How to Debug Memory Problems

Finding memory problems in code can be a difficult task but there are tools available on the National Facility machines to make it possible. Memory errors can arise in many ways such as referencing arrays out of their declared bounds, failing to allocate dynamic arrays, attempting to free memory that cannot be freed or memory leaks leading to an increase in the memory requirements of the running program.

- Debuggers such as **idb** and **gdb** can be used to pinpoint lines where segmentation violations occur. However this may not be the actual point at which the error occured. The difficulty with memory problems is that the error may be reported some time after the incorrect line of code. This requires that the code is compiled with the `-g` option.
- For Fortran code it is always worth compiling with the option `-check bounds` to pinpoint any array subscript or substring references that are out of declared bounds at runtime.
- **Electric Fence** can be used to detect errors in C code where the boundaries of a `malloc()` are overrun or where there is an attempt to touch a memory allocation that has already been freed. Although the Electric Fence documentation will say that it can be used to debug MPI codes this does not work on the AC. It appears that the start-up procedure for the executable linked with `-lefence` invokes some memory allocations that interfere with the SGI mpirun start-up. Code must be linked with the libefence.a library before being executed.
- **valgrind** is part of the system software on Raijin and is very useful for both Fortran and C code. It does not require linking to a library but simply enter

  ```
  valgrind ./a.out
  ```

  The Valgrind tool **Memcheck** checks all reads and writes of memory and calls to malloc/new/free/delete are intercepted.
- **Mudflap** is part of the more recent versions of `gcc/g++`. To use mudflap you need to do at least

  ```
  module load gcc/4.0.1
  gcc prog.c -fmudflap -lmudflap
  a.out
  ```

  The environment variable `MUDFLAP_OPTIONS` can be used to control the output from mudflap. See http://gcc.gnu.org/wiki/Mudflap_Pointer_Debugging for more details.
- Totalview is installed on Raijin and it can be used to debug sequential, MPI or OpenMP programs written in C, C++ or Fortran. In order to use the memory debugging tools of Totalview you need to recompile your code as follows

  ```
  module load totalview
  ifort -g prog.f -L$TVLIB -ltvheap
  icc -g prog.c -L$TVLIB -ltvheap
  ```

  Then start up Totalview and click on Tools -> Memory Debugging.
  Under the Configuration tab make sure that “enable Memory Debugging” is chosen and click on any of the options that you wish to investigate.
  Run the code to the point where a memory error occurs. Say, for example, that this is a segmentation violation and the code stops. Then click on the “Heap Status” tab and highlight the process you want to look at in the left hand column.
  You can chose to look at the Source View, Backtrace View, Graphical View or Corrupted Guard Blocks by making the relevant choice on the lower left hand side and clinking “Generate View”.
  To find the line where a memory error is occuring, one method is to generate the Graphical View then chose the “Backtrace/Source” tab and double click on lines in the Backtrace window.
  You may need to extend the size of the window to see all the details.
- Intel Inspector, https://software.intel.com/en-us/intel-inspector-xe
- DrMemory, https://github.com/dynamorio/drmemory
- GNU mtrace, http://www.gnu.org/software/libc/manual/html_node/Tracing-malloc.html