Debugging / Profiling

Chris Daley
23rd June
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 0x00000000000490ca4 in gr_expanddomain (mype=0, numprocs=1, particlesinitialized=.FALSE.) at gr_expandDomain.F90:157
#1 0x00000000000432d88 in grid_initdomain (mype=0, numprocs=1, restart=.FALSE., particlesinitialized=.FALSE.)
   at Grid_initDomain.F90:94
#2 0x00000000000432d88 in grid_initdomain (mype=0, numprocs=1, restart=.FALSE., particlesinitialized=.FALSE.)
   at Grid_initDomain.F90:94
#3 0x00000000000432d88 in grid_initdomain (mype=0, numprocs=1, restart=.FALSE., particlesinitialized=.FALSE.)
   at Grid_initDomain.F90:94

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.

```bash
> pgrep flash3
2553
2554

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

- No need to start the program within the debugger!
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)
  
  ==22257== 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.
  ```

The ASC/Alliances Center for Astrophysical Thermonuclear Flashes
The University of Chicago
Profiling

- Can use FLASH timers to measure 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.

    ```bash
    ./setup Sedov -auto -tau=/opt/tau-2.18.1/x86_64/lib/Makefile.tau-callpath-mpi-pdt
    ```
- Callpath profiling captures the time spent in a routine when it is called from each parent routine:

<table>
<thead>
<tr>
<th>File</th>
<th>Options</th>
<th>Windows</th>
<th>Help</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Exclusive Time</td>
<td>Inclusive Time</td>
<td>Calls</td>
</tr>
<tr>
<td>IO_UPDATESCALARS (IO_updateScalars.F90) (36,1)-(65,31)]</td>
<td>5.364</td>
<td>99.733</td>
<td>112</td>
</tr>
<tr>
<td>DRIVER_SENDOUTPUTDATA (Driver_sendOutputData.F90) (28,1)-(53,36)]</td>
<td>5.303</td>
<td>19.753</td>
<td>112</td>
</tr>
<tr>
<td>GRID_MOVEPARTICLES (Grid_moveParticles.F90) (77,1)-(124,33)]</td>
<td>5.123</td>
<td>166.196</td>
<td>200</td>
</tr>
<tr>
<td>GR_PTMOVEOFFBLK (gr_ptMoveOffblk.F90) (43,1)-(149,20)]</td>
<td>48.25</td>
<td>96.815</td>
<td>200</td>
</tr>
<tr>
<td>GR_PTMOVESLICE (gr_ptMoveSlice.F90) (79,1)-(294,29)]</td>
<td>2.634</td>
<td>61.613</td>
<td>200</td>
</tr>
<tr>
<td>GRID_GETLISTOFBLOCKS (Grid_getListOfBlocks.F90) (75,1)-(178,35)]</td>
<td>1.383</td>
<td>1.383</td>
<td>200</td>
</tr>
<tr>
<td>GR_ENSUREVALIDNEIGHBORINFO (gr.ensureValidNeighborInfo.F90) (64,1)-(113,41)]</td>
<td>1.256</td>
<td>1.256</td>
<td>200</td>
</tr>
<tr>
<td>FILL_OLD_LOC (FILL_OLD_LOC.F90) (66,7)-(162,33)]</td>
<td>5.104</td>
<td>8.214</td>
<td>41</td>
</tr>
<tr>
<td>NML_Alreduce0</td>
<td>1.93</td>
<td>1.93</td>
<td>41</td>
</tr>
<tr>
<td>NML_Barrier0</td>
<td>1.066</td>
<td>1.066</td>
<td>41</td>
</tr>
<tr>
<td>NML_Send0</td>
<td>0.048</td>
<td>0.048</td>
<td>15</td>
</tr>
<tr>
<td>NML_Wait0</td>
<td>0.047</td>
<td>0.047</td>
<td>15</td>
</tr>
<tr>
<td>NML_Irecv0</td>
<td>0.019</td>
<td>0.019</td>
<td>15</td>
</tr>
<tr>
<td>GRID_UPDATEREFINEMENT (Grid_updateRefinement.F90) (44,1)-(108,36)]</td>
<td>4.887</td>
<td>4,968.98</td>
<td>100</td>
</tr>
<tr>
<td>void io_h5SwmallocLists(int *, hid_t *, int <em>, char (</em>)[80], double *, int <em>, char (</em>)[80], int *, int *, cl</td>
<td>4.555</td>
<td>5.155</td>
<td>14</td>
</tr>
</tbody>
</table>

- 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
```