Parallel Debugging

Jonathan Schmalfuß

Chair of Scientific Computing
University of Bayreuth

March 20, 2025

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

Sequential Problem $\xleftarrow[\text{persists}]{\text{problem}}$ Force Sequential Execution via `MPI_Barrier()` $\xrightarrow[\text{now}]{\text{works}}$ Parallel Problem

Techniques in general

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

Distinguish Problem

Sequential Problem $\xleftarrow[\text{persists}]{\text{problem}}$ Force Sequential Execution via `MPI_Barrier()` $\xrightarrow[\text{now}]{\text{works}}$ Parallel 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 while reviewing multiprocess code

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

Parallel Debugging: How to?

the easy way

- use a debugger and or fronted made for debugging MPI code
- Industry Standard: ddt[1], TotalView[1]

should-always-work way

- mininmal requirements: a debugger (gdb) + a way of finding the running processes (ps/top)
- `mpirun` creates multiple processes –> attach to relevant processes or all –> debugg each of them sequentially
- limits: low number of processes, requires duplicating input for each process

Compromise? Open Source Projects / Free:

command line tool with plotting ability mdb, intel oneAPI with mpigdb or a shell script tmpi

[1] temporary free / student license available

MPI debugging - should-always-work

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

Recap Live-Session: MPI debugging - should-always-work

```
$ mpic++ deadlock_blocking_recieve_before_send.cpp -g -Wall
$ mpirun -np 2 ./a.out 10
0: Receiving 10 elements of type int from my left neighbor 1.
1: Receiving 10 elements of type int from my left neighbor 0.
```

```
$ ps -a | grep a.out
553640 pts/1    00:00:15 a.out
$ ps -a | grep a.out
553980 pts/1    00:00:02 a.out
553981 pts/1    00:00:02 a.out
$ gdb -p 553980
GNU gdb (Ubuntu 12.1-0ubuntu1~22.04.2) 12.1
...
(gdb) where
#0  in opal_progress () from ...
#1  in mca_pml_ob1_recv () from ...
#2  in PMPI_Recv () from ...
#3  in main () at deadlock_blocking_recieve_before_send.cpp:28
```

MPI debugging - multiple instances

Attaching debugger to serval instances of the executable [Live-session]

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

OpenMPI: FAQ: Debugging applications in parallel

- Attach a terminal with gdb to each process (opens multiple windows using xterm)

```
mpiexec -n 2 xterm -e gdb --args a.out 10
```

- Attach to single process from the beginning with arguments:

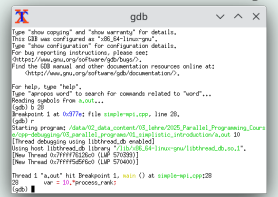
```
mpirun -n 1 gdb --args ./a.out 10 : -n 1 ./a.out 10
```

- Attach to single process with xterm from the beginning with arguments:

```
mpirun -n 1 xterm -e gdb --args ./a.out 10 : -n 1 ./a.out 10
```

Recap Live-Session: MPI debugging - multiple instances

```
$ mpicxx -g simple-mpi.cpp
$ mpirun -n 2 xterm -e gdb --args a.out 10
```




```
$ mpicxx -g simple-mpi.cpp
$ mpirun -n 1 gdb --args ./a.out 10 : -N 1 ./a.out 10
GNU gdb
(gdb) b 28
Starting program: /data/01_simplistic_introduction/a.out 10
[New Thread 0x7ffff76126c0 (LWP 572127)]
[New Thread 0x7ffff5d5f6c0 (LWP 572128)]

Thread 1 "a.out" hit Breakpoint 1, main () at simple-mpi.cpp:28
28        var = 10.*process_rank;
```

MPI debugging - mgdb

mgdb [Live-session]

- easy installable :

```
python3 -m venv .mdb
source .mdb/bin/activate
pip install mdb-debugger[termgraph]
```

- potentially powerful
- drawback: is command line only / early development stages
- QuickStart

Recap Live-Session: MPI debugging - mgdb

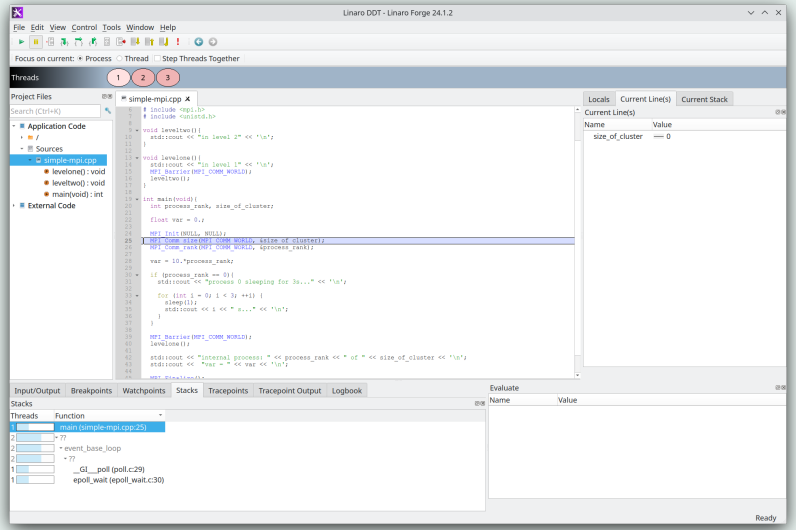
```
$ python3 -m venv .mdb
$ source .mdb/bin/activate
$ pip install mdb-debugger[termgraph]
...
$ mpicxx -g simple-mpi.cpp
$ mdb launch -b gdb -n 2 -t ./a.out --log-level=DEBUG
running on host: 132.180.176.41
to connect to the debugger run:
mdb attach -h 132.180.176.41 -p 2000
...
```

```
$ source .mdb/bin/activate
$ mdb attach -h 132.180.176.41 -p 2000
(mdb 0-1) command 0 b 38
0:      Breakpoint 2: file simple-mpi.cpp, line 39.
(mdb 0-1) broadcast start
(bcm 0-1) c
^C0:    Continuing.
0:      process 0 sleeping for 3s...
0:      Thread 1 "a.out" hit Breakpoint 2, main () at simple-mpi.cpp:39
0:      39        MPI_Barrier(MPI_COMM_WORLD);
```

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
↪  --suppressions=$PREFIX/share/openmpi/openmpi-valgrind.supp
↪  <executable name>
```

The easy way



ddt tutorial

Now what?

Use the tools / test them on simple and more complex examples

In the corresponding github project work through the folder `03_parallel_programs` content. The available tools on the PC-Pool workstations are gdb / xterm.

Possible tasks:

- Attach gdb in different ways to the same executable!
- Is it a sequential or parallel problem?
- Last but not least: Ask Questions, not only to me but also to each other about your understanding.