ORACLE
SOLARIS

Oracle Developer Studio
Fortran Runtime Checking Options

ORACLE WHITE PAPER | MAY 2016

ORACLE

Table of Contents

Introduction

The Oracle Solaris Studio Fortran compiler provides several command line options to enable runtime checks that can make it easier to find and fix bugs in your Fortran programs. This article describes the various options and how to use them.

Note: Oracle Solaris Studio was formerly called Sun Studio.

The following tools are not covered in this article, but they can be extremely helpful:

- The Thread Analyzer detects errors, such as data races and deadlocks, in parallel programs. See the *Oracle Solaris Studio 12.2: Thread Analyzer User's Guide:* http://docs.oracle.com/cd/E18659_01/html/821-2124/index.html.
- The dbx runtime checking feature, rtc, detects memory access errors, such as reading uninitialized variables, memory leaks, and so on. See the *Oracle Solaris Studio 12.2: Debugging a Program With dbx* manual: https://docs.oracle.com/cd/E18659_01/html/821-1380/toc.html.

Fortran Runtime Checking Options

Here is a description of the Oracle Solaris Studio Fortran runtime checking options:

» `-C`

Enables array bounds, conformance, and allocation status checking.

» `-xcheck=[init_local | stkovf]`

init_local initializes local and allocated variables to special garbage values that are likely to cause an arithmetic exception if the variables are read before valid values have been assigned.

`stkovf` detects a stack overflow at run time.

» `-xassume_control=[check[,fatal]]`

Checks assertions specified with the ASSUME pragma, emitting warning messages if any assertions are false. If fatal is used and an assertion is false, a message is emitted and the program is terminated.

» `-xcommonchk`

Detects when a common block is defined THREADPRIVATE in some but not all modules.

» `-fpover`

Detects floating-point overflow in formatted input.

Array Bounds Checking

The -C option is probably the most helpful of the runtime checking options.

It enables, among other things, array bounds checking. If an array subscript is beyond the declared bounds of the array, an error message is printed and the program is aborted.

Example:

```
% cat test.f
        subroutine copy(a,b)
        integer a(:), b(:), i
        do i = lbound(a,1), ubound(a,1)
                a(i) = b(i)
        end do
        end subroutine

        Program
        integer, parameter :: n = 20
        integer a(1:n), b(1:n-1), i ! notice b is declared 1:n-1
        interface
                subroutine copy(a,b)
                integer a(:), b(:), i
```

```
              end subroutine
       end interface

       call copy(a,b)
       end program

% f95 -C test.f
% a.out

       ******  FORTRAN RUN-TIME SYSTEM ******
       Subscript out of range. Location: line 4 column 12 of 'x.f'
       Subscript number 1 has value 20 in array 'B'
       Abort
```

Optimization and -C

The `-C` option can be used with any level of optimization. At higher optimization levels, checking usually has only minimal effects on performance, because the checks are optimized along with the rest of the code.

Arrays with a Last Dimension of Size 1

Some FORTRAN 77 programs declare dummy arguments with the last dimension of size 1, rather than *. In order to avoid runtime errors in working programs, the compiler does not generate bounds checks for the last dimension for dummy or Cray pointer arrays whose last dimension has size 1.

Array Syntax Conformance Checking

The `-C` option also enables conformance checking of array syntax statements.

Example:

```
% cat test.f
       subroutine copy(a,b,n,m)
       integer a(n), b(m)

       a = b
       end

       program
```

```
        integer, parameter :: n = 10
        integer a(n), b(n-1)
        call copy(a,b,n,n-1)
        end

% f95 -O3 -C test.f
% a.out
        ******  FORTRAN RUN-TIME SYSTEM ******
        An array expression does not conform : dim 1 is 9 , not 10
        Location:  line 4 column 11 of 'test.f'
        Abort
```

Garbage Initialization for Local Variables

The `-xcheck=init_local` option can help you detect when a local variable is used before it is set. When this option is used, local variables are initialized to unusual values that are likely to cause an exception when used. The initialization is currently done for the following variables:

- Local stack or automatic variables
- Allocated variables (those allocated with `MALLOC` or `ALLOCATE`)

This option is especially helpful when using the `-stackvar` option, which causes local variables to be allocated on the stack, rather than statically. Note that the `-stackvar` option is implicitly included in `-xopenmp`.

Example:

```
% cat test.f
subroutine init_local

real a, b, arr(10)
integer c, d
logical e
complex f
character*4 string
integer, pointer :: ptr(:)
integer  v(10)
pointer(p,v)

print *, 'real=', a, ', int=', c
```

```
print *, 'logical=', e, ', complex=', f

print *, 'real array=', arr

print *, 'char*4=', string

allocate(ptr(6))

print *, 'allocated ptr=', ptr

p = malloc(sizeof(integer)*10)

print *, 'malloced cray ptr=', v

end subroutine


call init_local end


% f95 -stackvar -xcheck=init_local test.f

% a.out


real= -NaN , int= -8388565 logical= T , complex= (-NaN,-NaN)

real array= -NaN -NaN -NaN -NaN -NaN -NaN -NaN -NaN -NaN --NaN
char*4=\205\205\205\205

allocated ptr= -8388565 -8388565 -8388565 -8388565 -8388565 -8388565

malloced cray ptr= -8388565 -8388565 -8388565 -8388565 -8388565 -8388565

-8388565 -8388565 -8388565 -8388565
```

If, while debugging, you encounter a local variable with an unusual value, such as those shown above, there is a good chance the variable has not been initialized.

Caution: Do not rely on the particular initial values that the compiler currently uses, because they are subject to change.

Stack Overflow Checking

The `-xcheck=stkovf` option can be very useful for OpenMP programs, where the stack can be smaller than usual.

Example:

```
% cat test.f

program

call overflow

end
```

```
subroutine overflow

integer, parameter :: n = 16000

integer*8 :: a(n), b(n), c(n), d(n)

a = b

c = d

print *,a(1:10),c(1:10)

end


% f95 -stackvar -xcheck=stkovf test.f

% limit stacksize 100 # to force a very small stack

%  a.out

Segmentation Fault


% dbx a.out

dbx> run

t@1 (l@1) signal SEGV (no mapping at the fault address) in

_stack_grow_probing_impl at 0x1347c

0xfefcca94: _stack_grow+0x0048: ldub       [%o1 - 1], %g0
```

The previous example is on a machine that has a SPARC processor. On a machine that has an x86 processor, you may see the following:

```
0x080509c9: overflow_+0x0009:       movl    $0x00000001,0xfff82fdc(%ebp)
```

The segmentation fault may indicate that stack overflow may have taken place. To check, run dbx; if the executable terminates in one of the above functions or in similar functions, stack overflow has occurred.

ASSUME Pragmas

The `-xassume_control=check` option causes the compiler to check the value of `ASSUME` pragmas and generate a message when the value is not `true`.

ASSUME pragmas are described in detail in the *Oracle Solaris Studio 12.2: Fortran User's Guide* at http://docs.oracle.com/cd/E18659_01/html/821-1382/index.html.

When they are not being used for runtime checking, `ASSUME` pragmas can improve optimization by specifying invariants that the compiler can use to improve the generated code.

Example:

```
subroutine check (a,b,c)
integer a, b, c, d

c$pragma assume (a > 0, 1.0) ! 1.0 means 100% certainty, which enables checking
c$pragma assume (b > 0, 1.0)

print *, a, b, c

d = 0
c$pragma begin assume ( tripcount().lt.1000, 1.0)
do i = 1, c
       d = d + 1
end do
c$pragma end assume

print *, d
end

call check(1, -1, 1000)
end
```

With `-xassume_control=check`, when an `ASSUME` error is detected, a message is printed and execution continues:

```
% f95 -xO3 -xassume_control=check test.f
% a.out
        Assume check failure at assume.f:5
        1 -1 1000
        Assume check failure at assume.f:10
        1000
```

With `-xassume_control=check,fatal`, when an `ASSUME` error is detected, a message is printed and the program is terminated:

```
% f95 -xO3 -xassume_control=check,fatal test.f
% a.out
```

```
Assume check failure at assume.f:5
```

Note: Only `ASSUME` pragmas with probabilities of 1.0 are checked at run time, and any level of optimization can be used.

THREADPRIVATE Common Checking

The `-xcommonchk` option adds runtime checking to detect when a common block is defined `THREADPRIVATE` in some, but not all, program units. This option should be used for debugging only because it can degrade performance.

Example:

```
common.f:

program

real a(100)

common /mycommon/ a

call abc()

end program


abc.f:

       subroutine abc()

       real a(100)

       common /mycommon/ a

       !$omp threadprivate (/mycommon/)

       a = 0

end subroutine


% f95 -O3 -xopenmp -xcommonchk common.f abc.f

% setenv OMP_NUM_THREADS 2

% a.out


       ERROR (libmtsk): at abc.f:3. Inconsistent declaration of threadprivate

       mycommon_ : Not declared as threadprivate at line 4 of common.f
```

Floating Point Overflow Checking

The `-fpover` option enables a program to handle floating-point overflow in formatted input.

Example:

```
test.f:
       integer status
       real r

       read (*, *, iostat=status) r
       if (status.eq.1031) then
       print *, 'Floating point overflow on input', r
       end if
       end

input:
       1e+100

The -ftrap=%none flag is required with -fpover.

% f95 -fpover -ftrap=%none test.f
% a.out <input
Floating point overflow on input Inf
```

If the `read` statement does not contain an `iosat=` argument, a Fortran runtime error is emitted when floating point overflow is detected in a formatted input value.

Example:

```
test.f:
       integer status
       real r

       read (*, *) r
       print *, r

       end

% f95 -fpover -ftrap=%none test.f
% a.out <input
```

```
******  FORTRAN RUN-TIME SYSTEM  ******
Error 1031: floating-point overflow on input
Unit: *
File: standard input
Abort
```

Conclusion

This paper described how to use the Oracle Solaris Studio Fortran compiler options to enable runtime checks. Using the options can help you find and fix bugs in your Fortran programs.

For More Information

Here are additional resources.

- Oracle Solaris Studio 12.2 f95(1) man page (you can access this man page using the `man` command from the installed product)
- *Oracle Solaris Studio 12.2 Fortran User's Guide*: http://docs.oracle.com/cd/E18659_01/html/821-1382/index.html
- *Oracle Solaris Studio 12.2 Debugging a Program With dbx*: https://docs.oracle.com/cd/E18659_01/html/821-1380/toc.html
- Oracle Solaris Studio 12.2 documentation page: http://docs.oracle.com/cd/E18659_01/index.html

Oracle Corporation, World Headquarters
500 Oracle Parkway
Redwood Shores, CA 94065, USA

Worldwide Inquiries
Phone: +1.650.506.7000
Fax: +1.650.506.7200

ORACLE

CONNECT WITH US

blogs.oracle.com/oracle
facebook.com/oracle
twitter.com/oracle
oracle.com

Integrated Cloud Applications & Platform Services

Copyright © 2016, Oracle and/or its affiliates. All rights reserved. This document is provided *for* information purposes only, and the contents hereof are subject to change without notice. This document is not warranted to be error-free, nor subject to any other warranties or conditions, whether expressed orally or implied in law, including implied warranties and conditions of merchantability or fitness for a particular purpose. We specifically disclaim any liability with respect to this document, and no contractual obligations are formed either directly or indirectly by this document. This document may not be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose, without our prior written permission.

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Intel and Intel Xeon are trademarks or registered trademarks of Intel Corporation. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. AMD, Opteron, the AMD logo, and the AMD Opteron logo are trademarks or registered trademarks of Advanced Micro Devices. UNIX is a registered trademark of The Open Group. 0615

Oracle Solaris Studio Fortran Runtime Checking Options
May 2016
Author: Diane Meirowitz

Oracle is committed to developing practices and products that help protect the environment