Debuggers

2022-06-28 | Gerald Mathias

Hint: Use a debugger, always.

"I use gdb all the time, but I tend to use it not as a debugger, but as a disassembler on steroids that you can program." (Linus Torvalds, 2000)

Use-cases:

- Step through the flow of an unknown program
- Determine the most importatant code path
- Break at time consuming routines
- Acess contents of local variables, loop sizes, data
- Test new routines (you can independently execute each function out of gdb)
- Debug programs (spot the segfault, floating point exception, etc.)
- Examine a core dump

Debuggers

Textbased CPU Debuggers:

- The mothership: **GNU Debugger** (**GDB**)
  - Languages: Ada, C, C++, Objective-C, Free Pascal, Fortran, Go, …
  - Processors: Alpha, ARM, X86 , X86-64, IA-64 "Itanium", Motorola 68000, MIPS, PA-RISC, PowerPC, SPARC, …
  - Exhaustive Documentation: https://sourceware.org/gdb/current/onlinedocs/gdb/
- Others: Used to be shipped with different UNIX versions and compilers
  (hardly relevant anymore)

Graphical frontends:

- ddd: Data Display Debugger (ships with some Linux variants)
- IDEs: KDevelop, Eclipse, GNU Emacs, NetBeans

GDB invocation

```
# load module for recent gdb
:~> module load gdb

# recompile code, if necessary
:~/COW-Code/ver0>  make -B -f ../Makefile nbody-aos-gdb CFLAGS="-g -O2 -DFLAGS=aos,double"
icc -I.. -DUSE_AOS -g -O2 -DFLAGS=aos,double -I.. -fno-inline-functions -qopt-report=5 -qopt-
report-phase=vec -qopt-report-phase=openmp -qopt-report-routine=kick,drift,accel -qopt-report-
file=stdout  -DUNITY_OUTPUT_COLOR -DUNITY_INCLUDE_DOUBLE ../unity.c ../nb-aos-tests.c ../nb-
main.c ../nb-aos-support.c nb-aos-alloc.c nb-aos-kda.c -o nbody-aos-gdb > nbody-aos-gdb.optrpt

# start gdb
:~/COW-Code/ver0> gdb nbody-aos-gdb

GNU gdb (GDB) 11.1
Copyright (C) 2021 Free Software Foundation, Inc.
…
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.


For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from nbody-aos-gdb
...
(gdb)
```

GDB: Running a program

```
# list command (l)
(gdb) list
…
850 //      main
…
855 int main(int argc, char *argv[]) {

# set a breakpoint (b)
(gdb) break accel
Breakpoint 1 at 0x408ce5: file nb-aos-kda.c, line 54

# run the program
(gdb) run --run-sim=medium --steps=5 --run-perf
Starting program: /dss/dsshome1/07/lu64bag3/COW-Code/ver0/nbody-test --run-sim=medium --
steps=5 --run-perf
…
  N-Body Simulator for PRACE Code Optimization Course 2022
[… more output …]

Breakpoint 1, accel (N=4096, p=0x7ffff7f85010, rsoft=0.001) at nb-aos-kda.c:54
54 {
(gdb)
```

GDB: Navigation

```
# print the current stack of frames
(gdb) backtrace
#0  accel (N=4096, p=0x7ffff7f85010, rsoft=0.001) at nb-aos-kda.c:54
#1  0x000000000040884d in run_perf (simopts=0x1000, … ) at ../nb-aos-support.c:189
#2  0x0000000000406be2 in main (argc=4096, argv=0x7ffff7f85010) at ../nb-main.c:880

# examine variables
(gdb) p p[2].r.x
$3 = 0.91740860908345745

# define own functions:
(gdb) define vp_rv
>p (p[$arg0].r.x * p[$arg0].v.x + p[$arg0].r.y * p[$arg0].v.y + p[$arg0].r.z * p[$arg0].v.z)
>end
(gdb) vp_rv 4
$7 = 0.84119974595568703

# document your function
(gdb) document vp_rv
>calculate vector product between r and v for particle n
>usage vp_rv <n>
>end
(gdb) help vp_rv
calculate vector product between r and v for particle n
usage vp_rv <n>
```

GDB: Hooks

```
(gdb) set $i=0
(gdb) define hook-stop
>printf "particle pair %d\n", $i++
>end
(gdb) b 65
Breakpoint 5 at 0x408ef8: file nb-aos-kda.c, line 65.

(gdb) cont
Continuing.
particle pair 0

Breakpoint 2, accel (N=4096, p=0x1000, rsoft=0.001) at nb-aos-kda.c:65
65               real dx = p[i].r.x - p[j].r.x;
(gdb) cont
Continuing.
particle pair 1

Breakpoint 2, accel (N=4096, p=0x1000, rsoft=0.001) at nb-aos-kda.c:65
65               real dx = p[i].r.x - p[j].r.x;
```

Hooks have the form `hook-<cmd>` or `hookpost-<cmd>`, e.g. `hookpost-list`

for pre- and post-command execution. Unset hooks with an empty redefinition of the hook.

Summary GDB

- gdb is a feature-rich debugger
- text-based version is worthwhile to learn (at least some basics)
- get a gdb-cheat sheet for a convienient overview of commands
  (e.g. http://www.cheat-sheets.org/saved-copy/gdb-refcard-a4.pdf)
- graphical frontends may be more convenient, but will not provide all features

Advanced debuggers

Parallel debuggers to debug programms with many 1000 MPI tasks/OpenMP threads

- ARM Forge (formerly Allinea DDT)
- TotalView

Debuggers for GPU Programming:

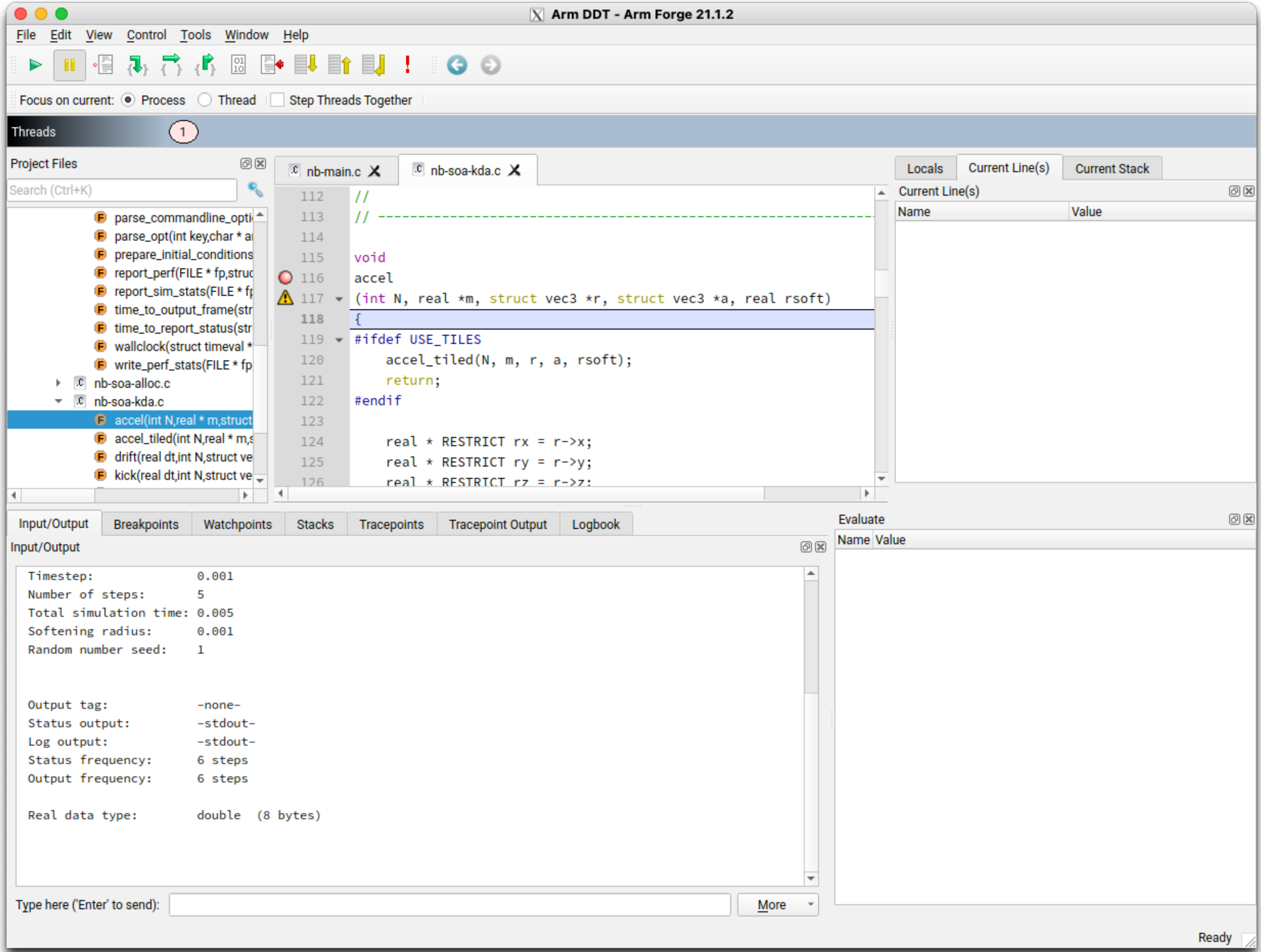
- partially supported by ARM Forge and TotalView
- NVIDIA Nsight
- CUDA-GDB
- CUDA-MEMCHECK
- AMD ROCm Debugger (ROCgdb)

Next: Example ARM Forge

DDT ARM Forge

Features

- MPI tasks
- OpenMP threads
- task/thread independent or parallel execution
- step in/over/out
- move up and down the stack
- graphically set (conditional) breakpoints
- compare variable across tasks/threads
- Memory debugging/ memory view
- multi-dimensional array viewer


Arm DDT - Arm Forge 21.1.2
File Edit View Control Tools Window Help
Threads 1
Project Files
Search (Ctrl+K)
parse_commandline_opti
parse_opt(int key,char * a
prepare_initial_conditions
report_perf(FILE * fp,struc
report_sim_stats(FILE * fp
time_to_output_frame(str
time_to_report_status(str
wallclock(struct timeval *
write_perf_stats(FILE * fp
nb-soa-alloc.c
nb-soa-kda.c
accel(int N,real * m,struct
accel_tiled(int N,real * m,s
drift(real dt,int N,struct ve
kick(real dt,int N,struct ve
nb-main.c
nb-soa-kda.c
Locals
Current Line(s)
Current Stack
Current Line(s)
Name
Value
Input/Output
Breakpoints
Watchpoints
Stacks
Tracepoints
Tracepoint Output
Logbook
Input/Output
Evaluate
Name
Value
More
Ready

Focus on current: ● Process ○ Thread □ Step Threads Together

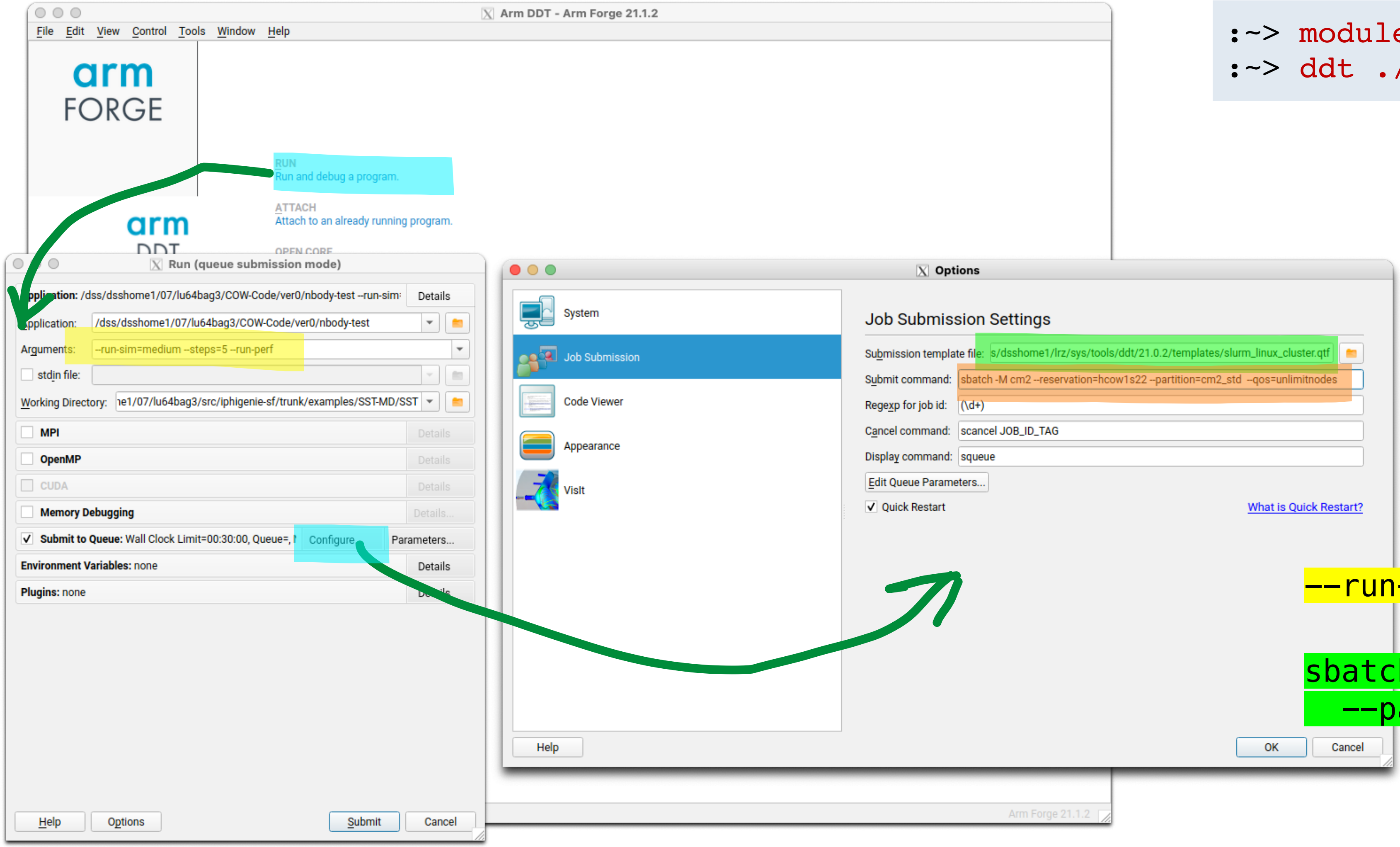
```
//
// -----------------------------------------------------------------

void
accel
(int N, real *m, struct vec3 *r, struct vec3 *a, real rsoft)
{
#ifdef USE_TILES
    accel_tiled(N, m, r, a, rsoft);
    return;
#endif

    real * RESTRICT rx = r->x;
    real * RESTRICT ry = r->y;
    real * RESTRICT rz = r->z;
```

```
Timestep:              0.001
Number of steps:       5
Total simulation time: 0.005
Softening radius:      0.001
Random number seed:    1


Output tag:            -none-
Status output:         -stdout-
Log output:            -stdout-
Status frequency:      6 steps
Output frequency:      6 steps

Real data type:        double  (8 bytes)
```

Type here ('Enter' to send):

DDT configuration to test on CoolMUC-2

```
:~> module load ddt
:~> ddt ./nbody-aos-gdb
```



Application: /dss/dsshome1/07/lu64bag3/COW-Code/ver0/nbody-test --run-sim:

Application: /dss/dsshome1/07/lu64bag3/COW-Code/ver0/nbody-test

Arguments: --run-sim=medium --steps=5 --run-perf

□ stdin file:

Working Directory: ıe1/07/lu64bag3/src/iphigenie-sf/trunk/examples/SST-MD/SST

☑ Submit to Queue: Wall Clock Limit=00:30:00, Queue=, I

Environment Variables: none

Plugins: none

Job Submission Settings

Submission template file: s/dsshome1/lrz/sys/tools/ddt/21.0.2/templates/slurm_linux_cluster.qtf

Submit command: sbatch -M cm2 --reservation=hcow1s22 --partition=cm2_std --qos=unlimitnodes

Regexp for job id: (\d+)

Cancel command: scancel JOB_ID_TAG

Display command: squeue

☑ Quick Restart

```
--run-sim=medium --steps=5 --run-perf
```

```
sbatch -M cm2 --reservation=hcow1s22
  --partition=cm2_std --qos=unlimitnodes
```

```
…/slurm_linux_cluster.qtf
```

Hands on:

Try to run either GDB on the command line or start the DDT debugger.