Debugging with **gdb** and **valgrind**

Dr. Axel Kohlmeyer

Associate Dean for Scientific Computing, CST
Associate Director, Institute for Computational Science
Assistant Vice President for High-Performance Computing

Temple University
Philadelphia PA, USA

a.kohlmeyer@temple.edu

TEMPLE UNIVERSITY®

What is Debugging?

- Identifying the cause of an error and correcting it
- Once you have identified defects, you need to:
  - find and understand the cause
  - remove the defect from your code
- Statistics show about 60% of bug fixes are wrong:
  -> they remove the symptom, but not the cause
- Improve productivity by getting it right the first time
- A lot of programmers don't know how to debug!
- Debugging needs practice and experience:
  -> understand the science and the tools

More About Debugging

- Debugging is a last resort:
  - Doesn't add functionality
  - Doesn't improve the science
- The best debugging is to avoid bugs:
  - Good program design
  - Follow good programming practices
  - Always consider maintainability and readability of code over getting results a bit faster
  - Maximize modularity and code reuse

Errors are Opportunities

- Learn from the program you're working on:
  - Errors mean you didn't understand the program.
    If you knew it better, it wouldn't have an error.
    You would have fixed it already
- Learn about the kinds of mistakes you make:
  - If you wrote the program, you inserted the error
  - Once you find a mistake, ask yourself:
    - Why did you make it?
    - How could you have found it more quickly?
    - How could you have prevented it?
    - Are there other similar mistakes in the code?

How to NOT do Debugging

- Find the error by guessing
- Change things randomly until it works (again)
- Don't keep track of what you changed
- Don't make a backup of the original
- Fix the error with the most obvious fix
- If wrong code gives the correct result,
  and changing it doesn't work, don't correct it.
- If the error is gone, the problem is solved.
  Trying to understand the problem, is a waste of time

The Physics of Bugs

- Heisenbug: bug disappears when debugging a problem (compiling with -g or adding prints) => often a bug exposed by compiler optimizations or a possible bug in the compiler
- Schroedingbug: bug only shows up after you found out that the code could not have worked at all in the first place => program design error
- Mandelbug: bug whose causes are too complex to be reliably reproduced; it thus defies repair
- Bohrbug: “regular”, straightforward to solve bug

Debugging Tools

- Source code comparison and management tools: diff, vimdiff, emacs/ediff, cvs/svn/git
  - Help you to find differences, origins of changes
- Source code analysis tools: compiler warnings, ftnchek, lint
  - Help you to find problematic code
    -> Always enable warnings when programming
    -> Always take warnings seriously (but not all)
    -> Always compile/test on multiple platforms
  - Bounds checking allows checking of (static) memory allocation violations (no malloc)

More Debugging Tools

- Using different compilers (Intel, GCC, Clang, ...)
- Debuggers and debugger frontends:
  gdb (GNU compilers), **idb** (Intel compilers),
  ddd (GUI), **eclipse** (IDE), and many more...
- **gprof** (profiler) as it can generate call graphs
- **valgrind**, an instrumentation framework
  - Memcheck: detects memory management problems
  - Cachegrind: cache profiler, detects cache misses
  - Callgrind: call graph creation tool

Purpose of a Debugger

- More information than print statements
- Allows to stop/start/single step execution
- Look at data and modify it
- *'Post mortem'* analysis from core dumps
- Prove / disprove hypotheses
- No substitute for good thinking
- But, sometimes good thinking is not a substitute for effectively using a debugger!
- Easier to use with modular code

Using a Debugger

- When compiling use -g option to include debug info in object (.o) and executable
- 1:1 mapping of execution and source code only when optimization is turned off
  -> problem when optimization uncovers bug
- GNU compilers allow -g with optimization
  -> not always correct line numbers
  -> variables/code can be 'optimized away'
  -> progress confusing with loop unrolling
- **`strip`** command removes debug info

Using **gdb** as a Debugger

- **`gdb ex01-c`** launches debugger, loads binary, stops with **`(gdb)`** prompt waiting for input:
- **`run`** starts executable, arguments are passed
- Running program can be interrupted (ctrl-c)
- **`gdb ./prog --args arg1 -flag`** passes all arguments to the run command inside gdb
- **`continue`** continues stopped program
- **`finish`** continues until the end of a subroutine
- **`step`** single steps through program line by line
- **`next`** single steps but doesn't step into subroutines

More Basic **`gdb`** Commands

- **`print`** displays contents of a known data object
- **`display`** is like print but shows updates every step
- **`where`** shows stack trace (of function calls)
- **`up down`** allows to move up/down on the stack
- **`break`** sets break point (unconditional stop), location indicated by file name+line no. or function
- **`watch`** sets a conditional break point (breaks when an expression changes, e.g. a variable)
- **`delete`** removes display or break points

Post Mortem Analysis

- Enable core dumps: **`ulimit -c unlimited`**
- Run executable until it crashes; will generate a file **`core`** or **`core.<pid>`** with memory image
- Load executable and core dump into debugger **`gdb myexe core.<pid>`**
- Inspect location of crash through commands: **`where, up, down, list`**
- Use **`directory`** to point to location of sources

Using **`valgrind`**

- Run **`valgrind ./myprog`** to instrument and run
- **`--leak-check=full --track-origins=yes`**
- Output will list individual errors and summary
- With debug info present can resolve problems to line of code, otherwise to name of function
- Also monitors memory allocation / deallocation to flag memory leaks (“forgotten” allocations)
- Instrumentation slows down execution **a lot**
- Can produce “false positives” (flag non-errors)

How to Report a Bug(?) to Others

- Research whether bug is known/fixed
  -> web search, mailing list archive, bugzilla
- Provide description on how to reproduce the problem. Find a minimal input to show bug.
- Always state hardware/software you are using (distribution, compilers, code version)
- Demonstrate, that you have invested effort
- Make it easy for others to help you!