Sequential and Parallel Debugging

Jonathan Schmalfuß

Chair of Scientific Computing
University of Bayreuth

March 14, 2025

Sequential debugging

> “Debugging is twice as hard as writing the code in the first place. Therefore, if you write the code as cleverly as possible, you are, by definition, not smart enough to debug it.” - Brian Kernighan, professor at Princeton University.

Recommended Flags by Jason Turner

GNU command line debugger: gdb

Live session inspired by hackingcpp: Debugging With gdb

GNU command line debugger: gdb

key / command	short form	meaning
`<Enter>`		repeat previous command
`<Tab>`		complete command or function name
`run [<arg>...]`	`r [<arg>...]`	run program (with command line argument(s))
`break <loc>`	`b <loc>`	set breakpoint at beginning of function or at a specific line
`step`	`s`	next instruction, step into function
`next`	`n`	next instruction, step over function
`jump <loc>`	`j <loc>`	jump to location (useful for exiting long/endless loops)
`continue`	`c`	continue until next breakpoint or end of program
`until <loc>`	`u <loc>`	continue until location (function /line)
`finish`	`fin`	finish (step out of) current function
`print <expression>`	`p`	print value of expression, e.g., variable
`info breakpoints`	`i b`	list all breakpoints
`info locals`	`i locals`	list local variables and their values
`backtrace`	`bt`	show call stack

Table: Useful gdb Commands

Helpful compilation flags: g++ compiler

Warning flags

- `-Wall`: Enables many warning flags
- `-Werror`: Converts any warning into an error
- `-Wextra`/`-W`: Additional warnings not in `-Wall`
- `-pedantic`/`-Wpedantic`: ISO standard compliance warnings
- Specific warnings:
  - `-Wconversion`, `-Wcast-align`, `-Wunused`, `-Wshadow`, `-Wold-style-cast`
  - `-Wpointer-arith`, `-Wcast-qual`, `-Wmissing-prototypes`, `-Wno-missing-braces`

CPP/C++ Compiler Flags and Options

Valgrind's tool suite

- Memcheck: detects memory-management problems:
  `valgrind --leak-check=full --show-leak-kinds=all --track-origins=yes --verbose <myexecutable>`
- Helgrind: thread debugger
  `valgrind --tool=helgrind <myexecutable>`
- Callgrind: cache profiler plus extra information about callgraphs

Sanitizers

Sanitizers

- AddressSanitizer: `-fsanitize=address`
- UndefinedBehaviorSanitizer: `-fsanitize=undefined`
- ThreadSanitizer: `-fsanitize=thread`

Honorable Mentions: Static analyzers / Linters

- cppcheck
- Clang-Tidy

GCC compiler flags, search for fsanitize

Time travel debugging

Idea: run gdb; stop at the point where problem occurs; walk backwards figure out what happend

$\Rightarrow$ theoretically supported by gdb via commands such as

`reverse-next`

- Problem: avx instruction $\Rightarrow$ unusable most of the time
- Solution: `rr`[a]: record a failure once, then debug the recording, deterministically, as many times as you want

[a] https://rr-project.org/

Parallel debugging

"Sequential programming is really hard, and parallel programming is a step beyond that." - Andrew S. Tanenbaum, professor at Vrije Universiteit Amsterdam

"Debugging is twice as hard as writing the code in the first place. Therefore, if you write the code as cleverly as possible, you are, by definition, not smart enough to debug it." - Brian Kernighan, professor at Princeton University.

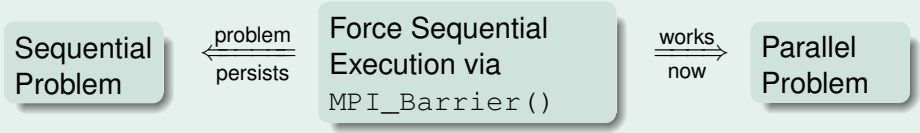
Techniques in general

> “Most bugs also appear in the sequential version of the code” - me

Techniques in general

"Most bugs also appear in the sequential version of the code" - me

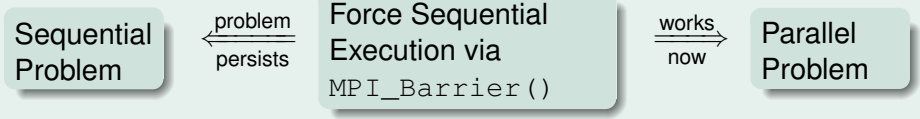
Distinguish Problem



Techniques in general

"Most bugs also appear in the sequential version of the code" - me

Distinguish Problem



The experts opinion: Anthony Williams - author/ coauthorof the thread library in C++

1. Reviewing code to locate potential bugs
2. Locating concurrency-related bugs by testing / Designing for testability

Questions to think about when reviewing multithreaded code

- Which data needs to be protected from concurrent access?
- How do you ensure that the data is protected?
- Where in the code could other threads be at this time?
- Which mutexes does this thread hold?
- Which mutexes might other threads hold?
- Are there any ordering requirements between the operations done in this thread and those done in another? How are those requirements enforced?
- Is the data loaded by this thread still valid? Could it have been modified by other threads?
- If you assume that another thread could be modifying the data, what would that mean and how could you ensure that this never happens?

Debug Deadlocks: Attach to the process [Live-session]

Situation: you have a deadlock, i.e. your executable is stuck

1. Compile with debug flags: `mpic++ -g -Wall <file> -o <name>`
2. Wait until stuck
3. Figure out process id's via `top` or `ps -a | grep <name>`
4. Attach to the process and see where you are stuck –> figure out what the problem is

MPI debugging

Debug Deadlocks: Attach to the process [Live-session]

Situation: you have a deadlock, i.e. your executable is stuck

1. Compile with debug flags: `mpic++ -g -Wall <file> -o <name>`
2. Wait until stuck
3. Figure out process id's via `top` or `ps -a | grep <name>`
4. Attach to the process and see where you are stuck –> figure out what the problem is

Attaching debugger to serval instances of the executable

- Use mpirun to launch separate instances of serial debuggers
- Drawback: many process, usually problematic

OpenMPI: FAQ: Debugging applications in parallel

MPI debugging: Memchecker

- Requires: Open MPI 1.3 or later, and Valgrind 3.2.0 or later
- Otherwise: works, but with many false positives
- Needs to be enable at compilation state of OpenMPI, unfortunately often is not
- to enable locally, see How can I use Memchecker
- `mpirun -np 2 valgrind <executable name>`

Exercises

Brought you some programs, which you can check out. Find the errors.

1. Use pure gdb `sequential_prefix_sum.cpp`
2. Use valgrind memchecker `memcheck_example.cpp`
3. Use address sanitizer `adress_sanitizer_problem.cpp`
4. Use undefined behavior sanitizer for `rounding_coordinates.cpp`
5. Is it a sequential problem or parallel for `mpi_reusing_a_buffer.cpp`