IBM XL Fortran for AIX

Performance Benefits of Using the F2008 CONTIGUOUS Attribute

By: Bardia Mahjour

Level: Introductory

May 2012

Contents

Before you start

About this Tutorial

This tutorial demonstrates the support for the Fortran 2008 CONTIGUOUS attribute in the XL Fortran compiler and how it can help improve the runtime performance of your Fortran applications. The CONTIGUOUS attribute allows Fortran programmers to give assertions to the compiler about the contiguity of pointer arrays. The CONTIGUOUS attribute can also be used to instruct the compiler to ensure contiguity of assumed-shape arrays upon argument association. The XL Fortran compiler uses these F2008 semantics to generate more optimized code when possible.

In this tutorial, a few examples are provided to demonstrate the use of CONTIGUOUS attribute. In addition, the runtime performance of these examples are measured to illustrate the advantage of using the CONTIGUOUS attribute in various contexts. In order to exclude the effects of other optimizations from these measurements, these examples will be compiled without enabling optimization options such as -O2, -O3, or higher.

Objectives

- Demonstrate and measure potential runtime performance improvement for Fortran applications containing the CONTIGUOUS attribute, using the the IBM XL Fortran compiler.

- Total time: 20 minutes

Prerequisites

- Basic Unix skills
- Basic Source code compile and build experience
- Basic knowledge of Fortran Language

System Requirements

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

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.

Getting Started

Start the Terminal Emulator to AIX System

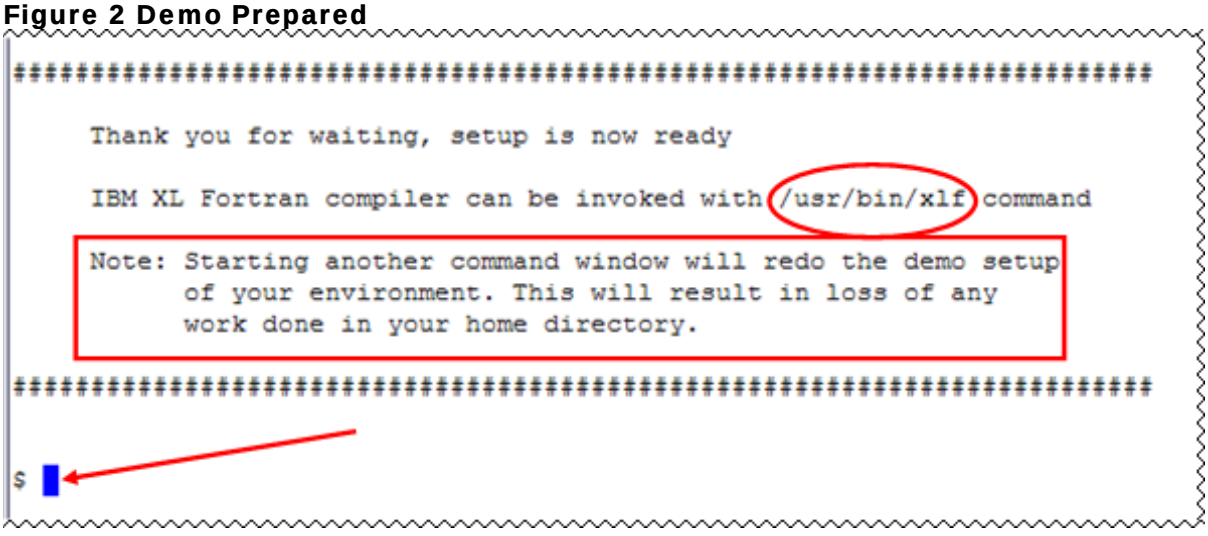
Figure 1 Get Started

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 Fortran Compiler

A successful login will result with the user presented with a menu of the demos hosted on the server. Type 15 and press Enter to select the “Using the F2008 CONTIGUOUS” demo.

Figure 2 Demo Prepared



On the terminal window you will see important information and the directory path to the compiler invocation command (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.

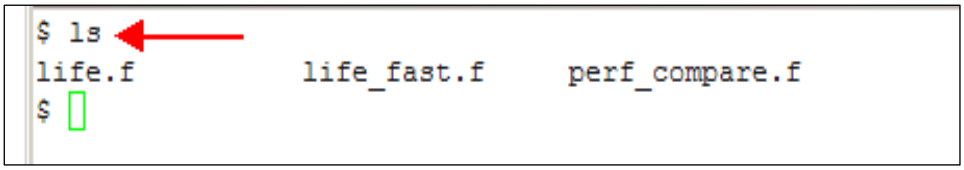
This demo does not require more than one terminal window. However, if you prefer more than one terminal window then you may open them before going forward.

The terminal window is now ready for commands. Your home directory contains the source code for three Fortran programs. Type the ls command to see the directory content (See Figure 3 Contents).

Command:

```
ls
```

Figure 3 Contents



These Fortran programs comply with the Fortran 2008 standard.

The file “life.f” contains an implementation of a simplified version of Conway's Game of Life. In this implementation the life matrix is represented as a logical pointer array of rank 2.

The file “life_fast.f” contains the same implmenentation as “life.f”, except that it gives the life matrix the CONTIGUOUS attribute.

We will be measuring the running time of each of these implementations and compare them to see how much performance improvement can be achieved with the CONTIGUOUS attribute.

The file “perf_compare.f” contains a contrived example, where the use of CONTIGUOUS attribute on an assumed-shape array is being illustrated.

Three programs are provided here:

life.f	An implementation of the Game Of Life without the use of CONTIGUOUS attribute
life_fast.f	Same as life.f except that the life matrixes are declared with the CONTIGUOUS attribute.
perf_compare.f	A self-contained program for measuring performance improvement when using CONTIGUOUS attribute on an assumed-shape array that is argument associated with an explicit-shape array.

Note: The data structures and algorithms used in these samples do not necessarily represent the most efficient or optimal implementation. Rather they are used to illustrate the optimization and performance improvement that can be achived by using CONTIGUOUS attribute in appropriate parts of the code.

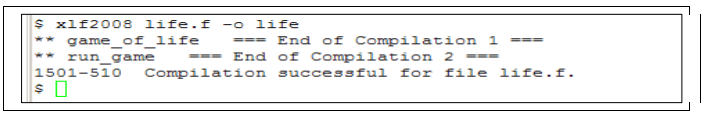
Note: Depending on the workload and the state of the host machine, the actual measurement values you will see may be slightly different from the values shown in this document.

Steps:

1. Compile and run life.f program. See figure 4

   Command:
   ```
   xlf2008 life.f -o life
   ```

   Figure 4 Compile life.f

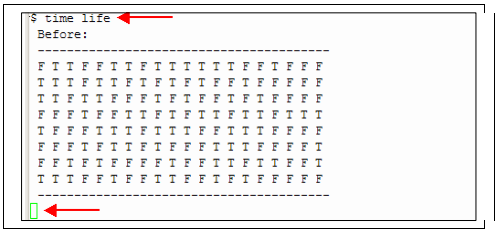


   Note: You could use compiler invocations other than the xlf2008. For example you could use xlf2003. If you use an invocation other than xlf2008 or xlf2003, you will need to specify the -qxlf2003=polymorphic option in order to be able to compile this program.

   Run the program life and measure the time it takes to complete (See Figure 5 Run life).

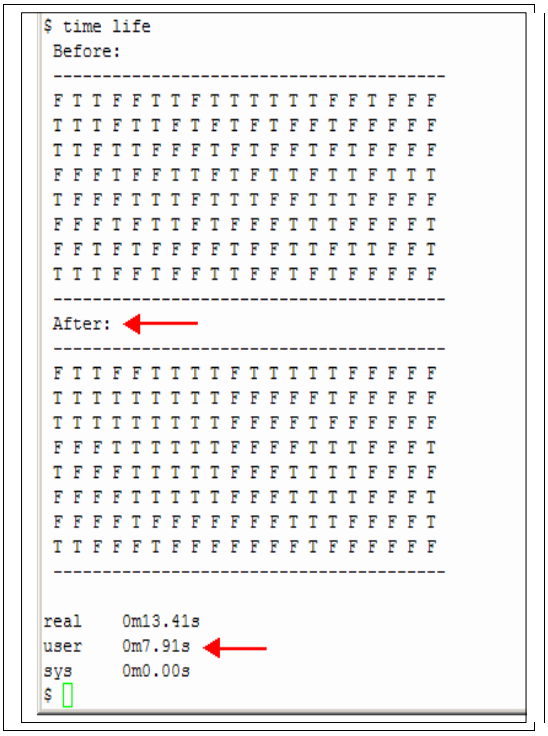
   Command:
   ```
   time life
   ```

   Figure 5 Run life



   The program may take about half a minute or so to complete. Please wait until it finishe. Once it is finished, you'll see the time it took to execute it as show in Figure 6.

Figure 6 running times for life



2. Compile and run life_fast.f. (See Figure 7 compile life_fast.f)

Note: life_fast.f declares the life and life_copy matrixes with the CONTIGUOUS attribute:

Program life_fast.f

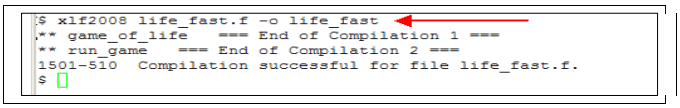
```
…
…
        type GOL
         logical, pointer, contiguous :: life(:,:)
         logical, pointer, contiguous :: life_copy(:,:)

         contains
…
…
```

Compile life_fast.f program.

Command:

```
xlf2008 life_fast.f -o life_fast
```

Figure 7 compile life_fast.f

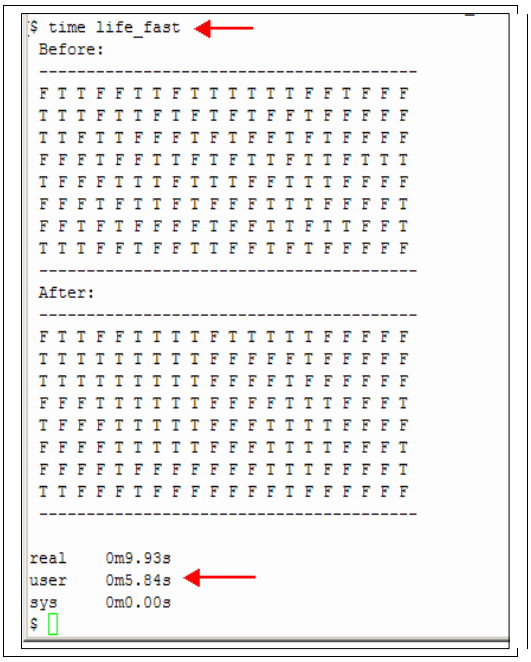


Run life_fast program and measure the time it takes to complete (See Figure 8 Run life_fast).

Command:

```
time life_fast
```

Figure 8 Run life_fast



Notice that this version of the program finished executing about 2 seconds earlier than the program compiled in step 1. That shows a performance improvement of about 26% for this particular application.

This demonstrates the XL Fortran Compiler's ability in taking advantage of compile-time known contiguity of the matrixes to generate more efficient code.

In both “life.f” and “life_fast.f”, the arrays *life(:,:)* and *life_copy(:,:)* get argument associated with explicit shape arrays. For example:

life_fast.f : Passing life and life_copy as arguments.

```
  …
  …
          if (equal(gol%life_copy, gol%life)) then
             print *, "equilibrium! (after", current_generation, "tries)"
             stop
```

```
          end if
  …
  …
        function equal(a, b)
          logical :: equal
          logical, intent(in) :: a(num_rows,num_cols), b(num_rows,num_cols)
          equal = .not. any(a .neqv. b)
        end
```

In the case of “life.f”, the compiler needs to generate code to check whether the actual argument is contiguous or not. If the argument is not contiguous, it has to be “copied-in” to a contiguous temporary array, so that the contiguous temporary can be associated with the explicit shape array. Depending on the INTENT of the argument, a “copy-out” may also be necessary. This is necessary in order to conform to the Fortran language rules.

In the case of “life_fast.f”, however, the CONTIGUOUS attribute acts as an assertion to the compiler that *life(:,:)* and *life_copy(:,:)* are always contiguous. As a result, XL Fortran compiler stops generating extra code to check whether copy-in/copy-out is necessary.

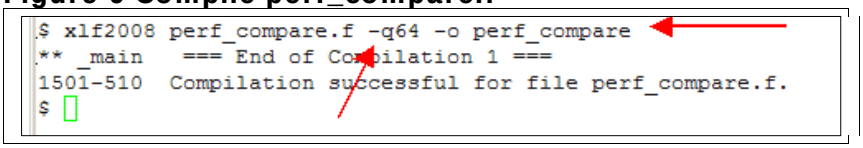
Since *life* and *life_copy* get argument associated with explicit shape arrays many times during the execution of this program, the benefit of not having to execute the contiguity checking code adds up to a significant amount.

3. Compile and run perf_compare.f:

Command:

```
xlf2008 perf_compare.f -q64 -o perf_compare
```

Figure 9 Compile perf_compare.f



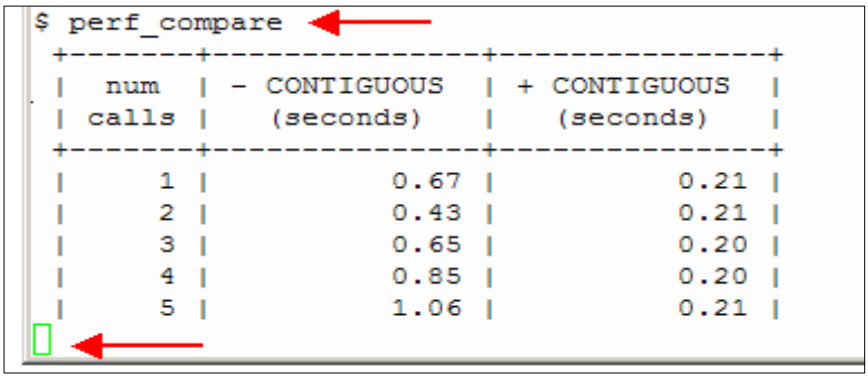
Note: You could use compiler invocations other than the xlf2008. For example you could use xlf2003, xlf95, etc.

Note: You need to specify -q64 when compiling this program because the program will allocate large arrays. If you compile this program in 32-bit mode, the application may fail to run due to lack of available address space.

Run the output program (See Figure 10 Run perf_compare)

Command:

```
perf_compare
```



```
$ perf_compare
+-------+---------------+---------------+
|   num   | - CONTIGUOUS   | + CONTIGUOUS   |
| calls |    (seconds)    |    (seconds)    |
+-------+---------------+---------------+
|      1 |           0.67 |           0.21 |
|      2 |           0.43 |           0.21 |
|      3 |           0.65 |           0.20 |
|      4 |           0.85 |           0.20 |
|      5 |           1.06 |           0.21 |
```

Figure 10: Run perf_compare

Note: It may take a while for the program to finish executing.

The program completes 10 trials. In each iteration a non-contiguous pointer array, *ptr,* is passed to subroutines *without_contig* and *with_contig.* The time it takes to execute each call is measured and displayed in tabular format. (See Figure 11)

Program perf_compare: iterations:

```
…
      ptr => t(1:DIM_SIZE:STRIDE, :)
…
      do i = 1, N_TRIALS, 1
…
         call without_contig(ptr, i)
…
         call with_contig(ptr, i)
…
      end do
…
…
```

The implementations of *without_contig* and *with_contig* are the same, except that in *with_contig* the assumed-shape dummy argument *arr* is marked with the CONTIGUOUS attribute. Each of these subroutines call a helper subroutine *sub* inside a loop to increment the first column of the input array by one.

Program perf_compare: without_contig and helper subroutine

```
...
      subroutine without_contig(arr, ncalls)
        integer :: arr(:,:)
        integer :: ncalls
        integer :: i
        do i=1, ncalls, 1
           call sub(arr)
        end do
      end subroutine

      subroutine sub(arg)
        integer :: arg(DIM_SIZE/STRIDE, DIM_SIZE)
        arg(:,1) = arg(:,1) + 1
      end subroutine
```

```
   ...
```

In the case of the subroutine *without_contig*, the assumed-shape argument *arr* is a non-contiguous array. As a result when subroutine *sub* is called, *arr* has to be copied-in and copied-out for each call to *sub.*

As the number of times *sub* gets called increases, the number of times *arr* needs to get copied in and out increases. Note that the example is written such that the number of times *sub* gets called is incremented in each trial. (ie *sub* gets called inside *without_contig* once in trial number 1, twice in trial number 2, and so on)

In the case of the subroutine *with_contig*, however, the XL Fortran Compiler uses the CONTIGUOUS attribute to guarantee that *arr* is a contiguous array, so the copy-ins and copy-outs for the call to *sub* are not necessary. The XL Fortran compiler ensures the contiguity of *arr* by performing a copy-in and copy-out when associating *ptr* with *arr*. This way, there is only one copy-in and one copy-out operation regardless of how many times *sub* gets called.

Figure 11: Complete output from perf_compare

```
$ perf_compare
+-------+---------------+---------------+
|  num  | - CONTIGUOUS  | + CONTIGUOUS  |
| calls |   (seconds)   |   (seconds)   |
+-------+---------------+---------------+
|     1 |          0.67 |          0.21 |
|     2 |          0.43 |          0.21 |
|     3 |          0.65 |          0.20 |
|     4 |          0.85 |          0.20 |
|     5 |          1.06 |          0.21 |
|     6 |          1.30 |          0.20 |
|     7 |          1.49 |          0.21 |
|     8 |          1.72 |          0.21 |
|     9 |          1.91 |          0.21 |
|    10 |          2.14 |          0.22 |
+-------+---------------+---------------+
$
```

As indicated in Figure 11, the time it takes to execute *without_contig* increases as the number of calls to *sub* increases. However, the time it takes to execute *with_contig* remains largely flat as the number of calls to *sub* increases.

What you have learned

In this exercise you learned how to:

- Use the IBM XL Fortran for AIX compiler to compile programs.
- Use the Fortran 2008 CONTIGUOUS attribute on pointer arrays that get argument associated with explicit-shape arrays.
- Use the Fortran 2008 CONTIGUOUS attribute on assumed-shape arrays that get argument associated with explicit-shape arrays.

Conclusion

This concludes the tutorial on using the IBM XL Fortran compiler. This tutorial has shown how to properly use the CONTIGUOUS attribute to help improve runtime performance of your application.

Trademarks

Resources

Learn

Overview of the IBM XL C/C++ and XL Fortran Compiler Family
http://www.ibm.com/support/docview.wss?uid=swg27005175

Getting Started with XL Fortran for AIX, V14.1
http://www.ibm.com/support/docview.wss?uid=swg27024216

Compiler Reference - XL Fortran for AIX, V14.1
http://www.ibm.com/support/docview.wss?uid=swg27024215

Language Reference - XL Fortran for AIX, V14.1
http://www.ibm.com/support/docview.wss?uid=swg27024218

Optimization:

Optimization and Programming Guide - XL Fortran for AIX, V14.1
http://www.ibm.com/support/docview.wss?uid=swg27024219

Code Optimization with the IBM XL Compilers
http://www.ibm.com/support/docview.wss?uid=swg27005174