IBM XL C/C++ and Fortran Compiler for AIX

Exploring auto-vectorization using MASS library supported in IBM XL C/C++ and Fortran Compilers for AIX

By: Robert Enenkel,
Ji Chao Zhang,
Jim Xia,
Daniel Zabawa

Level: Introductory

May 2012

Contents

Before you start

About this series

Walk through this scenario and others online as part of the IBM XL C/C++ and Fortran Compiler for AIX.

About this Tutorial

This demo explains how to use the MASS high-performance mathematical libraries via the auto-vectorization features supported by XL compilers. The demo is built upon the examples given in a developerWorks article titled “How to improve the performance of programs calling mathematical functions -- Taking advantage of IBM XL C/C++ or XL Fortran compiler auto-vectorization”. Please refer to the article: http://www.ibm.com/developerworks/rational/library/10/improveperformanceprogramsmathfunctions/index.html for detailed explanations on MASS libraries, auto-vectorization, and the Fortran and C source code.

Objectives

- Using IBM XL C/C++/Fortran Compiler for AIX to exploit auto-vectorization through MASS library
- Total time: 15 minutes

Prerequisites

- Basic Unix skills
- Basic command line compilation experience

System Requirements

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

Glossary

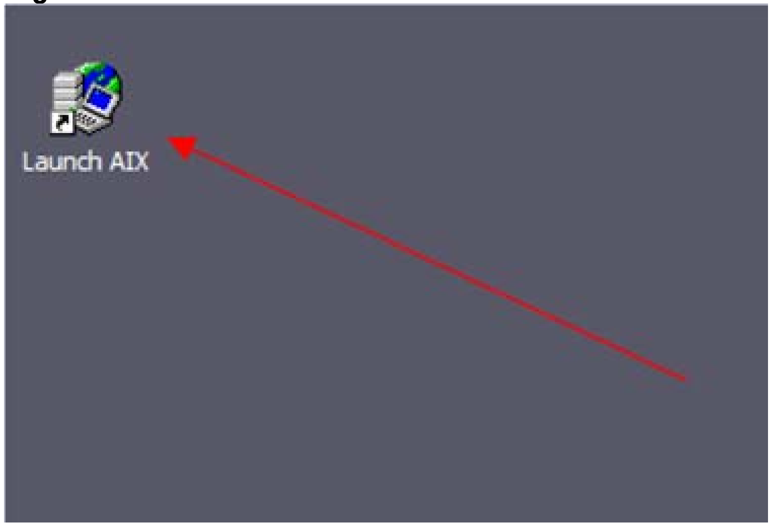
IBM XL C/C++ Compiler: IBM® XL C and C++ compilers offer advanced compiler and optimization technologies and are built on a common code base for easier porting of your applications between platforms. They comply with the latest C/C++ international standards and industry specifications and support a large array of common language features.

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.

Getting Started

Start the Terminal Emulator to AIX System

Figure 1 Get Started

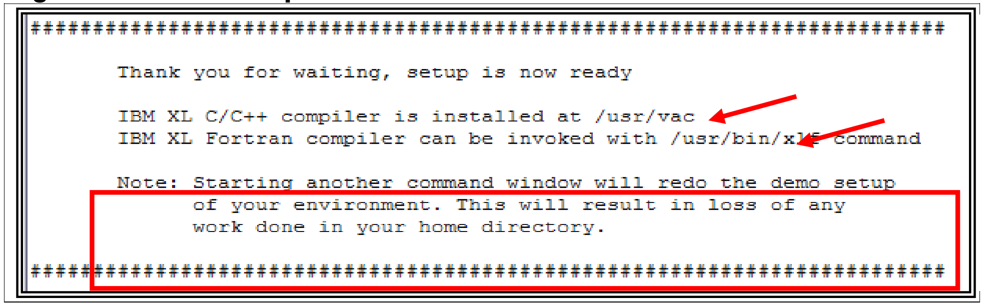


Double click the “Launch AIX” icon on the desktop (See Figure 1) to start the character terminal to AIX system.

Get started with auto-vectorization

Successful login will result with user presented with a menu of demo hosted on the server. Type 9 and press Enter to select the “Taking advantage of auto-vectorization in IBM XL C/C++ and Fortran compilers” demo.

Figure 2 Demo Prepared



```
########################################################################

       Thank you for waiting, setup is now ready

       IBM XL C/C++ compiler is installed at /usr/vac
       IBM XL Fortran compiler can be invoked with /usr/bin/xlf command

       Note: Starting another command window will redo the demo setup
             of your environment. This will result in loss of any
             work done in your home directory.

########################################################################
```

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

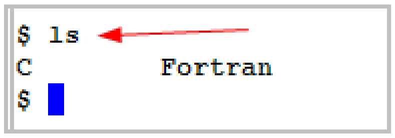
Note: Starting another command window will start the demonstration setup of your environment. This will result in loss of any work done in your home directory. This will impact any progress you have made on demo steps going forward.

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.

The terminal window is now ready for commands. 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



```
$ ls
C         Fortran
$
```

Notice two (2) directories in your home directory.

C	This directory contains C Source code for this demo
Fortran	This directory contains Fortran Source code for this demo

Steps:

The purpose of this demo is to demonstrate the performance gain by enabling auto-vectorization supplied by XL compilers at -O4 level and above. It studies the execution time spent on a discrete Fourier transform (DFT) at the following three compilation options: -qnoopt, -O3 -qnohot and -O4. Auto-vectorizations are enabled only at -O4 level in this example.

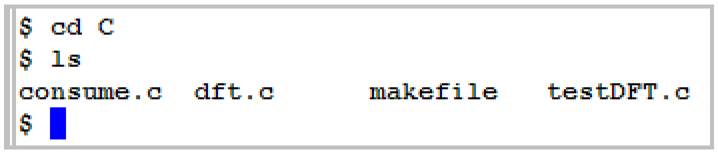
This demo has two sections for C/C++ code example and Fortran code example. You may choose to conduct either or both of them.

Part 1: C/C++ Source code steps to demonstrate the auto-vectorization:

1. Change directory to ‘C’ and list its contents

   Command:
   ```
   cd C
   ls
   ```

Figure 4 Change directory and list 1



```
$ cd C
$ ls
consume.c  dft.c      makefile   testDFT.c
$
```

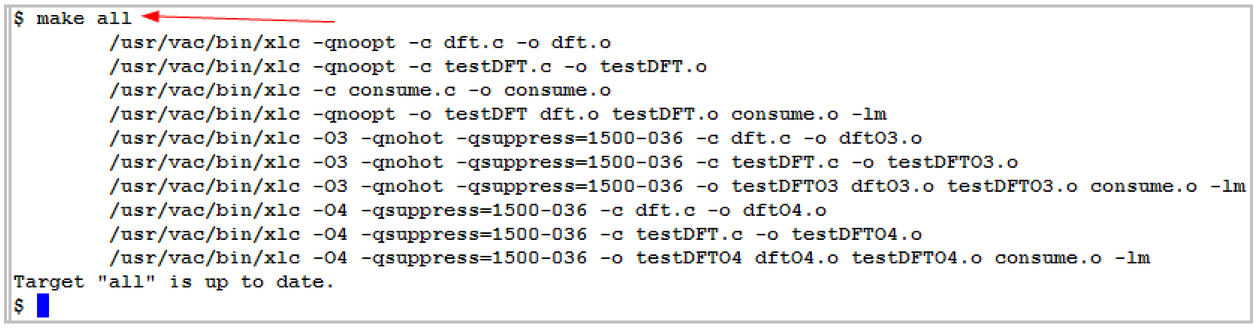
Notice four (4) file in C directory.

consume.c	C source code to be used to prevent IPA from optimizing away the DFT computation. The routine is empty, and is always compiled at noopt.
dft.c	A C source code that defines a DFT routine
testDFT.c	A C source code that tests the execution time of the DFT routine using a specified problem size (2000).
makefile	A makefile to build testDFT at noopt, O3 and O4 levels

2. Type ‘`make all`’ to compile the source to produce three executables testDFT, testDFTO3 and testDFT04

   Command:
   ```
   make all
   ```

Figure 5 make all 1

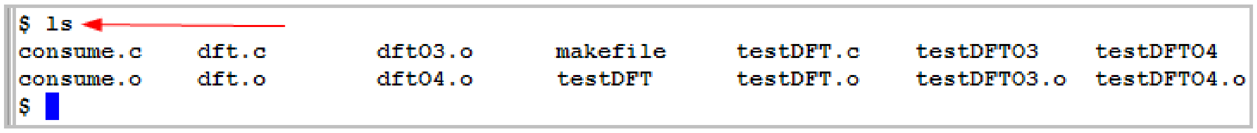


```
$ make all
        /usr/vac/bin/xlc -qnoopt -c dft.c -o dft.o
        /usr/vac/bin/xlc -qnoopt -c testDFT.c -o testDFT.o
        /usr/vac/bin/xlc -c consume.c -o consume.o
        /usr/vac/bin/xlc -qnoopt -o testDFT dft.o testDFT.o consume.o -lm
        /usr/vac/bin/xlc -O3 -qnohot -qsuppress=1500-036 -c dft.c -o dftO3.o
        /usr/vac/bin/xlc -O3 -qnohot -qsuppress=1500-036 -c testDFT.c -o testDFTO3.o
        /usr/vac/bin/xlc -O3 -qnohot -qsuppress=1500-036 -o testDFTO3 dftO3.o testDFTO3.o consume.o -lm
        /usr/vac/bin/xlc -O4 -qsuppress=1500-036 -c dft.c -o dftO4.o
        /usr/vac/bin/xlc -O4 -qsuppress=1500-036 -c testDFT.c -o testDFTO4.o
        /usr/vac/bin/xlc -O4 -qsuppress=1500-036 -o testDFTO4 dftO4.o testDFTO4.o consume.o -lm
Target "all" is up to date.
$
```

The three executable produced are –
1. `testDFT` with compiler flag –qnoopt that is no optimization.
2. `testDFTO3` with flag `-O3` –qnohot that is level 3 optimization and high-order transformations on loops.
3. `testDFTO4` at `-O4` that is level 4 optimization.

Note: Only `testDFTO4` has auto-vectorization enabled.

Figure 6 list compiled files 1

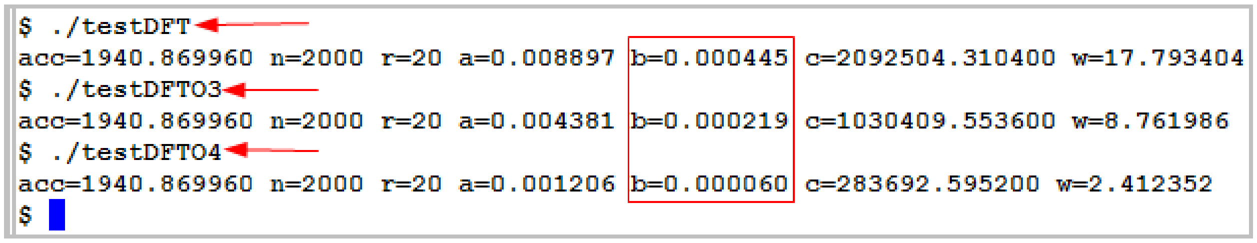


```
$ ls
consume.c     dft.c         dftO3.o       makefile      testDFT.c     testDFTO3     testDFTO4
consume.o     dft.o         dftO4.o       testDFT       testDFT.o     testDFTO3.o   testDFTO4.o
$
```

3. Run the programs and compare the results

   Command:
   ```
   ./testDFT
   ./testDFTO3
   ./testDFTO4
   ```

Figure 7 Run 1



```
$ ./testDFT
acc=1940.869960 n=2000 r=20 a=0.008897 b=0.000445 c=2092504.310400 w=17.793404
$ ./testDFTO3
acc=1940.869960 n=2000 r=20 a=0.004381 b=0.000219 c=1030409.553600 w=8.761986
$ ./testDFTO4
acc=1940.869960 n=2000 r=20 a=0.001206 b=0.000060 c=283692.595200 w=2.412352
$
```

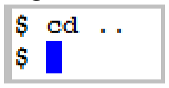
The values printed after b= are the elapsed seconds per element. You can see the time taken for running the program at level 4 (O4) optimization with auto-vectorization enabled is significantly lower than the time taken for the program optimized at level 3 (O3).

(The exact timings you obtain may differ from those in Fig. 8, since they will vary with host processor type (which may change) and processor load.)

4. Change to your home directory

   Command:
   ```
   cd ..
   ```

Figure 8 Change Dir 1



```
$ cd ..
$
```

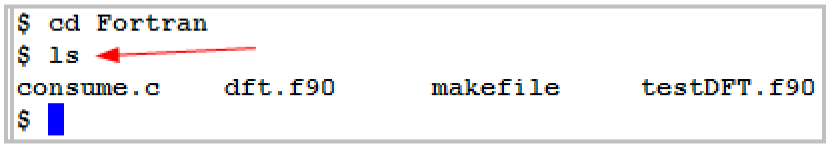
This concludes this tutorial for C/C++ source code part of "Explore Exploring auto-vectorization using MASS library supported in IBM XL C/C++ and Fortran Compilers for AIX".

Part 2: Fortran Source code steps to demonstrate the auto-vectorization:

1. Change directory to 'Fortran' and list its contents

   Command:
   ```
   cd Fortran
   ls
   ```

Figure 9 Change directory and list 2



```
$ cd Fortran
$ ls
consume.c    dft.f90      makefile     testDFT.f90
$
```

Notice four (4) file in Fortran directory.

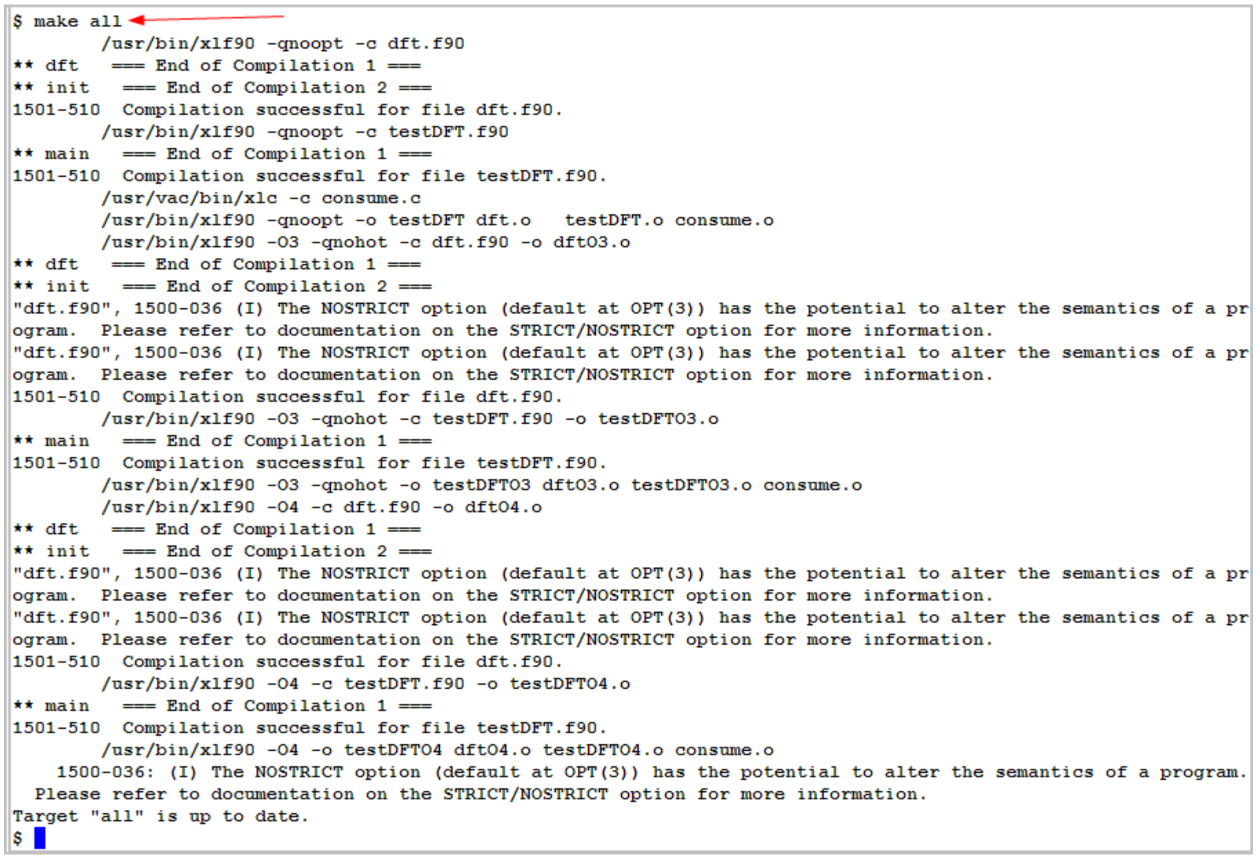
consume.c	C source code to be used to prevent IPA from optimizing away the DFT computation. The routine is empty, and is always compiled at noopt.
dft.f90	A Fortran source code that defines a DFT routine
testDFT.f90	A Fortran source code that tests the execution time on DFT routine using a

	specified problem size (2000).
makefile	A makefile to build testDFT at noopt, O3 and O4 levels

2. Type ‘`make all`’ to compile the source to produce three executables testDFT, testDFTO3 and testDFT04

   Command:
   ```
   make all
   ```

Figure 10 make all 2

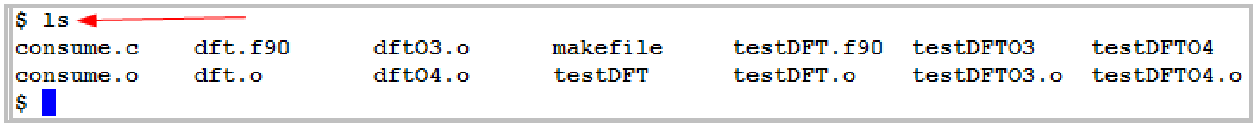


```
$ make all
        /usr/bin/xlf90 -qnoopt -c dft.f90
** dft    === End of Compilation 1 ===
** init    === End of Compilation 2 ===
1501-510  Compilation successful for file dft.f90.
        /usr/bin/xlf90 -qnoopt -c testDFT.f90
** main    === End of Compilation 1 ===
1501-510  Compilation successful for file testDFT.f90.
        /usr/vac/bin/xlc -c consume.c
        /usr/bin/xlf90 -qnoopt -o testDFT dft.o   testDFT.o consume.o
        /usr/bin/xlf90 -O3 -qnohot -c dft.f90 -o dftO3.o
** dft    === End of Compilation 1 ===
** init    === End of Compilation 2 ===
"dft.f90", 1500-036 (I) The NOSTRICT option (default at OPT(3)) has the potential to alter the semantics of a pr
ogram.  Please refer to documentation on the STRICT/NOSTRICT option for more information.
"dft.f90", 1500-036 (I) The NOSTRICT option (default at OPT(3)) has the potential to alter the semantics of a pr
ogram.  Please refer to documentation on the STRICT/NOSTRICT option for more information.
1501-510  Compilation successful for file dft.f90.
        /usr/bin/xlf90 -O3 -qnohot -c testDFT.f90 -o testDFTO3.o
** main    === End of Compilation 1 ===
1501-510  Compilation successful for file testDFT.f90.
        /usr/bin/xlf90 -O3 -qnohot -o testDFTO3 dftO3.o testDFTO3.o consume.o
        /usr/bin/xlf90 -O4 -c dft.f90 -o dftO4.o
** dft    === End of Compilation 1 ===
** init    === End of Compilation 2 ===
"dft.f90", 1500-036 (I) The NOSTRICT option (default at OPT(3)) has the potential to alter the semantics of a pr
ogram.  Please refer to documentation on the STRICT/NOSTRICT option for more information.
"dft.f90", 1500-036 (I) The NOSTRICT option (default at OPT(3)) has the potential to alter the semantics of a pr
ogram.  Please refer to documentation on the STRICT/NOSTRICT option for more information.
1501-510  Compilation successful for file dft.f90.
        /usr/bin/xlf90 -O4 -c testDFT.f90 -o testDFTO4.o
** main    === End of Compilation 1 ===
1501-510  Compilation successful for file testDFT.f90.
        /usr/bin/xlf90 -O4 -o testDFTO4 dftO4.o testDFTO4.o consume.o
    1500-036: (I) The NOSTRICT option (default at OPT(3)) has the potential to alter the semantics of a program.
  Please refer to documentation on the STRICT/NOSTRICT option for more information.
Target "all" is up to date.
$ 
```

The three executable produced are –
`testDFT` with compiler flag –qnoopt that is no optimization.
`testDFTO3` with flag `-O3` –qnohot that is level 3 optimization and high-order transformations on loops.
`testDFTO4` at -O4 that is level 4 optimization.

Note: Only `testDFTO4` has auto-vectorization enabled.

Figure 11 list compiled files 2

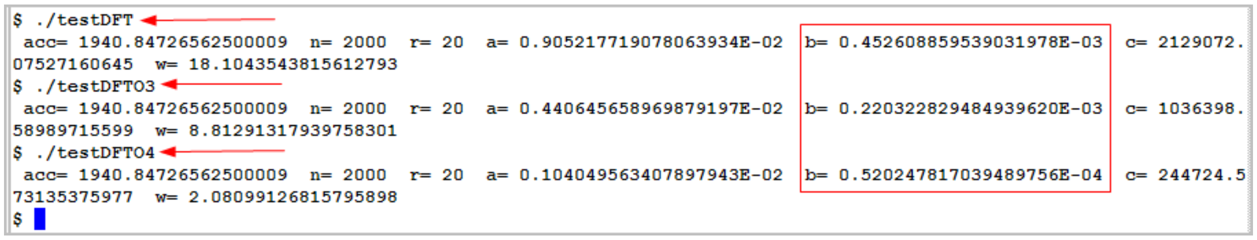


```
$ ls
consume.c    dft.f90     dft03.o     makefile    testDFT.f90   testDFT03     testDFT04
consume.o    dft.o       dft04.o     testDFT     testDFT.o     testDFT03.o   testDFT04.o
$ 
```

3. Run the programs and compare the results

   Command:
   ```
   ./testDFT
   ./testDFT03
   ./testDFT04
   ```

Figure 12 Run 2



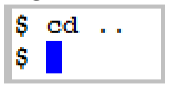
```
$ ./testDFT
 acc= 1940.84726562500009  n= 2000  r= 20  a= 0.905217719078063934E-02  b= 0.452608859539031978E-03  c= 2129072.
07527160645  w= 18.1043543815612793
$ ./testDFT03
 acc= 1940.84726562500009  n= 2000  r= 20  a= 0.440645658969879197E-02  b= 0.220322829484939620E-03  c= 1036398.
58989715599  w= 8.81291317939758301
$ ./testDFT04
 acc= 1940.84726562500009  n= 2000  r= 20  a= 0.104049563407897943E-02  b= 0.520247817039489756E-04  c= 244724.5
73135375977  w= 2.08099126815795898
$ 
```

The values printed after b= are the elapsed seconds per element. You can see the time taken for running the program compiled at level 4 (O4) optimization with auto-vectorization enabled is significantly lower than the time taken for the program optimized at level 3 (O3). (The exact timings you obtain may differ from those in Fig. 8, since they will vary with host processor type (which may change) and processor load.)

4. Change to your home directory

   Command:
   ```
   cd ..
   ```

Figure 13 Change Dir 2



```
$ cd ..
$ 
```

This concludes this tutorial for Fortran source code part of “Explore Exploring auto-vectorization using MASS library supported in IBM XL C/C++ and Fortran Compilers for AIX”.

What you have learned

In this exercise you learnt how to:
- Use IBM XL C/C++ and Fortran compiler for AIX to build source code.
- Use optimization flags to optimize the program

Conclusion

This tutorial demonstrated how to exploit auto-vectorization with XL C/C++ and XL Fortran for AIX compilers. Demonstration also presents why you should optimize you code at higher optimization levels to achieve better performance with advance features in IBM XL compilers.

Trademarks

Resources

XL C/C++ for AIX, Library :

http://www.ibm.com/software/awdtools/xlcpp/aix/library/

XL Fortran for AIX, Library :

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

Community Cafe :

http://www.ibm.com/software/rational/cafe/community/ccpp

DeveloperWorks article:

How to improve the performance of programs calling mathematical functions -- Taking advantage of IBM XL C/C++ or XL Fortran compiler auto-vectorization
http://www.ibm.com/developerworks/rational/library/10/improveperformanceprogramsmathfunctions/index.html