The Center for Astrophysical Thermonuclear Flashes

Debugging / Profiling

Chris Daley

23rd June

An Advanced Simulation & Computing (ASC)
Academic Strategic Alliances Program (ASAP) Center
at The University of Chicago

Motivation

- Bugs are inevitable (even in FLASH):
  - Compiler options can help locate bugs.
  - Good practise to customize [C|F|L]FLAGS_DEBUG in Makefile.h to use the full range of debugging options for your compiler.
  - The sites directory in the FLASH source tree contains sample Makefile.h files with good starting points.
- The infamous: “Segmentation fault (core dumped)” can be extremely hard to resolve unless you use compiler options and/or a memory debugger.
- Note: Debugging becomes much easier when binaries are compiled with debugging information (-g option).

Segmentation fault (core dumped)

- Always worth investigating the stack backtrace using gdb (or some other debugger, e.g. totalview):

```
gdb flash3 core
(gdb) bt
#0  0x0000000000490ca4 in gr_expanddomain (mype=0, numprocs=1,
      particlesinitialized=.FALSE.) at gr_expandDomain.F90:157
#1  0x0000000000432d88 in grid_initdomain (mype=0, numprocs=1, restart=.FALSE.,
      particlesinitialized=.FALSE.)
    at Grid_initDomain.F90:94
#2  0x0000000000420f02 in driver_initflash () at Driver_initFlash.F90:152
#3  0x0000000000427fc9 in flash () at Flash.F90:38
```

- Frame #0 shows the line containing the error.
  - Note: This error may itself be a symptom of an earlier memory error.

Compiler options

- The original cause of most segfaults is an array that is accessed out of bounds or accessed before being allocated:
  - Add bounds checking: -fbounds-check.

- Other important options:
  - Add a default initial value: -finit-real=nan
  - Add a check for floating point exceptions: -ffpe-trap=invalid,zero,overflow
  - Add a stacktrace print out: -fbacktrace

- Compiler options help catch the original memory violation, e.g. the seg-fault shown on previous slide was caused by:

  At line 100 of file Simulation_initBlock.F90

  Fortran runtime error: Array reference out of bounds for array 'blklimits', upper bound of dimension 1 exceeded (890 > 2)

Deadlocks

- A debugger can be attached to a running process.
  - This is very useful especially when the program deadlocks.

```
> pgrep flash3
2553
2554

> gdb flash3 2553
(gdb) bt
...
```

- No need to start the program within the debugger!

Valgrind

- A collection of programming tools including a memory debugger, cache simulator and heap memory profiler (http://valgrind.org/).
- No special compilation or linking required:
  - Raw binary runs in the valgrind CPU simulator.
  - Can detect errors in libraries (no need to have source).
- Most popular tool is the memory debugger named memcheck:
  - Detects usage of uninitialized memory.
  - Detects reads or writes beyond array bounds.
    - But only for heap allocated arrays.
  - Detects memory leaks.

Valgrind's memcheck tool

- Usage and example output:

mpirun -np N valgrind --tool=memcheck --track-origins=yes
--log-file=valgrind.log.%p ./flash3

```
==22257== Conditional jump or move depends on uninitialised value(s)
==22257==    at 0x42BF36: grid_getcellcoords_ (Grid_getCellCoords.F90:144)
==22257==    by 0x448B93: simulation_initblock_ (Simulation_initBlock.F90:136)
==22257==    by 0x490871: gr_expanddomain_ (gr_expandDomain.F90:161)
==22257==    by 0x43255B: grid_initdomain_ (Grid_initDomain.F90:94)
==22257==    by 0x42076D: driver_initflash_ (Driver_initFlash.F90:152)
==22257==    by 0x42776C: MAIN__ (Flash.F90:38)
==22257==    by 0x56E359: main (in /home/chris/Flash/Flash3_trunk/sedov_info/flash3)
==22257==  Uninitialised value was created by a stack allocation
==22257==    at 0x44843D: simulation_initblock_ (Simulation_initBlock.F90:45)
```

- A way to locate hard to find errors. But:
  - Output can be extremely verbose.
  - False-positives can happen.
  - Program usually runs an order of magnitude slower.

Profiling

- Can use FLASH timers to meaure time spent:
  - Timers_start() and Timers_stop().

- For more complete (and automated) measurement use a specialist tool, e.g. TAU (http://www.cs.uoregon.edu/research/tau/home.php):
  - Very simple to use TAU to measure time spent in the application at a finer level of granularity (e.g. subroutine and loop level) and with information about MPI calls.
  - TAU provides tools to analyse scaling performance.
  - Setup FLASH with *-tau* argument containing the name of a TAU Makefile, e.g.

./setup Sedov -auto -tau=/opt/tau-2.18.1/x86_64/lib/Makefile.tau-callpath-mpi-pdt

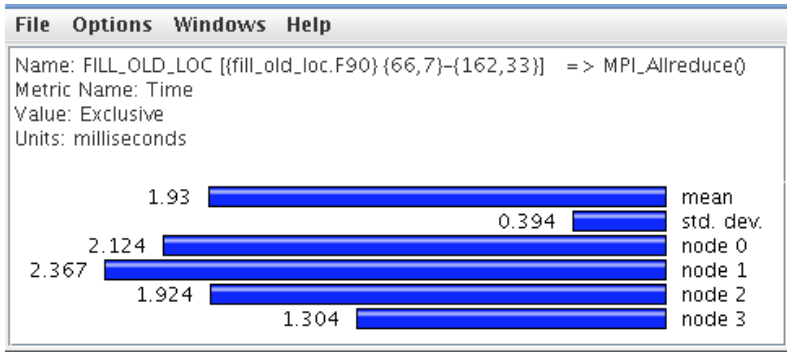
TAU

- Callpath profiling captures the time spent in a routine when it is called from each parent routine:

File Options Windows Help

Name	Exclusive Time ▽	Inclusive Time	Calls
IO_UPDATESCALARS [{IO_updateScalars.F90}{36,1}-{65,31}]	5.364	99.733	112
DRIVER_SENDOUTPUTDATA [{Driver_sendOutputData.F90}{28,1}-{53,36}]	5.303	19.753	112
GRID_MOVEPARTICLES [{Grid_moveParticles.F90}{77,1}-{124,33}]	5.129	166.196	200
GR_PTMOVEOFFBLK [{gr_ptMoveOffBlk.F90}{43,1}-{149,30}]	48.25	96.815	200
GR_PTMOVESIEVE [{gr_ptMoveSieve.F90}{79,1}-{294,29}]	2.638	61.613	200
GRID_GETLISTOFBLOCKS [{Grid_getListOfBlocks.F90}{73,1}-{178,35}]	1.383	1.383	200
GR_ENSUREVALIDNEIGHBORINFO [{gr_ensureValidNeighborInfo.F90}{64,1}-{113,41}]	1.256	1.256	200
FILL_OLD_LOC [{fill_old_loc.F90}{66,7}-{162,33}]	5.104	8.214	41
MPI_Allreduce()	1.93	1.93	41
MPI_Barrier()	1.066	1.066	41
MPI_Ssend()	0.048	0.048	1.5
MPI_Waitall()	0.047	0.047	1.5
MPI_Irecv()	0.019	0.019	1.5
GRID_UPDATEREFINEMENT [{Grid_updateRefinement.F90}{44,1}-{108,36}]	4.887	4,968.98	100
void io_h5writeLists(int *, hid_t *, int *, char (*)[80], double *, int *, char (*)[80], int *, int *, cl	4.555	5.155	14

- Can detect load balance issues by clicking on the routine name:



Custom profiling

- TAU selective instrumentation file in FLASH source tree at tools /tau/select.tau.
  - We may wish to exclude certain files from the list to reduce measurement overhead.

In select.tau edit exclude list:

```
BEGIN_EXCLUDE_LIST
INTERP
MONOT
AMR_1BLK_CC_CP_REMOTE
END_EXCLUDE_LIST
```