

Introduction to Debugging on HPC

Holger Obermaier | 22. June 2022



Debugging

1. GNU Compiler Collection

- Static Analyzer
- Sanitizer

2. Strace

3. Ltrace

4. Valgrind

5. GDB (The GNU Project Debugger)

6. ARM Forge DDT (Distributed Debugging Tool)

Debuggin ...

- Software contains errors and bugs!
- Debugging: Process of finding and fixing these bugs
- Debugging steps
 - Recognize that a bug exists
 - Determine the origin of the bug
 - Find a fix
 - Apply and test the fix
- Complex task \Rightarrow Tools can assist with this
- Special attention should be paid to avoiding bugs (e.g. unit tests, documentation, ...)
- See: WikiBook: Computer Programming Principles [↗](#)

GNU Compiler Collection: Static Analyzer

- Static-analysis of source code during compile time
 - ⇒ A kind of extended warnings
 - ⇒ No need to run the binary
- Potentially false negatives
- Problems that can be detected
 - double memory free
 - double file close
 - use of memory after free
 - use of uninitialized values
 - ...
- GCC Static Analyzer Options [↗](#)
- Basic usage

```
gcc -fanalyzer ${SOURCE}
```

Example

- Example: Static Analyzer - Use after free [↗](#)

GNU Compiler Collection: Sanitizer

- Compiler adds run-time instrumentation
 - Analysis during runtime
 - Problems that can be detected
 - out of bounds memory access
 - use of uninitialized values
 - undefined behaviour
 - use of memory after free
 - thread data race conditions
 - ...
 - GCC Program Instrumentation Options [↗](#)
 - Basic usage
- ```
gcc -fsanitize=${SANITIZER_TYPE} ${SOURCE}
```

## Example

- Example: Sanitizer - Use after free [↗](#)
- Example: Sanitizer - Division by zero [↗](#)
- Example: Sanitizer - Data race [↗](#)

# Strace

- Linux syscall tracer
  - ⇒ Monitor interactions between processes and the Linux kernel
- Dynamic-analysis during runtime
- Potentially large overhead
- [strace.io](#)
- [Man page strace](#)
- Basic usage



```
module add devel/strace
strace ${BINARY} # trace binary
strace -p ${PID} # trace already running process
```



## Example


- Example: [strace - Basic usage and MPI usage scenarios](#)

# Ltrace

- Intercepts and records dynamic library calls
- Dynamic-analysis during runtime
- Potentially large overhead
- [ltrace.org](https://ltrace.org) 
- Man page [ltrace](#) 
- Basic usage

```
ltrace ${BINARY} # trace binary
ltrace -p ${PID} # trace already running process
```

## Example

- Example: [ltrace - Basic usage and MPI usage scenarios](#) 

# Valgrind

- Tool to detect
  - memory management bugs
  - threading bugs
  - profile your programs
- Prepare unoptimized binary with debug symbols during compile time
- Dynamic-analysis during runtime
- Binary is executed in virtual machine with JIT
  - ⇒ Massive increase in runtime and memory consumption
- Problems that can be detected by *memcheck*
  - use of memory after free
  - use of uninitialized values
  - out of bounds memory access
  - memory leaks
- Potentially many false positives ⇒ suppress mechanisms





# Valgrind ...

- [valgrind.org](http://valgrind.org)
- [Valgrind User Manual](#)
- Basic usage

```
module add \
 compiler/gnu \
 devel/valgrind
gcc -O1 -g ${SRC} -o ${BINARY}
valgrind ${BINARY}
```

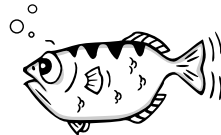
## Example

- [Example: Valgrind - Use after free](#)
- [Example: Valgrind - MPI usage scenarios](#)

# GDB (The GNU Project Debugger)

- Inspect a program during execution
- Pause the program (at breakpoints or on conditions)
- Read values from memory or stack
- Change values in memory or stack
- GDB: The GNU Project Debugger [↗](#)
- GDB User Manual [↗](#), Man page GDB [↗](#)
- Basic usage

```
module add compiler/gnu
gcc -O1 -g ${SRC} -o ${BINARY}
gdb ${BINARY}
```





# GDB (The GNU Project Debugger) ...

## Most frequently used GDB commands

`break <func>` Set a breakpoint at `<func>`  
    `run` Run program  
    `bt` Backtrace (show call stack)  
    `next` Execute next program line. Do not step into function calls  
    `step` Execute next program line. Step into function calls  
    `c` Continue running program (until next break point)  
`print <expr>` Print value of `<expr>`

## Example

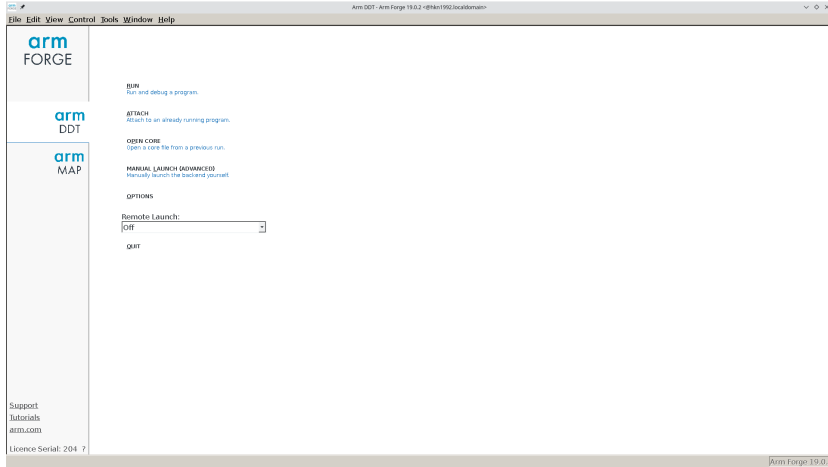
- Example: GDB - Division by zero 
- Example: GDB - Remote debugging 

# ARM Forge DDT (Distributed Debugging Tool)

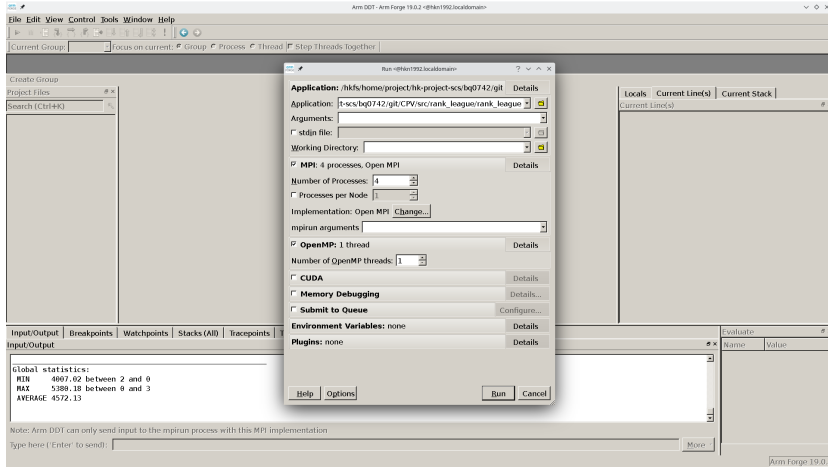
- Graphical debugger
- Supports OpenMP, MPI and GPUs
- Commercial license available on HoreKa
- Arm DDT [↗](#)
- Get started [↗](#), User Guide [↗](#), Tutorials [↗](#)
- Basic usage

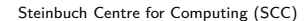
```
module add devel/ddt/
ddt
```

# ARM Forge DDT (Distributed Debugging Tool) ...



# ARM Forge DDT (Distributed Debugging Tool) ...





# ARM Forge DDT (Distributed Debugging Tool) ...

