



Debugging

Johnny Chang

NAS Division

NASA Ames Research Center

2012 Summer Short Course for Earth System
Modeling and Supercomputing



Bugs? What bugs?

Why is this topic being covered in this Summer School?

1. You might have to write some code (Fortran/C/C++) and need to debug
2. You get someone else's code and need to debug (an onerous task(!) especially if it is a "community" code developed by several people)
 - Why? Because you are getting wrong or different or no results
 - The first 2 of these could be a porting issue ...
 - The 3rd could be due to a program hang, segfault, abort, etc.
3. Debugging can be difficult ...
..., but there are many tricks and **tools** to help in this regard
4. I wish I knew this stuff when I was in grad school ...

Porting -- Getting a code, generally from somewhere else, to compile/build and generate "expected" results on the target machine

1. Compile with `-fp-model precise`
2. Lower compiler optimization: try compiling with `-O1` or `-O2` (default for Intel)
3. Try a different combination of compilers and libraries

There are 30+ compilers, 20+ MPI libraries, 6 NetCDF and 15 HDF libraries on Pleiades!



Debugging with Intel Compiler Options

For Fortran use: `-g -traceback -check -fpe0`

For C/C++ use: `-g -traceback`

`-g`

- Tells compiler to generate full debugging information in the object file (.o file)
- Changes the default optimization to `-O0`, so need to explicitly add `-O2` if no optimization level was previously specified
- Always compile with `-g` when using debuggers (i.e., TotalView)

`-traceback`

- Provides traceback information with source file, routine name, and line number when a severe error occurs at run time

```
forrtl: error (73): floating divide by zero
```

Image	PC	Routine	Line	Source
buggy	0000000000403144	sub1_	24	buggy.f
buggy	0000000000402ECF	MAIN__	13	buggy.f
buggy	0000000000402C0C	Unknown	Unknown	Unknown
libc.so.6	00007FFFECF2FBC6	Unknown	Unknown	Unknown
buggy	0000000000402B09	Unknown	Unknown	Unknown

Debugging with Intel Compiler Options (2 of 3)



-check (same as -check all)

- checks for array bounds violation (same as -check bounds)
 - Example: dimension a(100) and the code uses a(101) = ...
- checks for use of uninitialized variables (same as -check uninit)
 - Caution: the checking is very limited in scope
- checks for format, output_conversion, pointers, etc. ... generally, unlikely to be the culprit
- **Important: -check causes the program to run slow! Leave off after debugging**

-fpe0

- traps floating-point exceptions, i.e., divide-by-zero, sqrt of negative, etc.
- when compiling with -fpe0, **all** source files need to be compiled with this flag (or with -fp-speculation=off) to avoid false-positives

• Example: if (z .ne. 0.0) then

```
    y = 1/z
else
    y = 1 + z
endif
```

both branches are executed simultaneously
and one branch is discarded after evaluating
the if conditional

Debugging with Intel Compiler Options (3 of 3)



Example of array bounds violation message

```
forrtl: severe (408): fort: (2): Subscript #1 of the array Y has value 101 which
is greater than the upper bound of 100
```

Image	PC	Routine	Line	Source
buggy	000000000046AFCA	Unknown	Unknown	Unknown
buggy	0000000000469AC6	Unknown	Unknown	Unknown
buggy	0000000000421DE0	Unknown	Unknown	Unknown
buggy	0000000000404B6E	Unknown	Unknown	Unknown
buggy	0000000000405091	Unknown	Unknown	Unknown
buggy	00000000004033D9	sub1_	27	buggy.f
buggy	0000000000402F82	MAIN__	13	buggy.f
buggy	0000000000402C0C	Unknown	Unknown	Unknown
libc.so.6	00007FFFECECF2FBC6	Unknown	Unknown	Unknown
buggy	0000000000402B09	Unknown	Unknown	Unknown

Demo #1

buggy.f

- debug using `-g -traceback -check -fpe0`
- if arrays are dimensioned with 1 or *, i.e., `a(1)` or `a(*)`, then array bounds checking is ineffective
- note that checking of uninitialized data is limited in scope

Take home:

Debugging with compiler debug options is useful for catching most common bugs, but **not** all bugs. So, a code that passes a “health check” with the debug compiler options, does not mean it’s bug-free.

Demo #2



tv_ex1.f (and tv_ex2.f)

- run on cfe2 (Columbia)
- run on pfe with Intel compiler
- run on pfe with PGI compiler
- run with debug flags

Take home:

- bugs can cause different results with different compilers, different machines, different compiler options, etc.
- just because you get the same results doesn't mean the code is bug-free!



Debugging MPI codes with TotalView

1. Make sure that you can display X11 graphics from pfe
 - log in with `ssh -X` or add "ForwardX11 yes" to your `.ssh/config` file on your workstation
 - `echo $DISPLAY` should show some setting for the DISPLAY environment set by ssh
2. Compile your code with the `-g` compiler flag
3. Submit an interactive PBS job and forward your DISPLAY environment
 - `qsub -I -v DISPLAY -q devel@pbspl3 -lselect=N:ncpus=XX`
4. Once the PBS job has started, load the totalview module if it is not automatically loaded from your `.login` or `.cshrc` file
 - `module load totalview/8.9.2-1`
5. Start your mpiexec command with:
 - `mpiexec_mpt -tv -np nprocs ./a.out` (assumes using SGI's MPT library)

Online references on using TotalView available at:

- <http://www.roguewave.com/support/product-documentation/totalview.aspx#totalview>
- <http://www.roguewave.com/products/totalview/resources/videos.aspx>

Demo #4

What is $\lim_{x \rightarrow \infty} \sqrt{x^2 + 3x} - \sqrt{x^2 + 2x}$?

Write a program to calculate this limit.

What are the “Take home” lessons from this demo?

1. Is the issue round-off error?
2. Is the issue improper algorithm?
3. Sensitivity analysis: how sensitive are the results to uncertainties in (i) model parameters, (ii) measurement data, (iii) input data, (iv) algorithm approach, ...
4. Butterfly effect: Can the flapping of a butterfly's wing in Brazil cause a tornado in Texas?
5. What's the difference between non-determinism due to chaotic system vs. propagation of round-off errors.