GWDG
AG Computing
Lars Quentin

Exercise 1 / 02.04.2026
Practical Course High-Performance Computing / SoSe 2026
40 Minutes Total

Debugging with GDB, Valgrind, and Sanitiziers

Learning Objectives

The learning objectives in this tutorial are:

- Get used to debugging with gdb and gdb-dashboard
- Get some experience on using the sanitizers and valgrind on the cluster to be equipped for your own debugging

Contents

Debugging with GDB 1: Tutorial (10 min) 1

Setting up gdb-dashboard 2: Tutorial (10 min) 2

Debugging memory issues with Valgrind 3: Tutorial (5 min) 2

Debugging memory issues with Sanitizers 4: Tutorial (5 min) 3

Evaluate differences between Valgrind and Sanitizers 5: Tutorial (10 min) 3

(Advanced) Build your own zero-cost print macros 6: Tutorial (N min) 3

Debugging with GDB 1: Tutorial (10 min)

Use GDB to debug `gdb_me.c` as shown in the live demo.

Steps

1. Start with a clean shell (for example, by reconnecting)
2. Download all files onto the cluster
3. Load in a modern gcc: `module load gcc`
4. Compile `gdb_me.c` with at least no optimization and maximal gdb-optimiezd debug symbols:
   `gcc -O0 -ggdb3 gdb_me.c -o gdb_me`
5. Run the code, notice that it fails
6. Load it into gdb, if you want in TUI mode:
   `gdb -tui ./gdb_me`
7. `run` it, see where it fails
   - Pro tip: if the program output screwed your screen, force a redraw by pressing CTRL-L

8. Evaluate the call stack with `bt` or `bt full`

Afterwards, play around as you see fit. Maybe create a break point, and then step through the program yourself. See the GDB slides for a reference.

Setting up gdb-dashboard 2: Tutorial (10 min)

In order to make your debugging experience easier, try our if you prefer raw gdb or a more noob friendly interface such as gdb-dashboard.

For more reference, you can also see its website: https://github.com/cyrus-and/gdb-dashboard.

This is how you set it up (with syntax highlighting):

Steps

1. Start with a clean shell (for example, by reconnecting)
2. Load in both gcc and python modules:
   `module load gcc; module load python`
3. If you want syntax highlighting: Create a virtual python environment and activate it; then install pygments in it.
   ```
   python3 -m venv gdb-venv
   source ./gdb-venv/bin/activate
   pip install pygments
   ```
   - Note: whenever you want to now use gdb-dashboard with syntax highlighting enabled, you have to load the modules and source the venv first
4. Next, try to load in as done in the first module. After run, you should see the dashboard
5. install gdb-dashboard as described in the README

You can always explicily call the dashboard when using the `dashboard` command.

Debugging memory issues with Valgrind 3: Tutorial (5 min)

Try out valgrind on an incorrect program.

Steps

1. Start with a clean shell (for example, by reconnecting)
2. Look at `demo.c` and try to find its mistakes
3. Load in valgrind via `module load valgrind`
4. Compile with `-O0 -g3` as above and load the binary into valgrind

Afterwards, you should see the result errors

Debugging memory issues with Sanitizers 4: Tutorial (5 min)

Now, evaluate the same program using the sanitizers.

- Start with a clean shell (for example, by reconnecting)
- Compile `demo.c` with at least ASan+UBSan, see the slides for more information.
- Run the binary. If you compiled it correctly, you should see the errors.

Compare the output between Valgrind and the sanitizers. What do you notice?

Evaluate differences between Valgrind and Sanitizers 5: Tutorial (10 min)

Try out both valgrind and sanitizers on `does_it_work.c` and `does_it_work2.c`.
What do you notice?

(Advanced) Build your own zero-cost print macros 6: Tutorial (N min)

! If you plan to use C or C++ for your final PCHPC project, it is very useful to already have some zero-cost debug macros. In `debug_prints.hh` I provided the same macros as shown on the slides. Feel free to take the remaining time to implement or adapt the macros to your favorite use case, such as graph-, vector-, or map-specific debugging printers.