QA Analysis of the WRF Program

John Collins[1,2,4,*] Mark Anderson[1] Brian Farrimond[1,4] David Gill[3]

1. Edge Hill University UK, 2. University of Cape Town, 3. NCAR, 4. SimCon Ltd. UK, * john.collins@simconglobal.com

Edge Hill University

Are there coding errors in WRF?

Are there Coding Errors in WRF?

All large programs contain errors. WRF is no exception.

Errors are of three types:

- Modeling Errors
- Coding Errors
- Compiler Errors

We have analyzed WRF as a computer program, *not* as a climatological model. The analysis exposes coding errors and compiler errors.

In WRF version 3.4, there are, for example:

2352	Inconsistent sub-program arguments
142	Errors in INTENT (IN) Declarations
62	Confirmed errors in INTENT (OUT) Declarations
552	Sections of unreachable code
4722	Mixed precision arithmetic statements
322	Uninitialized variables
8	Failed overloaded assignments

Is This Unusual?

The UK MOD asked us to create a simple metric of code quality. The metric we use is the ***millibug*** – the number of anomalies per 1000 lines of code, where errors are weighted as 10 times more significant than warnings.

WRF 3.4 has a score of about 20 anomalies per 1000 lines. The typical aerospace programs we work on have about 10.

WRF is about half as good as the code which designed the aircraft which got us here.

Does This Matter?

Possibly.

WRF is very well designed, well tested and well exercised. Most of the results are probably a good representation of the modeling carried out. However there will be occasions where the errors have significant effects.

We are developing tools to analyze the effects of the errors which have been identified, and these tools will be made available to the WRF community.

But the errors may not be as serious as they first appear.

Many Apparent Errors in WRF are Spurious or Harmless

Many of the errors and anomalies detected in WRF have no effect on the program performance. Over half of the inconsistent sub-program arguments are a consequence of re-mapping of array shapes across the call-site. Others are data type inconsistencies in routines which simple move data and do not use the data types for arithmetic processing. Many of the INTENT (IN) errors are a consequence of incorrect INTENT (INOUT) declarations elsewhere in the code, and many unreachable code sections are a deliberate consequence of pre-processing for parallel execution

Some error reports from our tools are incorrect. The analysis tools are not yet fully compliant with Fortran 2003, and some of the language features used in WRF (e.g. FLUSH statements and tagged IF-THEN-ELSE constructs) generate spurious errors. Work is in train to update the tools.

The most important systematic issue in the code is probably loss of precision due to mixed precision arithmetic, and this may be corrected by compiler options to force double precision throughout the program.

What Can Be Done?

We can correct the code.

All corrections to the code must be tested. Regression tests and tools for test result analysis are under development.

Some errors can be corrected simply by hand. The eight failed overloaded assignment operations, for example, are corrected by a one-line change in the code.

Correction of most of the errors must be automated. It is not practicable, for example, to correct 4722 mixed arithmetic expressions manually without injecting more (and worse) errors. Our Computer Aided Software Engineering (CASE) tools can carry out many of the changes automatically. Before this can be done, we must revise the WRF build procedure.

Why Must We Revise the Build Procedure?

WRF source code is not distributed as directly compilable code. It is pre-processed before compilation by custom-written programs and by the C pre-processor, cpp, to adapt to the specific compiler and multi-processor configuration. The analysis tools cannot analyze the pre-processor code.

The intention is to encapsulate the statements which are pre-processed in Fortran INCLUDE files, leaving a core of pure Fortran code which is not pre-processed and which may be re-engineered automatically.

Errors in WRF – Mixed Precision

Mixed precision arithmetic may result in an unexpected loss of precision. In the PARAMETER statement below, for example, the parameters `clcl1`, `clcl2`, `clcl3` and `clcl4` are declared to be 8-byte reals, and the values assigned to them are written with 6 digits after the decimal point. However, the numbers are written without a data kind tag and the exponent character is E, not D. They are therefore single precision values. The parameters have only single precision wherever they are used.

295 parameters in WRF are affected in this way.

```
File: phys/module_gfs_funcphys.f90
     REAL(krealfp),PARAMETER ::                                  &
      clcl1 = 0.954442E+0, clcl2 = 0.967772E-3,                  &
 !--------^--------------------^-----------------------------------
 !!! FPT - 3295 Parameter will not have the expected precision
 !!! FPT - 3295 Parameter will not have the expected precision
 !-----------------------------------------------------------------
      clcl3 = -0.710321E-3,clcl4 = -0.270742E-5
 !--------^--------------------^-----------------------------------
 !!! FPT - 3295 Parameter will not have the expected precision
 !!! FPT - 3295 Parameter will not have the expected precision
 !-----------------------------------------------------------------
```

There are about 3,000 similar cases where single and double precision numbers are mixed within the same expression. Not all are errors, but a majority almost certainly are.

Errors in WRF - What are Intent Errors?

A potentially serious error in WRF is the declaration of a sub-program argument to be INTENT (IN) when the argument is actually modified within the routine. For example:

```
File: share/module_io_domain.f90
SUBROUTINE input_boundary(fid,grid,config_flags,ierr)
        IMPLICIT NONE
        TYPE (domain) :: grid
        TYPE (grid_config_rec_type),INTENT(IN) :: config_flags
!-----------------------------------------------------------^----
!!! FPT - 2491 INTENT declared IN but argument is written to:
!---------------------------------------------------------------
        INTEGER,INTENT(IN) :: fid
        INTEGER,INTENT(INOUT) :: ierr
        IF (config_flags%io_form_boundary .GT. 0) THEN
           CALL input_wrf(fid,grid,config_flags,        &
            boundary_only,ierr)
        ENDIF
        RETURN
END SUBROUTINE input_boundary
```

The variable config_flags is then modified in input_wrf:

```
File: share/input_wrf.f90
SUBROUTINE input_wrf(fid,grid,config_flags,switch,ierr)
            :
        TYPE (grid_config_rec_type),INTENT(INOUT) :: config_flags
              :
        IF (IERR .NE. 0) THEN
           IF (MMINLU=='UMD') THEN
              config_flags%iswater = 14
           ELSE
              config_flags%iswater = 16
           ENDIF
        ENDIF
```

The problem is that the compiler may use the INTENT statement to optimize the code, and may do so incorrectly.

Clutter

A difficulty in analyzing WRF is the large number of harmless inconsistencies in the code. These do not cause errors, but real errors are hidden amongst them. For example, there are 580 inconsistencies in array bounds in subroutine arguments, such as that shown below.

```
Line:  2508,   File: phys/module_mp_wdm6.f90
CALL slope_graup(qr2,den,denfac,tk,tmp,tmp1,tmp2,tmp3,wa2,1,1,1 &
 ,km)

Argument 1   Formal argument            Actual argument
----------   ---------------            ---------------
Argument     qrs                        qr2
Protocol     By reference               By reference
Usage        Symbol                     Symbol
Access       Read                       Read/Write
Data type    REAL                       REAL
Data size    *4 df                      *4 df
Dimensions   (*,*)                      (*)                    ***
```

The array qr2 has been re-mapped across the call-site from one dimension to two. This almost certainly works as intended, but is trapped as a potential error and all occurrences must be checked by hand. We hope to correct issues such as this.

Can You Help?

If you are aware of specific errors in WRF, or of conditions which cause exceptions or invalid results, please tell us. We may know possible causes or we may be able to refine our analyses to find them.

An Apology

This poster describes errors and inconsistencies in the WRF Fortran code. It is important to stress that the authors view WRF as an outstanding engineering achievement and have great respect for the program developers.

We come to contribute to WRF, not to criticize it.

Acknowledgments

This work is supported by:

The United States Air Force

UCAR and NCAR

Edge Hill University

The UK Higher Education Innovation Fund (HEIF) at Liverpool Hope University

SimCon Ltd.

The authors wish to thank:

Prof. W. J Gutowski Jr. Iowa State University

Prof. Bruce Hewitson, University of Cape Town

for their help and support.