Barrier points
- Message queue state graph and display

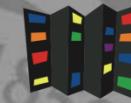


Message State for "submit_ib.0" (20534.1)

Communicator	Comm_size	Comm_rank	Pending receives	Unexpected messages	Pending sends
MPI_COMM_WORLD	2	0	: none	: none	: none
MPI_COMM_WORLD_collective	2	0	: none	: none	: none
MPI_COMM_SELF	1	0	: none	: none	: none
MPI_COMM_SELF_collective	1	0	: none	: none	: none



- Dynamic memory debugging tool
- display memory status
- paint allocated and deallocated blocks
- find memory leaks
- identify dangling pointers
- enable with **Tools > Memory Debugger**
- > **Enable memory debugging checkbox**



- Allows to reversely debug the code
- Must be turned on at the start of debugging session
- Run to the error, then backtrack to the source of the problem
- Helps to capture race conditions and other hard to reproduce bugs



Accelerator debugging

- Nvidia CUDA or OpenACC on GPU
- Intel Xeon Phi
- I haven't used it myself but some of our users did
- If you want to use it, let us know



- Totalview webpage

<http://www.roguewave.com/products-services/totalview>

- Location of Totalview

Arches:/uufs/chpc.utah.edu/sys/installdir/totalview

Some group desktops: inquire at CHPC

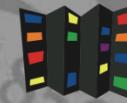
- Documentation

<http://www.roguewave.com/help-support/documentation/totalview>

<http://www.chpc.utah.edu/software/docs/par-devel.html>

<http://www.chpc.utah.edu/software/docs/totalview.html>

http://www.chpc.utah.edu/short_courses/Totalview

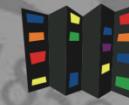


- Free for students
- Limited to one computer, 4 processes
- To sign up, e-mail m.cuma@utah.edu:
 - name
 - e-mail
 - university ID
 - anticipated year of graduation



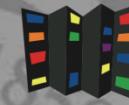
Code checkers

- compilers check for syntax errors
 - some compiler flags help too (-C)
- memory checking tools - many errors are due to bad memory management
 - valgrind – easy to use
 - purify – harder to use



- We have a 2 concurrent user license
- Tools for all stages of development
 - Compilers and libraries
 - Verification tools
 - Profilers
- More info

<https://software.intel.com/en-us/intel-parallel-studio-xe>



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

<http://software.intel.com/en-us/intel-inspector-xe/>



Intel Inspector

- Source the environment

```
module load inspectorxe
```

- Compile with -tcheck -g

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

- Run tcheck

inspxe-gui – graphical user interface

inspxe-cl – command line

- Tutorial

<https://software.intel.com/en-us/articles/inspectorxe-tutorials>

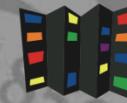


- MPI profiler and correctness checker
- Detects violations of MPI standard and errors in execution environment
- To use correctness checker

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

- ITAC documentation

<https://software.intel.com/en-us/intel-trace-analyzer-support/documentation>



Conclusions

- Terminal debuggers
- Compiler vendor debuggers
- Totalview for graphical debugging
- Code checkers and memory checkers
- InspectorXE for thread and memory debugging
- ITAC MPI checker