IBM XL Fortran for AIX

Debugging Optimized Code

By: Grigor Nikolov

Level: Introductory

May 2012

Contents

Copyright © 2012, IBM® Corporation. Published by IBM® developerWorks®.

About this Tutorial

This tutorial shows new options that control the debugging support and can be used to debug optimized Fortran code using the dbx debug program and the XL Fortran compiler.

Objectives

- Show the new options that are available to debug optimized code.
- Total time: 30 minutes

Prerequisites

- Basic Unix skills
- Basic Source code compile and build experience
- Basic knowledge of Fortran programming language
- Basic knowledge of dbx debug program

System Requirements

http://www.ibm.com/software/awdtools/fortran/xlfortran/aix/sysreq

Copyright © 2012, IBM® Corporation. Published by IBM® developerWorks®.

Glossary

IBM XL Fortran Compiler: The IBM® XL Fortran compiler offers advanced compiler and optimization technologies and is built on a common code base for easier porting of your applications between platforms. It complies with the latest Fortran international standards and industry specifications and supports a large array of common language features.

Copyright © 2012, IBM® Corporation. Published by IBM® developerWorks®.

Introduction

There are new options available to control the debugging support and to make debugging optimized code easier, specifically with –O2. When the –O2 optimization level is in effect the debug capability is completely supported. When an optimization level is higher than –O2 is in effect the debug capability is limited. The debug option –g has levels that range from 0 to 9.

-g0
: No debug information is generated. No program state is preserved.

-g1
: Generates minimal view-only debugging information about line numbers and source files names. No program state is preserved.

-g2
: Generates minimal view-only debugging information about line numbers, variables and source files names. At –O2 no program state is preserved.

-g3, -g4
: Same behavior as –g2 plus function parameter values are available to the debugger at the beginning of each function.

-g5, -g6, -g7
: Generates minimal view-only debugging information about line numbers, variables and source files names. When the -O2 optimization level is in effect program state is available to the debugger at if constructs, loop constructs, function definitions, and function calls, and function parameter values are available to the debugger at the beginning of each function.

-g8
: Generates view-only debugging information about line numbers, source file names, and variables. When the -O2 optimization level is in effect, program state is available to the debugger at the beginning of every executable statement, and function parameter values are available to the debugger at the beginning of each function.

-g9
: Same behavior as –g8 plus you can modify the value of the variables in the debugger.

This demo will demonstrate –g8, -g5 and –g9 behavior.

Copyright © 2012, IBM® Corporation. Published by IBM® developerWorks®.

Getting Started

Start the Terminal Emulator to AIX System

Figure 1 Get Started

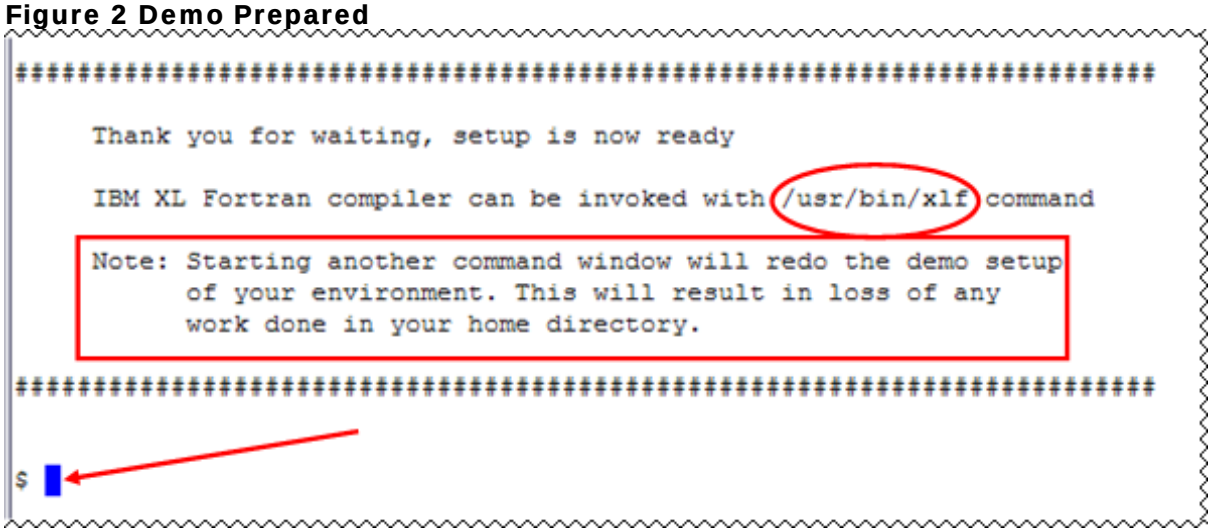
Double click the “Launch AIX” icon on the desktop (See Figure 1) to start the character terminal to AIX system.

Get Started with IBM XL Fotran Debugging Optimized Code

Successful login will result with user presented with a menu of demo hosted on the server. Type 21 and press Enter to select the “Debugging Optimized Fortran Code” demo.

Copyright © 2012, IBM® Corporation. Published by IBM® developerWorks®.

Figure 2 Demo Prepared



On the terminal window you will see important information and directory path to compiler install directory (See Figure 2 Demo Prepared oval red).

Note: Starting another command window will start the demonstration setup of your environment. This will result in loss of any work done in your home directory. This will impact any progress you have made on demo steps going forward.

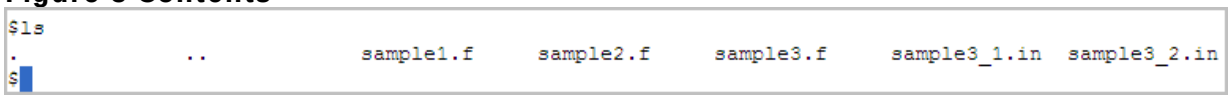
This demo does not require more than one terminal window. However, if you prefer more than one terminal window then you may open them before going forward.

Terminal window is now ready for commands (See Figure 2 Demo Prepared arrow). Your home directory contains necessary source code to perform the tutorial. Type `ls` command to see the directory content (See Figure 3 Contents).

Command:

```
ls
```

Figure 3 Contents



Two Fortran programs are provided here:

sample1.f	Basic code with internal, external and module procedures with variables name reused in different scopes, type variables, if and loop constructs
sample2.f	Basic code with many easy for optimization opportunities.
sample3.f	Basic code with loop with relatively much iterations.
sample3_1.in	Debugger input file for sample3.f source with breakpoint after the loop
sample3_2.in	Debugger input file for sample3.f source with stepping to the loop

Copyright © 2012, IBM® Corporation. Published by IBM® developerWorks®.

Showing the debugability of optimized code

1. To show the debugability of optimized code compile sample1.f with debug level 8 and optimization level 2 (See figure S1.1 Compile 1)

   Command:
   ```
   xlf2003 -g8 -O2 -o sample1 sample1.f
   ```

Figure S1.1 Compile 1

```
$xlf2003 -g8 -O2 -o sample1 sample1.f
** mmod   === End of Compilation 1 ===
** sample1   === End of Compilation 2 ===
** ext_sub   === End of Compilation 3 ===
1501-510  Compilation successful for file sample1.f.
$
```

2. Now with the executable program sample run dbx (See figure S1.2 Run dbx)

   Command:
   ```
   dbx sample1
   ```

Figure S1.2 Run dbx

```
$dbx sample1
Type 'help' for help.
reading symbolic information ...
(dbx)
```

3. Set the first breakpoint in the program and start the run – dbx stops at the first executable line of the program (See figure S1.3 Set breakpoint)

   DBX Command
   ```
   stop in sample1
   run
   ```

Figure S1.3 Set breakpoint

```
(dbx) stop in sample1
[1] stop in sample1
(dbx) run
[1] stopped in sample1 at line 46 in file "sample1.f"
   46         res = intro_value
(dbx)
```

4. At this point most of the variables are not yet initialized. Will step a little through the program and then output the type and value of a variable: (See figure S1.4 Step and Print [1])

   DBX Command:

Copyright © 2012, IBM® Corporation. Published by IBM® developerWorks®.

```
step
step
step
next
whatis res
print res
```

Figure S1.4 Step and Print [1]

```
(dbx) step
stopped in sample1 at line 47 in file "sample1.f"
   47         extres = (res*res, res/res)
(dbx) step
stopped in sample1 at line 48 in file "sample1.f"
   48         print *,"P1 res=", res, "    extres=",extres
(dbx) step
 P1 res= -4     extres= (16.00000000,1.000000000)
stopped in sample1 at line 50 in file "sample1.f"
   50         call int_sub()
(dbx) next
 I1 res= 5046     extres= (16.00000000,1.000000000)
stopped in sample1 at line 51 in file "sample1.f"
   51         print *,"P2 res=", res, "    extres=",extres
(dbx) whatis res
 integer*4 res
(dbx) print res
5046
(dbx)
```

5. Now we will continue to step inside a procedure and print some values in it (See figure S1.5 Step and Print [2]).

   DBX Command:

   ```
   step
   step
   step
   whatis res
   print res
   ```

Copyright © 2012, IBM® Corporation. Published by IBM® developerWorks®.

Figure S1.5 Step and Print [2]

```
(dbx) step
 P2 res= 5046     extres= (16.00000000,1.000000000)
stopped in sample1 at line 53 in file "sample1.f"
   53         call ext_sub(res)
(dbx) step
stopped in ext_sub at line 92 in file "sample1.f"
   92         res = 14
(dbx) step
stopped in ext_sub at line 93 in file "sample1.f"
   93         extres = extres+res
(dbx) whatis res
 integer*2 res
(dbx)  print res
14
(dbx)
```

6. Then through a new breakpoint will go back to the main program and after that through stepping will enter a module subroutine (See figure S1.6 Stepping into module subroutine)

   DBX Command:

```
stop at 56
cont
step
whatis res
print res
step
step
whatis res
print res
```

Copyright © 2012, IBM® Corporation. Published by IBM® developerWorks®.

Figure S1.6 Stepping into module subroutine

```
(dbx) stop at 56
[3] stop at "sample1.f":56
(dbx)  cont
 E1 res= 14     extres= 5060
 P3 res= 5060     extres= (16.00000000,1.000000000)
[3] stopped in sample1 at line 56 in file "sample1.f"
   56         res = intro_value
(dbx) step
stopped in sample1 at line 57 in file "sample1.f"
   57         call mod_sub(res)
(dbx) whatis res
 integer*4 res
(dbx) print res
-4
(dbx) step
stopped in sample1.__mmod_NMOD__&&_mmod.__mmod_NMOD_mod_sub at line 12 in file "sample1.f"
   12              res    = REAL(extres)
(dbx)  step
stopped in sample1.__mmod_NMOD__&&_mmod.__mmod_NMOD_mod_sub at line 13 in file "sample1.f"
   13              modres = INT(res)
(dbx) whatis res
 real*8  res
(dbx) print res
16.0
(dbx)
```

7. Finally we will go back in the main program before IF and DO statements, we will execute the loop once and exit (See figure S1.7 Stepping into loop)

   DBX Command:

```
stop at 62
cont
step
step
step
step
step
step
step
step
cont
quit
```

Copyright © 2012, IBM® Corporation. Published by IBM® developerWorks®.

Figure S1.7 Stepping into loop

```
(dbx) stop at 62
[4] stop at "sample1.f":62
(dbx) cont
 M1 res= 16.0000000000000000     extres= (32.00000000,2.000000000)    modres= 16
 P4 res= 16     extres= (32.00000000,2.000000000)
[4] stopped in sample1 at line 62 in file "sample1.f"
   62         odt1 = dt_l1(81, dt_l2(-13, -3.625, (8.5,-5.25)))
(dbx) step
stopped in sample1 at line 64 in file "sample1.f"
   64         if (res == 16) then
(dbx) step
stopped in sample1 at line 65 in file "sample1.f"
   65           print *, "\n P10 odt2=", odt2
(dbx) step

 P10 odt2= 5 78.50000000 (945.1250000,-3752.625000)
stopped in sample1 at line 66 in file "sample1.f"
   66           print *, "P11 odt1=", odt1
(dbx) step
 P11 odt1= 81 -13 -3.625000000 (8.500000000,-5.250000000)
stopped in sample1 at line 67 in file "sample1.f"
   67           do j=1, 1000,2
(dbx) step
stopped in sample1 at line 68 in file "sample1.f"
   68              odt1%i8 = j*res/odt2%i4 + (odt1%i8-odt2%i4)
(dbx) step
stopped in sample1 at line 69 in file "sample1.f"
   69              odt2%i4 = odt1%dt_in%i4-j
(dbx) step
stopped in sample1 at line 70 in file "sample1.f"
   70           end do
(dbx) step
stopped in sample1 at line 68 in file "sample1.f"
   68              odt1%i8 = j*res/odt2%i4 + (odt1%i8-odt2%i4)
(dbx) cont
 P12 odt2= -1012 78.50000000 (945.1250000,-3752.625000)
 P13 odt1= 248274 -13 -3.625000000 (8.500000000,-5.250000000)

execution completed
(dbx) quit
$
```

Note: Instead of setting breakpoints and jumping to them, you may use more extensively step command to see how it goes through all executable statements.

Copyright © 2012, IBM® Corporation. Published by IBM® developerWorks®.

Using debug option –g5

1. With optimization the debug option -g5 makes the program state available to the debugger at if constructs, loop constructs, procedure definitions, and procedure calls, and procedure parameter values are available to the debugger at the beginning of each procedure. This option reduces the object size and increases the compile time performance while still providing some ability to debug the program.

   Compile the sample1.f source code with debug level 5 and optimization level 2 (See figure S2.1 Compile).

   Command:
   ```
   xlf2003 -g5 -O2 -o sample1 sample1.f
   ```

Figure S2.1 Compile

```
$xlf2003 -g5 -O2 -o sample1 sample1.f
** mmod   === End of Compilation 1 ===
** sample1   === End of Compilation 2 ===
** ext_sub   === End of Compilation 3 ===
1501-510  Compilation successful for file sample1.f.
$
```

2. Run DBX and try to set breakpoints

   DBX Command:
   ```
   stop at 56
   stop at 57
   ```

Figure S2.2 Set breakpoints

```
$dbx sample1
Type 'help' for help.
reading symbolic information ...
(dbx) stop at 56
no executable code at line "sample1.f":56
(dbx) stop at 57
[2] stop at "sample1.f":57
```

The first set of breakpoint fails since implicit break points are inserted only before and after if constructs, loop constructs and function. An implicit break point is also inserted at the first executable line of a function to guarantee at least one stop in each function. The line numbers for the rest of the lines are cleared. So most lines with only calculations are not possible to stop at.

3. Now try several step commands. With g5, the debugger steps over variable declarations (unlike g8) to loops, control blocks and function calls (and first executable line of a function). In this case the first step brings the sessions to the first executable line of the main program, but the next one jumps to the following function call.

   DBX Command:
   ```
   step
   step
   ```

Copyright © 2012, IBM® Corporation. Published by IBM® developerWorks®.

```
step
step
step
step
step
whatis res
step
whatis res
print extres
step
```

Figure S2.3 Stepping with g5

```
(dbx) step
stopped in sample1 at line 46 in file "sample1.f"
   46         res = intro_value
(dbx) step
 P1 res= -4     extres= (16.00000000,1.000000000)
stopped in sample1 at line 50 in file "sample1.f"
   50         call int_sub()
(dbx) step
stopped in int_sub at line 78 in file "sample1.f"
   78           do i=1, size1
(dbx) step
stopped in int_sub at line 81 in file "sample1.f"
   81           print *,"I1 res=", res, "    extres=",extres
(dbx) step
 I1 res= 5046     extres= (16.00000000,1.000000000)
stopped in int_sub at line 83 in file "sample1.f"
   83         end subroutine int_sub
(dbx) step
stopped in sample1 at line 51 in file "sample1.f"
   51         print *,"P2 res=", res, "    extres=",extres
(dbx) step
 P2 res= 5046     extres= (16.00000000,1.000000000)
stopped in sample1 at line 53 in file "sample1.f"
   53         call ext_sub(res)
(dbx) whatis res
 integer*4 res
(dbx) step
stopped in ext_sub at line 92 in file "sample1.f"
   92         res = 14
(dbx) whatis res
 integer*2 res
(dbx) print extres
5046
(dbx)  step
 E1 res= 14     extres= 5060
stopped in ext_sub at line 96 in file "sample1.f"
   96       end subroutine ext_sub
```

Inside procedures (ext_sub in the example) we can view the value of the parameters.

4. Continue with step and next commands to investigate allowable statements.

Copyright © 2012, IBM® Corporation. Published by IBM® developerWorks®.

Using the debug option –g8 and optimization on highly optimizable code

1. Compile the sample2.f source code with debug level 8 and optimization level 2 (See figure S3.1 Compile).

   Command:

   ```
   xlf2003 –g8 -O2 -o sample2 sample2.f
   ```

Figure S3.1 Compile

```
$xlf2003 -g8 -O2 -o sample2 sample2.f
** sample2   === End of Compilation 1 ===
** same_val   === End of Compilation 2 ===
1501-510  Compilation successful for file sample2.f.
$
```

2. Run DBX and start stepping

   DBX Command:

   ```
   step
   step
   step
   step
   step
   print i
   step
   step
   step
   step
   step
   step
   step
   ```

Copyright © 2012, IBM® Corporation. Published by IBM® developerWorks®.

Figure S3.2 Stepping behavior

```
$dbx sample2
Type 'help' for help.
reading symbolic information ...
(dbx) step
stopped in sample2 at line 13 in file "sample2.f"
   13           i = 5
(dbx) step
stopped in sample2 at line 14 in file "sample2.f"
   14           i = -1
(dbx) step
stopped in sample2 at line 15 in file "sample2.f"
   15           j = i
(dbx) step
stopped in sample2 at line 16 in file "sample2.f"
   16           m = 5
(dbx) step
stopped in sample2 at line 17 in file "sample2.f"
   17           i = 4
(dbx) print i
-1
(dbx) step
stopped in sample2 at line 18 in file "sample2.f"
   18           print *, "(first)  i:", i
(dbx) step
 (first)  i: 4
stopped in sample2 at line 20 in file "sample2.f"
   20           if (i == 4) then
(dbx) step
stopped in sample2 at line 21 in file "sample2.f"
   21             do k=1,10
(dbx) step
stopped in sample2 at line 22 in file "sample2.f"
   22               r4 = 16.0
(dbx) step
stopped in sample2 at line 23 in file "sample2.f"
   23               j  = k
(dbx) step
stopped in sample2 at line 24 in file "sample2.f"
   24             end do
(dbx) step
stopped in sample2 at line 22 in file "sample2.f"
   22               r4 = 16.0
(dbx)
```

3. If you look at the code you can see that it offers many opportunities for the optimizer. Yet with –g8 every executable statement is preserved and corresponding values kept.

Please, continue further debugging to verify accessibility of every executable statement.

Copyright © 2012, IBM® Corporation. Published by IBM® developerWorks®.

Using the debug option –g9

1. Compile the sample2.f source code with debug level 9 and optimization level 2 (See figure S4.1 Compile).

   Command:
   ```
   xlf2003 -g9 -O2 -o sample2 sample2.f
   ```

Figure S4.1 Compile

```
$xlf2003 -g9 -O2 -o sample2 sample2.f
** sample2   === End of Compilation 1 ===
** same_val   === End of Compilation 2 ===
1501-510  Compilation successful for file sample2.f.
$
```

2. Run DBX and stop at line 15
   DBX Command:
   ```
   stop at 15
   run
   ```

Figure S4.2 Stop at desired line

```
$dbx sample2
Type 'help' for help.
reading symbolic information ...
(dbx) stop at 15
[1] stop at "sample2.f":15
(dbx) run
[1] stopped in sample2 at line 15 in file "sample2.f"
   15           j = i
(dbx)
```

3. Now print the value of ‘i’, then change it and move further
   DBX Command:
   ```
   print i
   assign i=77
   step
   print j
   ```

Figure S4.3 Variable modification

```
(dbx) print i
-1
(dbx) assign i=77
(dbx) step
stopped in sample2 at line 16 in file "sample2.f"
   16           m = 5
(dbx) print j
77
(dbx)
```

As you see the variable ‘j’ took the modified value of ‘i’.

Copyright © 2012, IBM® Corporation. Published by IBM® developerWorks®.

When debug option –g9 is used, variables modification will be visible to the executing program regardless of the optimization level. This is the traditional behavior for non optimized code. With the new debug levels it is only guaranteed at -g9

Copyright © 2012, IBM® Corporation. Published by IBM® developerWorks®.

Stepping behavior with big loops and debug option –g5

1. Compile the sample3.f source code with debug level 5 and optimization level 2 (See figure S5.1 Compile).

   Command:
   ```
   xlf2003 -g5 -O2 -o sample3 sample3.f
   ```

Figure S5.1 Compile

```
$xlf2003 -g5 -O2 -o sample3 sample3.f
** sample3   === End of Compilation 1 ===
1501-510  Compilation successful for file sample3.f.
$
```

2. Now we will run DBX using commands input file sample3_1.in (with breakpoint after the loop) and time the execution

   Command:
   ```
   time dbx sample3 <sample3_1.in
   ```

Figure S5.2 Jumping directly after the loop

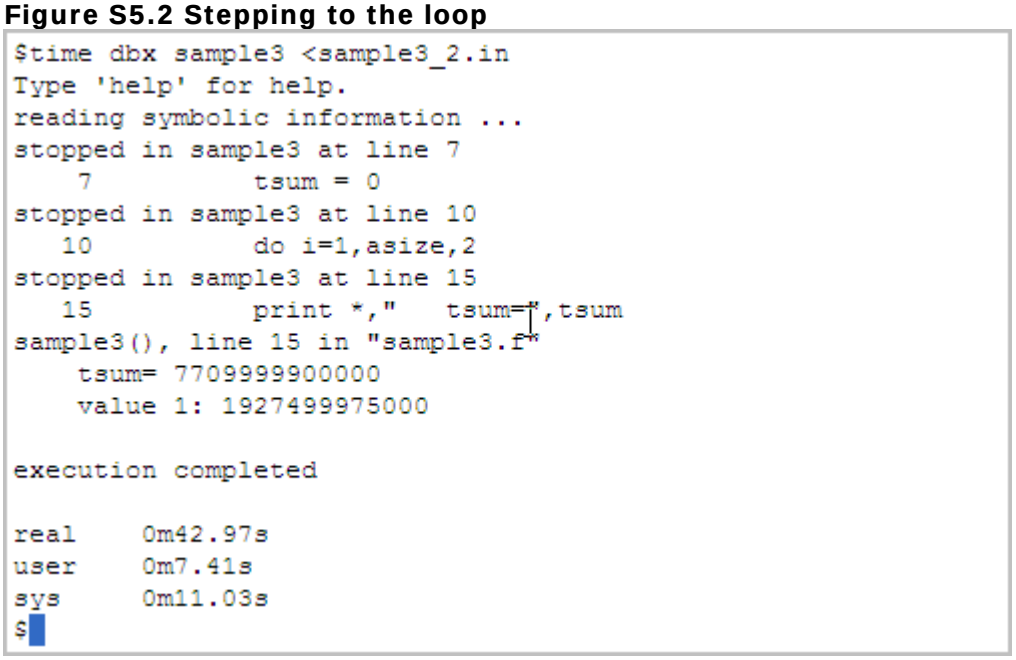
```
$time dbx sample3 <sample3_1.in
Type 'help' for help.
reading symbolic information ...
[1] stop at 15
stopped in sample3 at line 7
    7           tsum = 0
[1] stopped in sample3 at line 15
   15           print *,"   tsum=",tsum
sample3(), line 15 in "sample3.f"
    tsum= 7709999900000
    value 1: 1927499975000

execution completed

real    0m0.12s
user    0m0.02s
sys     0m0.01s
$
```

3. In similar fashion we will use the other commands input file sample3_2.in where we do step into the loop and again time the execution

   Command:
   ```
   time dbx sample3 <sample3_2.in
   ```

Copyright © 2012, IBM® Corporation. Published by IBM® developerWorks®.

Figure S5.2 Stepping to the loop



While in the first case the debug session completed almost instantaneously, the second took considerably more time (***Note:*** Times shown in above screenshots are just for reference. They vary depending on the system and the current load).

The example is only to show that stepping into loops when compiling with optimization and –g5 debug level should be used with caution and some performance variation should be expected.

Copyright © 2012, IBM® Corporation. Published by IBM® developerWorks®.

Conclusion

In this demo, you have observed the differences between the debug information displayed at various g levels in combination with optimization level 2.

Trademarks

IBM and the IBM logo are trademarks of International Business Machines Corporation in the United States, other countries or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Other company, product and service names may be trademarks or service marks of others.

Resources

XL Fortran for AIX, Library :

http://www.ibm.com/software/awdtools/fortran/xlfortran/aix/library/

Community Cafe :

http://www.ibm.com/rational/community/fortran

Copyright © 2012, IBM® Corporation. Published by IBM® developerWorks®.