# Introduction to debugging

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

# Overview

- Program errors
- Simple debugging
- Graphical debugging
- Totalview
- Intel tools
- Please fill survey
  https://www.surveymonkey.com/r/7B5FJRM

# 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/printf

- 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

# GUI debuggers

- 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`**

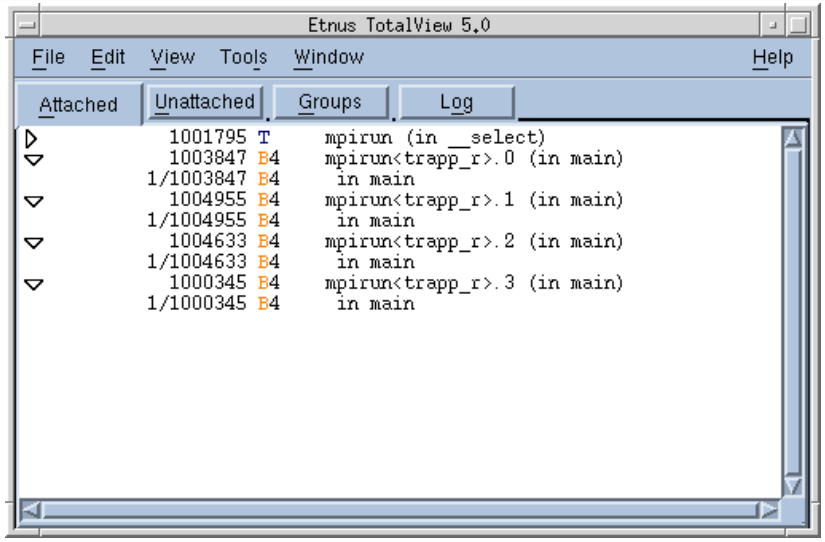
# How to use Totalview

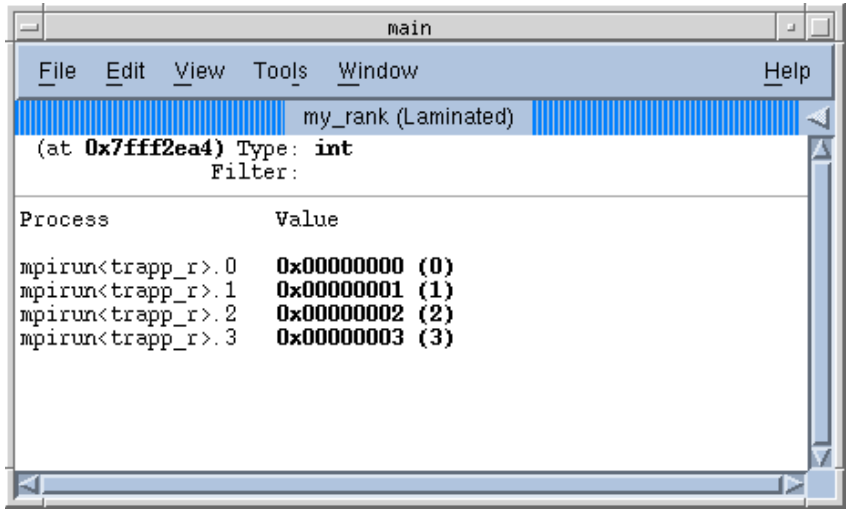
- run TV and attach the executable
  - start TV
  - Start a Debugging Session window
  - choose an existing program or define a new one
- run TV and attach running program
  - start TV
  - pick “A running program (attach)”
  - choose process ID and executable file name

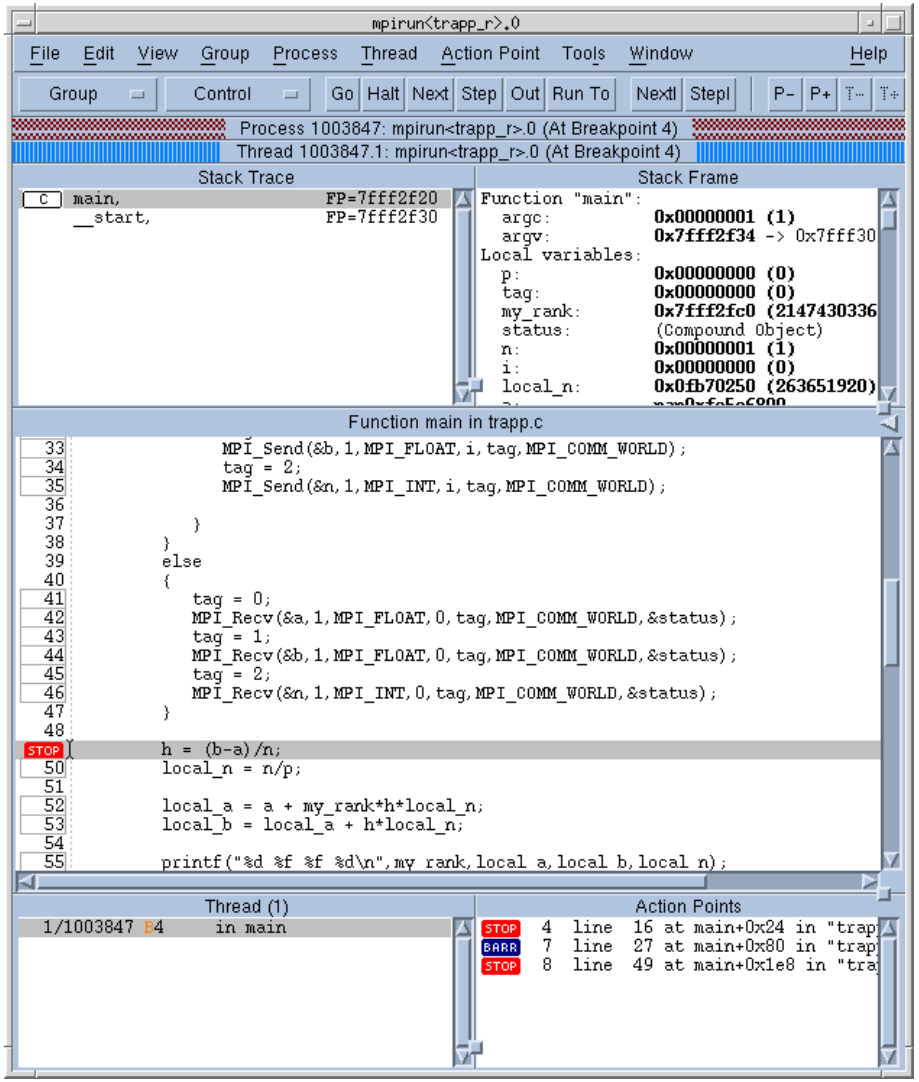
3. Totalview operation

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

# Totalview windows



# Totalview basic operations

- 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

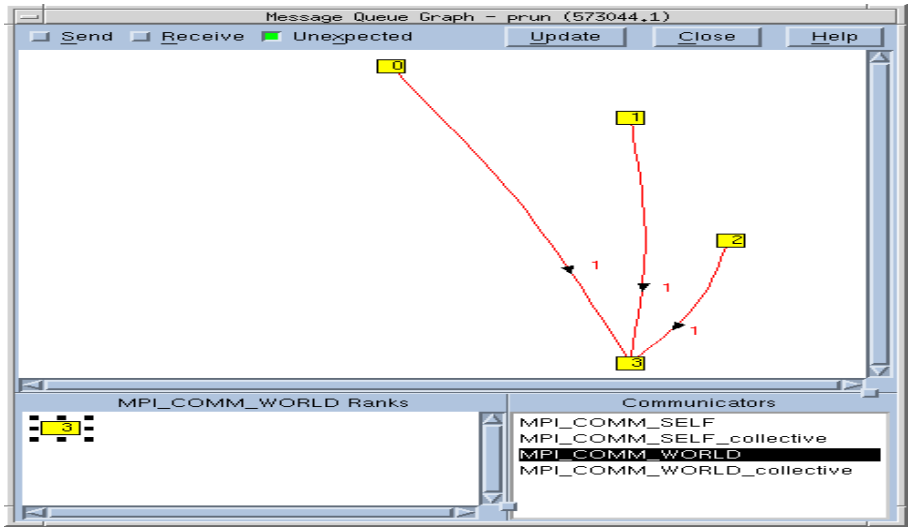
- Variable view
  - dive (right mouse) on any variable
  - change data type
  - select an array slice, e.g. (3:3,:)
  - filter array values, e.g. .ne. 0
- Variable visualization
  - menu Visualize – only up to 2D arrays

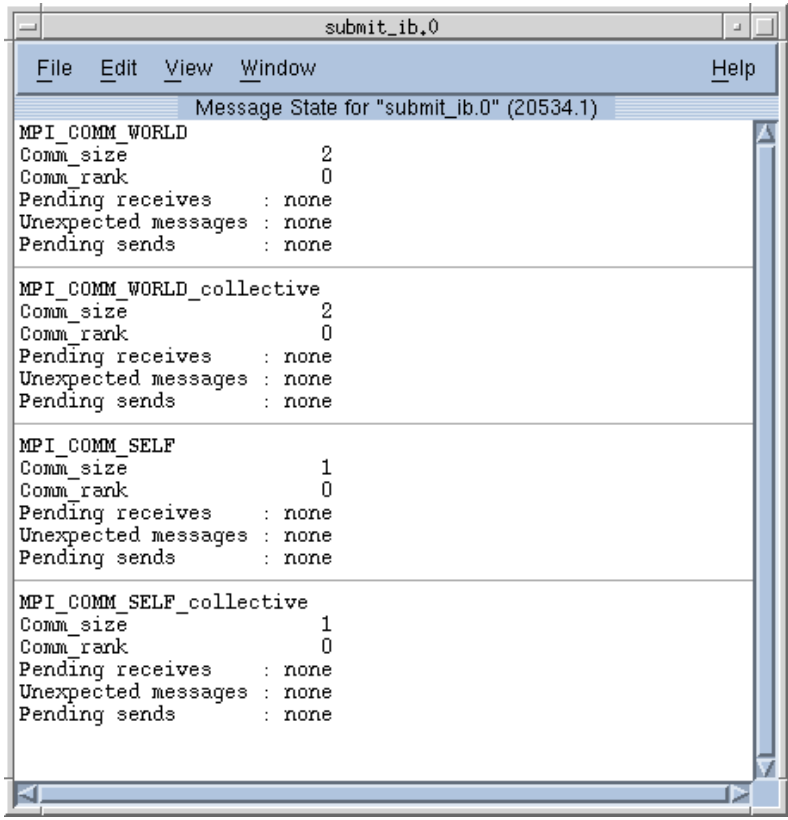
# OpenMP specific debugging

- 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

# MPI specific debugging

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



# MemoryScape

- 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

# Replay Engine

- 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
- Tried OpenACC, CUDA
  - New process window opens for the GPU code only, need to switch to the CPU process window(s) to see what's happening on the CPU(s)

# Some useful resources

- Totalview webpage

  **http://www.roguewave.com/products-services/totalview**

- Setting up Totalview

  Clusters: **module load 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**

# Totalview Student Edition

- 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

# Intel software development products

- 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

# Intel Inspector

- 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`

# Intel Trace Analyzer and Collector

- 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
- Please fill survey: https://www.surveymonkey.com/r/7B5FJRM