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

Explore Optimization Opportunities with XML Transformation Reports in IBM XL C/C++ and XL Fortran for AIX Compilers

By: Yong Du
Kobi Vinayagamoorthy
Kevin Yuen
Yan Zhang

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 XL Fortran for AIX® compilers.

About this Tutorial

This demo explains the benefits of using the -qlistfmt=xml=* option introduced in the IBM XL C/C++ V11.1/V12.1 and XL Fortran V13.1/V14.1 compilers. It uses the XL C/C++ V11.1 and XL Fortran V13.1 compilers to generate the XML report for the explanation, but it works with the XL C/C++ V12.1 and XL Fortran V14.1 compilers as well. There are minor changes in the generated XML report, such as compiler version numbers and the additional column information in the inline optimization table.

This option is useful when the user wants to explore optimization opportunities that the XL C/C++/Fortran compilers may provide, by going through the XML reports generated by the compiler through a web browser. Compared with textual listing files generated by previous releases of XL C/C++/Fortran compilers, the XML reports, supported by predefined or customized XML style sheets, present compiler optimization messages in a more readable way; and in some cases, produce messages that are not available in listing files.

Objectives

- Using IBM XL C/C++/Fortran for AIX compilers with the “-qlistfmt=xml=*” option to explore optimization opportunities by looking at the XML reports generated

- Total time: 45 minutes

Prerequisites

- Basic AIX/UNIX skills
- Basic command line compilation experience

System Requirements

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

Glossary

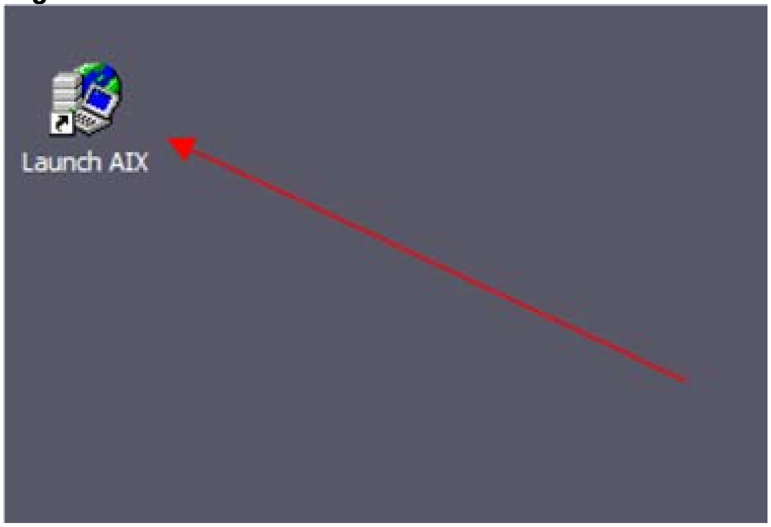
IBM XL C/C++ compiler: IBM XL C and C++ compilers offer advanced compiler and optimization technologies and are built on a common code base for easier porting of your applications between platforms. They comply with the latest C/C++ international standards and industry specifications and support a large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

IBM XL Fortran compiler: 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

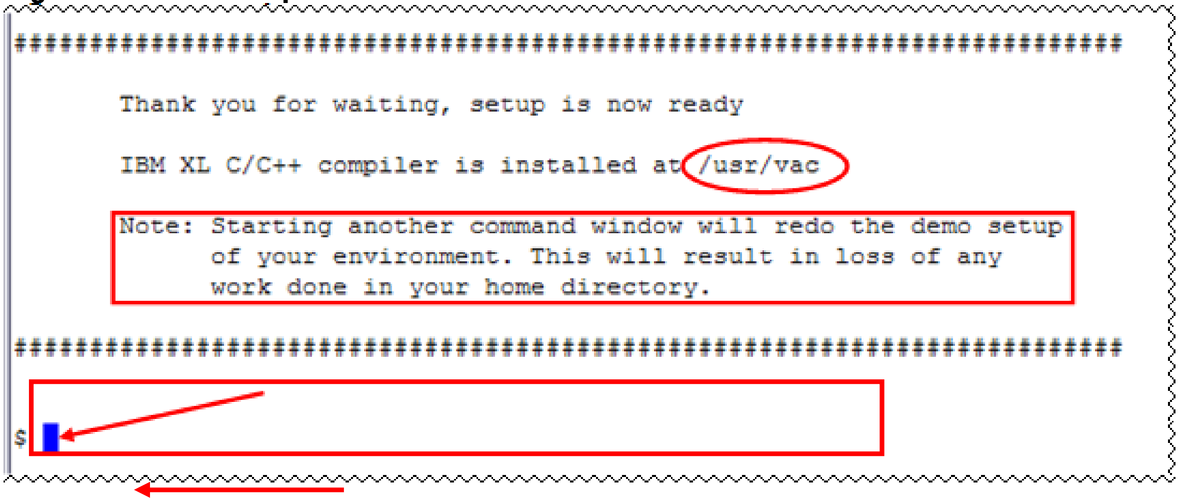


Double click the “Launch AIX” icon on the desktop (See Figure 1) to start the character terminal to AIX system.

Get Started with XML Reports

Successful login will result with user presented with a menu of demo hosted on the server. Type 8 and press Enter to select the “Explore Optimization Opportunities with XML Transformation Reports” demo.

Figure 2 Demo Prepared



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

       Thank you for waiting, setup is now ready

       IBM XL C/C++ compiler is installed at /usr/vac

       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 *oval red*).

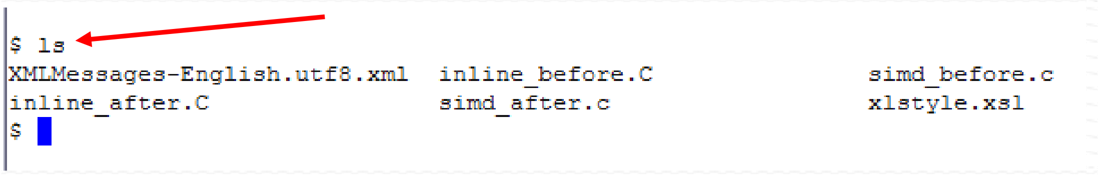
> **Note:** Starting another command window will start the demonstration setup of your environment. This will result in loss of any work done in your home directory. This will impact any progress you have made on demo steps going forward.
>
> This demo does not require more than one terminal window. 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



```
$ ls
XMLMessages-English.utf8.xml  inline_before.C           simd_before.c
inline_after.C                simd_after.c              xlstyle.xsl
$
```

Notice six (6) files under your home directory.

xlstyle.xsl	The XML style sheet used to display the generated XML reports in browsers
XMLMessages-English.utf8.xml	The XML reports message catalogue
inline_before.C	A C++ program to demonstrate how the XML feature reports inline optimization opportunities
inline_after.C	A C++ program to demonstrate how the XML feature reports inline optimization opportunities (after the suggested opportunity is adopted)
simd_before.c	A C program to demonstrate how the XML feature reports loop transformation (SIMD) opportunities
simd_after.c	A C program to demonstrate how the XML feature reports loop transformation (SIMD) opportunities (after the suggested opportunity is adopted)

Steps:

It is possible to generate XML optimization reports for XL C/C++ version 11.1/12.1 and XL Fortran version 13.1/14.1 compilers. However, in the following examples we will use XL C/C++ compiler version 11.1. There are two examples: one to demonstrate the inline optimization XML message, and the other the loop transformation XML message. We will go through these two examples separately.

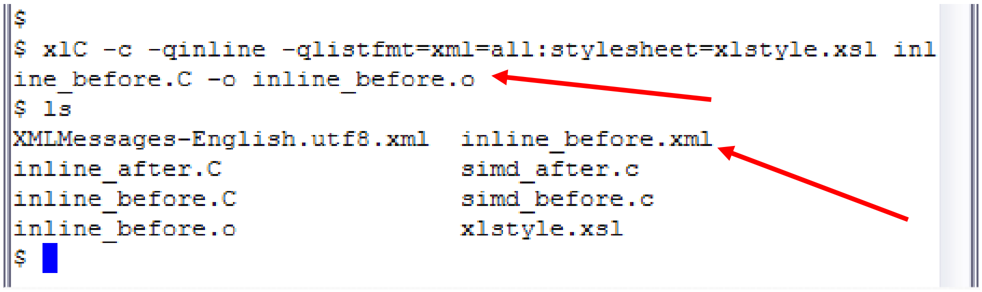
Part 1: Steps to demonstrate the inline optimization XML message:

1. Compile `inline_before.C` with `–qinline` and `–qlistfmt` options

   Command:

```
xlC -c -qinline -qlistfmt=xml=all:stylesheet=xlstyle.xsl \
inline_before.C -o inline_before.o
```

Figure 4 Compile 1



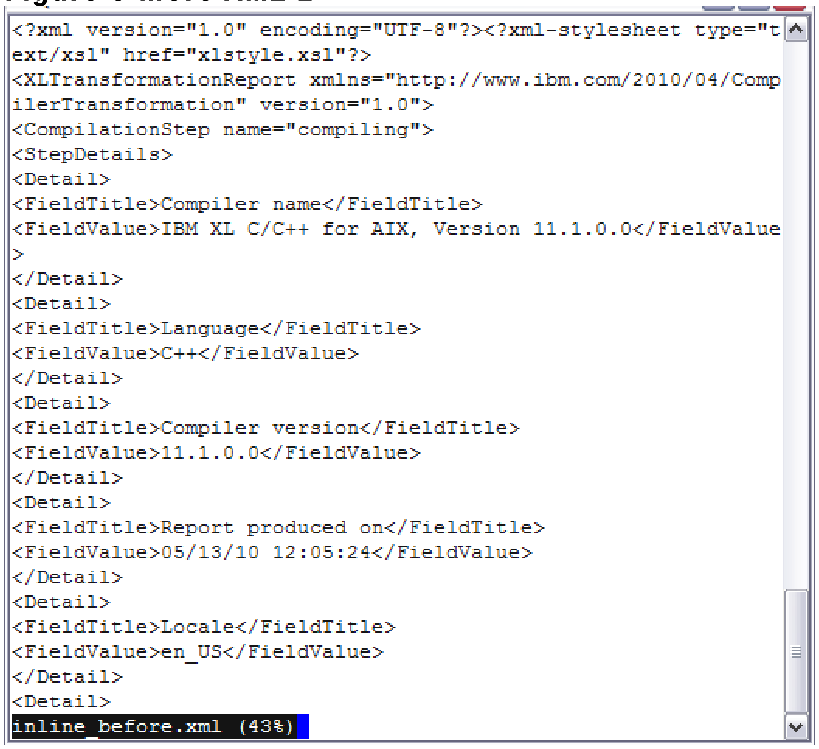
```
$
$ xlC -c -qinline -qlistfmt=xml=all:stylesheet=xlstyle.xsl inl
ine_before.C -o inline_before.o
$ ls
XMLMessages-English.utf8.xml  inline_before.xml
inline_after.C                simd_after.c
inline_before.C               simd_before.c
inline_before.o               xlstyle.xsl
$
```

2. View the `inline_before.xml` using `more` command

   Command:

   ```
   more inline_before.xml
   ```

Figure 5 more XML 1

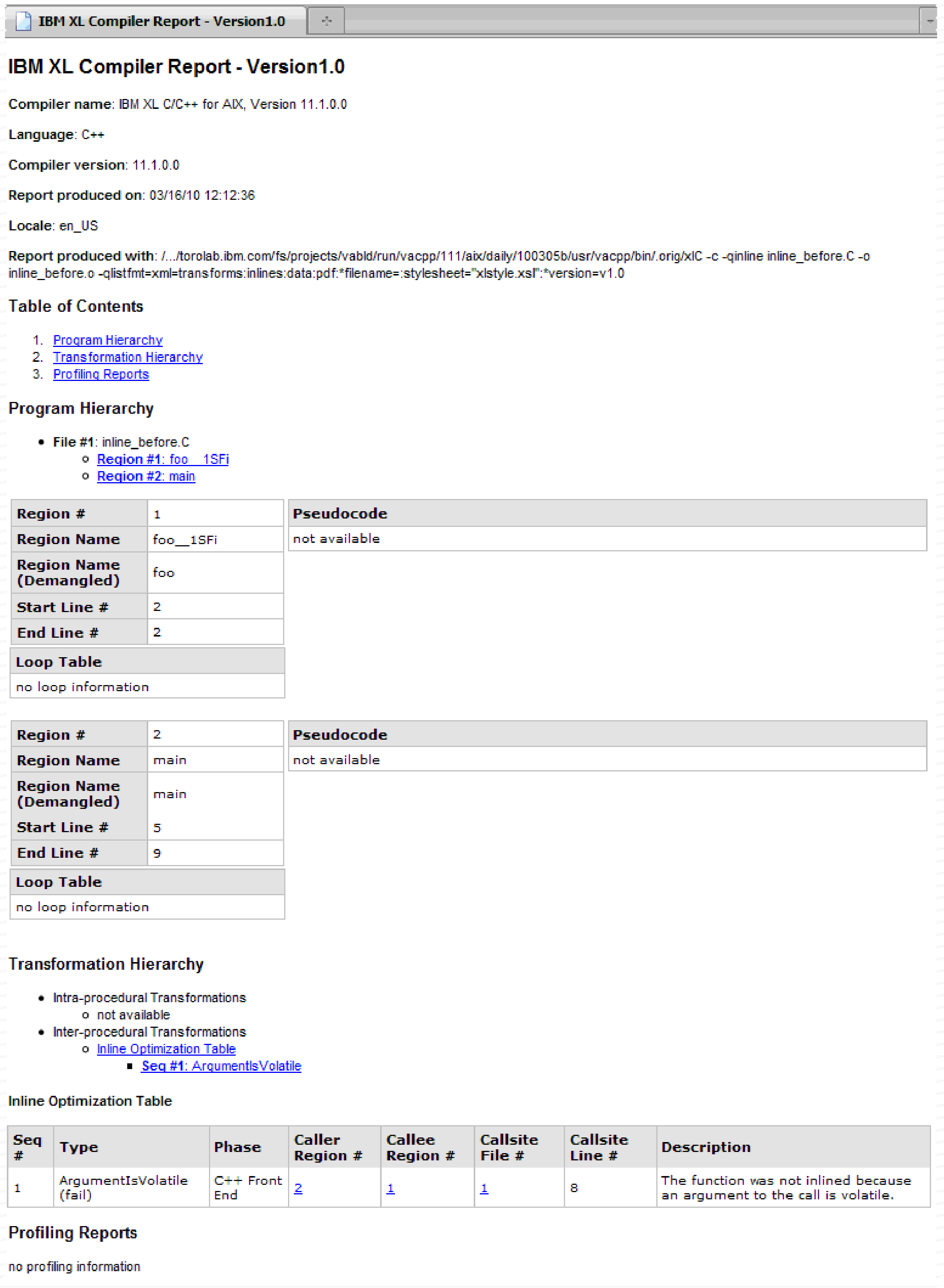


```
<?xml version="1.0" encoding="UTF-8"?><?xml-stylesheet type="t
ext/xsl" href="xlstyle.xsl"?>
<XLTransformationReport xmlns="http://www.ibm.com/2010/04/Comp
ilerTransformation" version="1.0">
<CompilationStep name="compiling">
<StepDetails>
<Detail>
<FieldTitle>Compiler name</FieldTitle>
<FieldValue>IBM XL C/C++ for AIX, Version 11.1.0.0</FieldValue
>
</Detail>
<Detail>
<FieldTitle>Language</FieldTitle>
<FieldValue>C++</FieldValue>
</Detail>
<Detail>
<FieldTitle>Compiler version</FieldTitle>
<FieldValue>11.1.0.0</FieldValue>
</Detail>
<Detail>
<FieldTitle>Report produced on</FieldTitle>
<FieldValue>05/13/10 12:05:24</FieldValue>
</Detail>
<Detail>
<FieldTitle>Locale</FieldTitle>
<FieldValue>en_US</FieldValue>
</Detail>
<Detail>
inline_before.xml (43%)
```

Notice the XML document that is generated. The Sandbox is limited and thus is not able to launch this file in a web browser. In your own environment, you should be able to view this report, formatted for easier readability, in a browser as shown below.

The generated XML report, as shown in a browser (using the provided style sheet `xlstyle.xsl`), shows the types of optimizations that the compiler has or has not performed, and why. For this example, the XML report gives the 'ArgumentIsVolatile' message (towards the bottom of Figure 6, and states that “function was not inlined because an argument to the call is volatile”; and this function call is on line 8.

Figure 6 XML Report 1



IBM XL Compiler Report - Version1.0

Compiler name: IBM XL C/C++ for AIX, Version 11.1.0.0

Language: C++

Compiler version: 11.1.0.0

Report produced on: 03/16/10 12:12:36

Locale: en_US

Report produced with: /.../torolab.ibm.com/fs/projects/vabld/run/vacpp/111/aix/daily/100305b/usr/vacpp/bin/.orig/xlC -c -qinline inline_before.C -o inline_before.o -qlistfmt=xml=transforms:inlines:data:pdf:*filename=:stylesheet="xlstyle.xsl":*version=v1.0

Table of Contents

Program Hierarchy

- **File #1**: inline_before.C
  - Region #1: foo__1SFi
  - Region #2: main

Region #	1
Region Name	foo__1SFi
Region Name (Demangled)	foo
Start Line #	2
End Line #	2

Loop Table
no loop information

Pseudocode
not available

Region #	2
Region Name	main
Region Name (Demangled)	main
Start Line #	5
End Line #	9

Loop Table
no loop information

Pseudocode
not available

Transformation Hierarchy

- Intra-procedural Transformations
  - not available
- Inter-procedural Transformations
  - Inline Optimization Table
    - Seq #1: ArgumentIsVolatile

Inline Optimization Table

Seq #	Type	Phase	Caller Region #	Callee Region #	Callsite File #	Callsite Line #	Description
1	ArgumentIsVolatile (fail)	C++ Front End	2	1	1	8	The function was not inlined because an argument to the call is volatile.

Profiling Reports

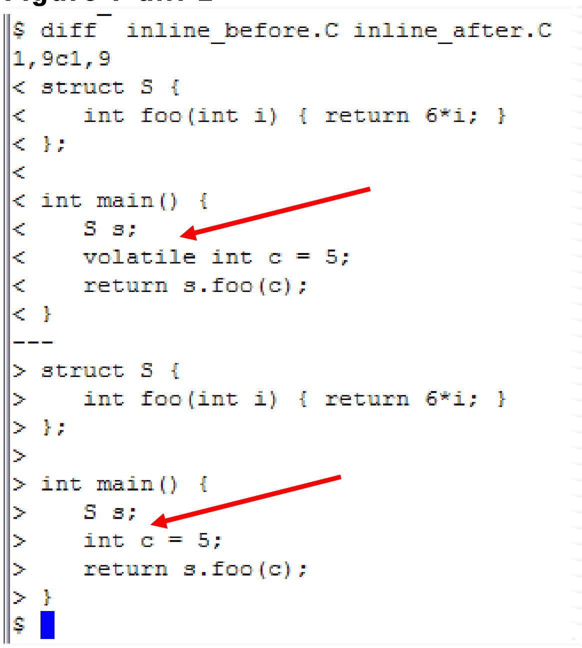
no profiling information

3. Look at the difference between inline_before.C and inline_after.C using diff command.

   Command:

   ```
   diff inline_before.C inline_after.C
   ```

Figure 7 diff 1



```
$ diff  inline_before.C inline_after.C
1,9c1,9
< struct S {
<    int foo(int i) { return 6*i; }
< };
<
< int main() {
<    S s;
<    volatile int c = 5;
<    return s.foo(c);
< }
---
> struct S {
>    int foo(int i) { return 6*i; }
> };
>
> int main() {
>    S s;
>    int c = 5;
>    return s.foo(c);
> }
$
```

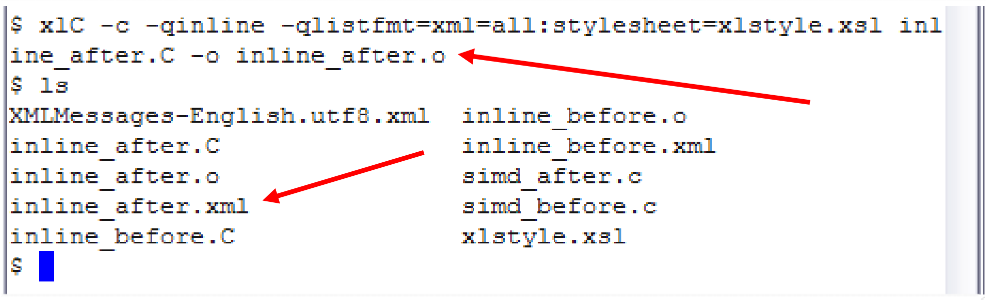
Notice the source difference where integer variable ‘c’, which is passed to function ‘foo’ on line 8 in inline_before.C, is declared as ‘volatile’ on line 7. Where as in file inline_after.C variable ‘c’ does not have a ‘volatile’ type qualifier.

4. Compile inline_after.C with –qinline and –qlistfmt options

   Command:

   ```
   xlC -c -qinline -qlistfmt=xml=all:stylesheet=xlstyle.xsl \
   inline_after.C -o inline_after.o
   ```

Figure 8 Compile 2

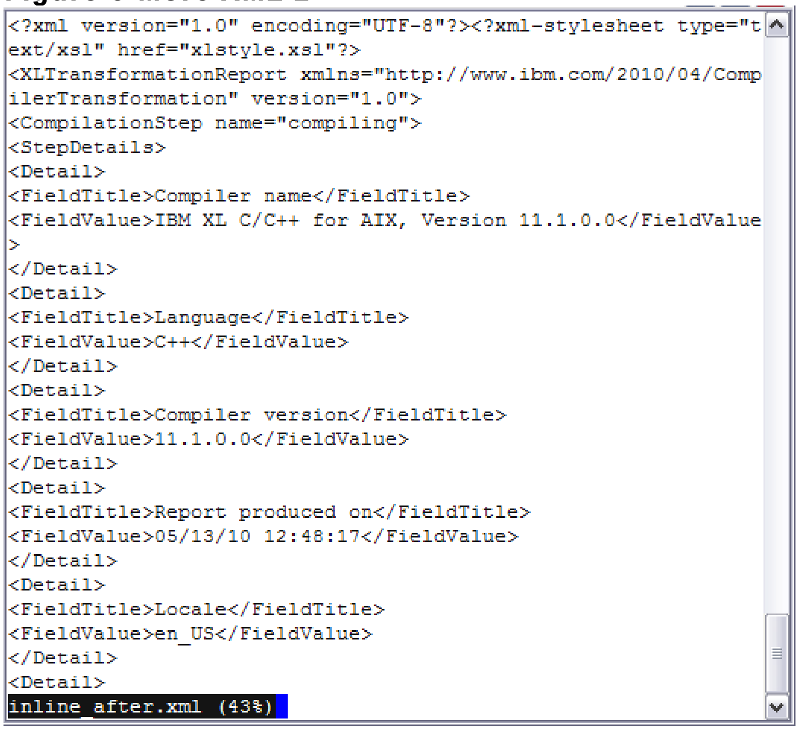


```
$ xlC -c -qinline -qlistfmt=xml=all:stylesheet=xlstyle.xsl inl
ine_after.C -o inline_after.o
$ ls
XMLMessages-English.utf8.xml  inline_before.o
inline_after.C                inline_before.xml
inline_after.o                simd_after.c
inline_after.xml              simd_before.c
inline_before.C               xlstyle.xsl
$
```

5. View the `inline_after.xml` using `more` command

   Command:
   ```
   more inline_after.xml
   ```

Figure 9 More XML 2

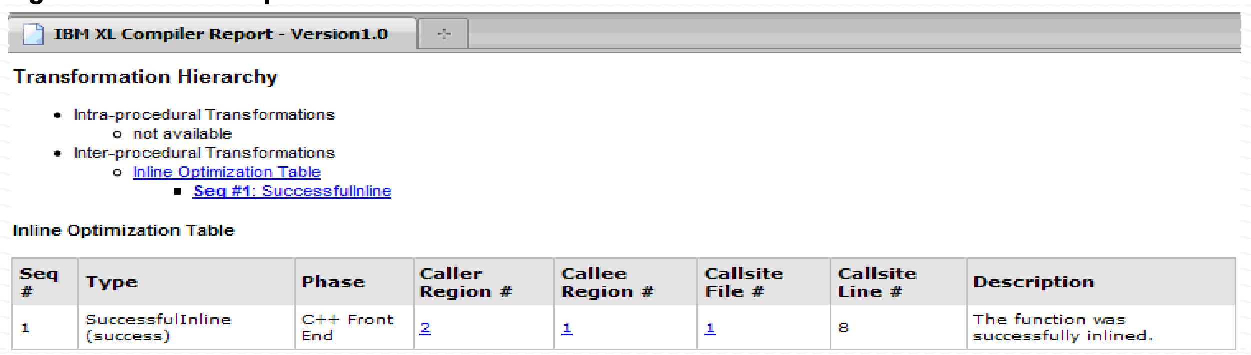


```
<?xml version="1.0" encoding="UTF-8"?><?xml-stylesheet type="t
ext/xsl" href="xlstyle.xsl"?>
<XLTransformationReport xmlns="http://www.ibm.com/2010/04/Comp
ilerTransformation" version="1.0">
<CompilationStep name="compiling">
<StepDetails>
<Detail>
<FieldTitle>Compiler name</FieldTitle>
<FieldValue>IBM XL C/C++ for AIX, Version 11.1.0.0</FieldValue
>
</Detail>
<Detail>
<FieldTitle>Language</FieldTitle>
<FieldValue>C++</FieldValue>
</Detail>
<Detail>
<FieldTitle>Compiler version</FieldTitle>
<FieldValue>11.1.0.0</FieldValue>
</Detail>
<Detail>
<FieldTitle>Report produced on</FieldTitle>
<FieldValue>05/13/10 12:48:17</FieldValue>
</Detail>
<Detail>
<FieldTitle>Locale</FieldTitle>
<FieldValue>en_US</FieldValue>
</Detail>
<Detail>
inline_after.xml (43%)
```

Notice the XML document that is generated. The sandbox is limited and thus is not able to launch this file in a web browser. In your own environment you should be able to view this report, formatted for easier readability, in a browser as shown below.

The generated XML report for the updated source code now shows that the function call on line 8 is successfully inlined. If the XML report is generated with the XL C/C++ 12.1 compiler, you will see the additional column number information in the "Callsite Column #" field of the inline optimization table.

Figure 10 XML Report 2



Transformation Hierarchy

- Intra-procedural Transformations
  - not available
- Inter-procedural Transformations
  - Inline Optimization Table
    - Seq #1: SuccessfulInline

Inline Optimization Table

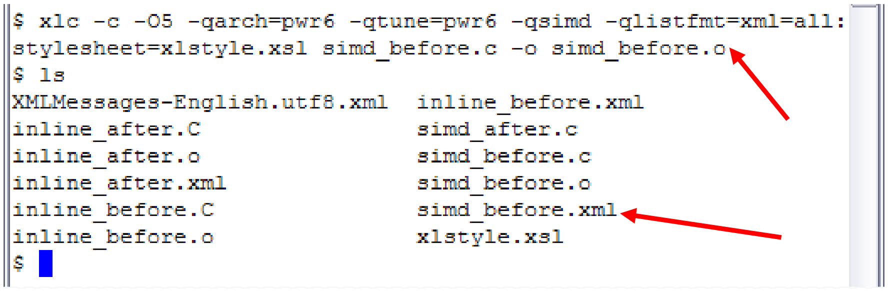
Seq #	Type	Phase	Caller Region #	Callee Region #	Callsite File #	Callsite Line #	Description
1	SuccessfulInline (success)	C++ Front End	2	1	1	8	The function was successfully inlined.

Part 2: Steps to demonstrate the loop transformation XML message:

1. Compile simd_before.c with -qarch=pwr6, -qtune=pwr6, -qsimd and –qlistfmt flags

   Command:
   ```
   xlc -c -O5 -qarch=pwr6 -qtune=pwr6 -qsimd \
   -qlistfmt=xml=all:stylesheet=xlstyle.xsl simd_before.c \
   -o simd_before.o
   ```

Figure 11 Compile 3

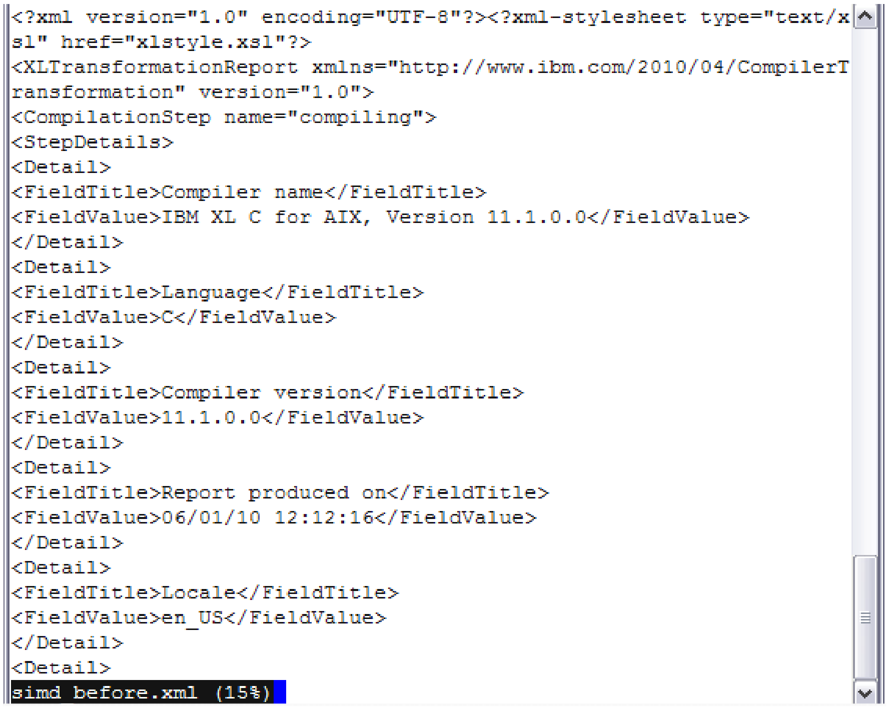


```
$ xlc -c -O5 -qarch=pwr6 -qtune=pwr6 -qsimd -qlistfmt=xml=all:
stylesheet=xlstyle.xsl simd_before.c -o simd_before.o
$ ls
XMLMessages-English.utf8.xml  inline_before.xml
inline_after.C                simd_after.c
inline_after.o                simd_before.c
inline_after.xml              simd_before.o
inline_before.C               simd_before.xml
inline_before.o               xlstyle.xsl
$
```

2. View simd_before.xml using more command

   Command:
   ```
   more simd_before.xml
   ```

Figure 12 More XML 3

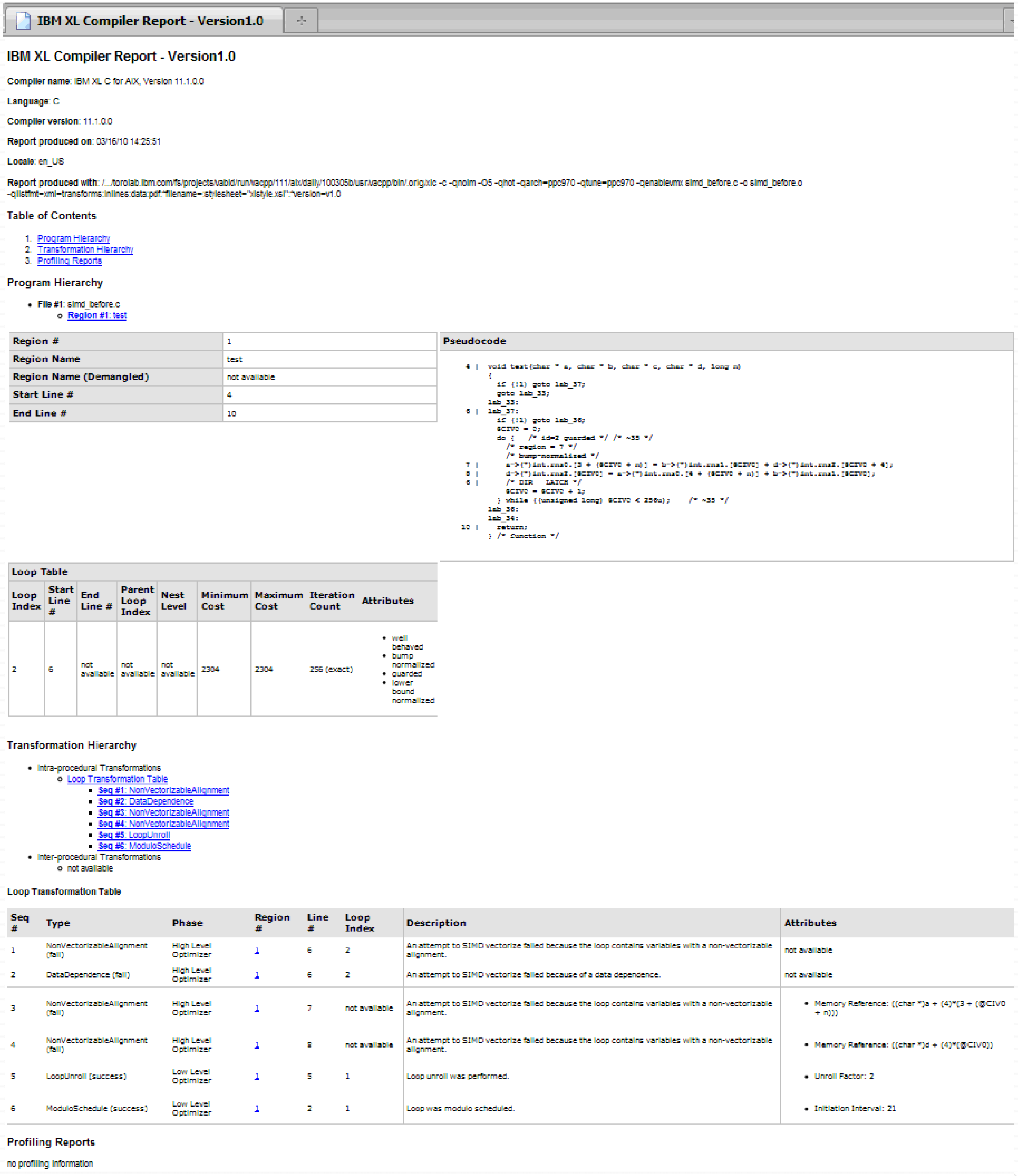


```
<?xml version="1.0" encoding="UTF-8"?><?xml-stylesheet type="text/x
sl" href="xlstyle.xsl"?>
<XLTransformationReport xmlns="http://www.ibm.com/2010/04/CompilerT
ransformation" version="1.0">
<CompilationStep name="compiling">
<StepDetails>
<Detail>
<FieldTitle>Compiler name</FieldTitle>
<FieldValue>IBM XL C for AIX, Version 11.1.0.0</FieldValue>
</Detail>
<Detail>
<FieldTitle>Language</FieldTitle>
<FieldValue>C</FieldValue>
</Detail>
<Detail>
<FieldTitle>Compiler version</FieldTitle>
<FieldValue>11.1.0.0</FieldValue>
</Detail>
<Detail>
<FieldTitle>Report produced on</FieldTitle>
<FieldValue>06/01/10 12:12:16</FieldValue>
</Detail>
<Detail>
<FieldTitle>Locale</FieldTitle>
<FieldValue>en_US</FieldValue>
</Detail>
<Detail>
simd_before.xml (15%)
```

Notice the XML document that is generated. The sandbox is limited and thus is not able to launch this file in a web browser. In your own environment, you should be able to view this report, formatted for easier readability, in a browser as shown below.

The generated XML report (full page shown in ‘Figure 13 XML Report 3’ and the “Loop Transformation Table” section shown in ‘Figure 14 Loop Table’) states that there are several failed attempts to SIMD vectorizing loops because of ‘non-vectorizable alignment’ and ‘data dependence’.

Figure 13 XML Report 3



IBM XL Compiler Report - Version1.0

Compiler name: IBM XL C for AIX, Version 11.1.0.0

Language: C

Compiler version: 11.1.0.0

Report produced on: 03/16/10 14:25:51

Locale: en_US

Report produced with: /.../torolab.ibm.com/fs/projects/vabld/run/vacpp/111/aix/daily/100305b/usr/vacpp/bin/.orig/xlc -c -qnolm -O5 -qhot -qarch=ppc970 -qtune=ppc970 -qenablevmx simd_before.c -o simd_before.o -qlistfmt=xml=transforms:inlines:data:pdf:"filename=:stylesheet="xlstyle.xsl":"version=v1.0

Table of Contents

1. Program Hierarchy
2. Transformation Hierarchy
3. Profiling Reports

Program Hierarchy

- File #1: simd_before.c
  - Region #1: test

Region #	1
Region Name	test
Region Name (Demangled)	not available
Start Line #	4
End Line #	10

Pseudocode

```
 4 |   void test(char * a, char * b, char * c, char * d, long n)
       {
         if (!1) goto lab_37;
         goto lab_33;
       lab_33:
 6 |   lab_37:
         if (!1) goto lab_36;
         @CIV0 = 0;
         do {   /* id=2 guarded */ /* ~35 */
           /* region = 7 */
           /* bump-normalized */
 7 |       a->(*)int.rns0.[3 + (@CIV0 + n)] = b->(*)int.rns1.[@CIV0] + d->(*)int.rns2.[@CIV0 + 4];
 8 |       d->(*)int.rns2.[@CIV0] = a->(*)int.rns0.[4 + (@CIV0 + n)] + b->(*)int.rns1.[@CIV0];
 6 |       /* DIR   LATCH */
           @CIV0 = @CIV0 + 1;
         } while ((unsigned long) @CIV0 < 256u);    /* ~35 */
       lab_36:
       lab_34:
10 |     return;
       } /* function */
```

Loop Table

Loop Index	Start Line #	End Line #	Parent Loop Index	Nest Level	Minimum Cost	Maximum Cost	Iteration Count	Attributes
2	6	not available	not available	not available	2304	2304	256 (exact)	• well behaved • bump normalized • guarded • lower bound normalized

Transformation Hierarchy

- Intra-procedural Transformations
  - Loop Transformation Table
    - Seq #1: NonVectorizableAlignment
    - Seq #2: DataDependence
    - Seq #3: NonVectorizableAlignment
    - Seq #4: NonVectorizableAlignment
    - Seq #5: LoopUnroll
    - Seq #6: ModuloSchedule
- Inter-procedural Transformations
  - not available

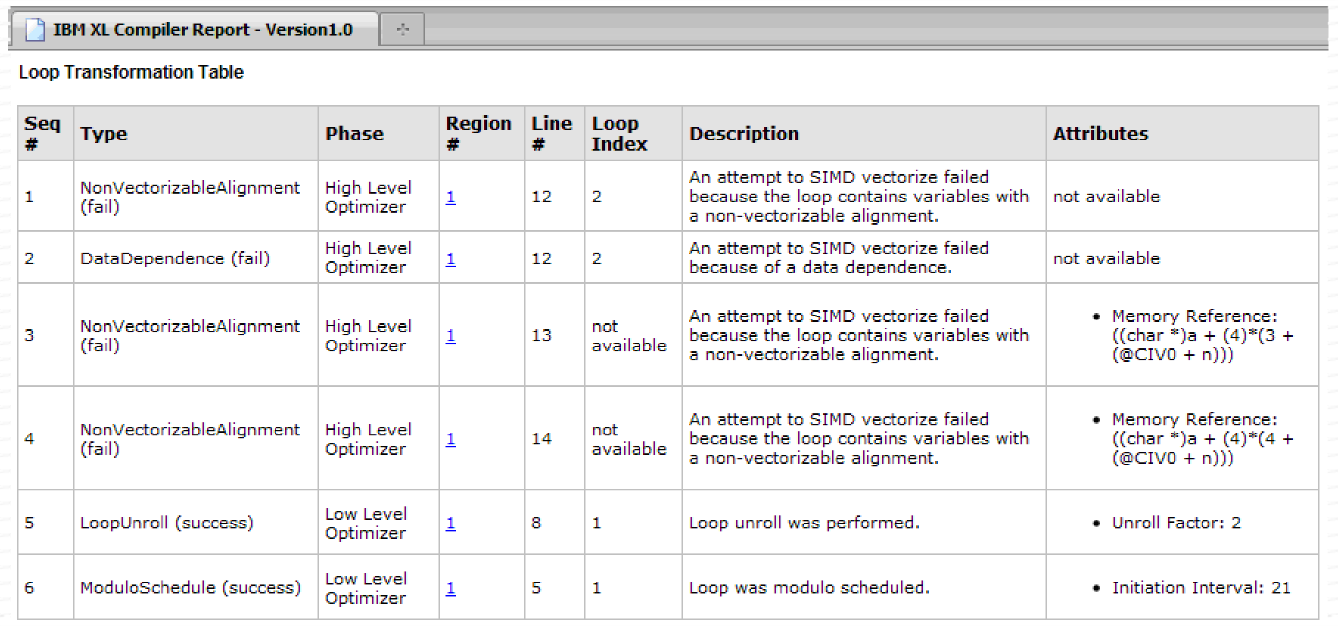
Loop Transformation Table

Seq #	Type	Phase	Region #	Line #	Loop Index	Description	Attributes
1	NonVectorizableAlignment (fail)	High Level Optimizer	1	6	2	An attempt to SIMD vectorize failed because the loop contains variables with a non-vectorizable alignment.	not available
2	DataDependence (fail)	High Level Optimizer	1	6	2	An attempt to SIMD vectorize failed because of a data dependence.	not available
3	NonVectorizableAlignment (fail)	High Level Optimizer	1	7	not available	An attempt to SIMD vectorize failed because the loop contains variables with a non-vectorizable alignment.	• Memory Reference: [(char *)a + (4)*(3 + (@CIV0 + n))]
4	NonVectorizableAlignment (fail)	High Level Optimizer	1	8	not available	An attempt to SIMD vectorize failed because the loop contains variables with a non-vectorizable alignment.	• Memory Reference: [(char *)d + (4)*(@CIV0)]
5	LoopUnroll (success)	Low Level Optimizer	1	5	1	Loop unroll was performed.	• Unroll Factor: 2
6	ModuloSchedule (success)	Low Level Optimizer	1	2	1	Loop was modulo scheduled.	• Initiation Interval: 21

Profiling Reports

no profiling information

Figure 14 Loop Table



Loop Transformation Table

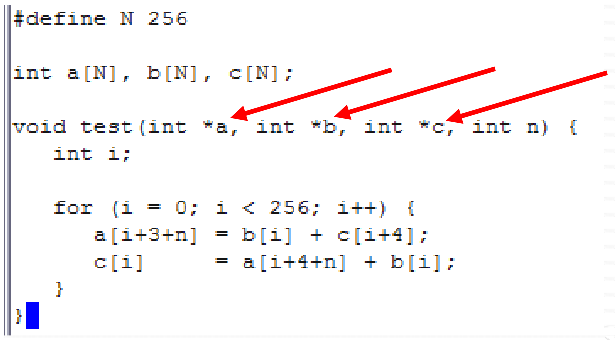
Seq #	Type	Phase	Region #	Line #	Loop Index	Description	Attributes
1	NonVectorizableAlignment (fail)	High Level Optimizer	1	12	2	An attempt to SIMD vectorize failed because the loop contains variables with a non-vectorizable alignment.	not available
2	DataDependence (fail)	High Level Optimizer	1	12	2	An attempt to SIMD vectorize failed because of a data dependence.	not available
3	NonVectorizableAlignment (fail)	High Level Optimizer	1	13	not available	An attempt to SIMD vectorize failed because the loop contains variables with a non-vectorizable alignment.	• Memory Reference: ((char *)a + (4)*(3 + (@CIV0 + n)))
4	NonVectorizableAlignment (fail)	High Level Optimizer	1	14	not available	An attempt to SIMD vectorize failed because the loop contains variables with a non-vectorizable alignment.	• Memory Reference: ((char *)a + (4)*(4 + (@CIV0 + n)))
5	LoopUnroll (success)	Low Level Optimizer	1	8	1	Loop unroll was performed.	• Unroll Factor: 2
6	ModuloSchedule (success)	Low Level Optimizer	1	5	1	Loop was modulo scheduled.	• Initiation Interval: 21

3. Look at the difference between simd_before.c and simd_after.c using `diff` command.

   Command:

   ```
   diff simd_before.c simd_after.c
   ```

Figure 15 simd_before.c



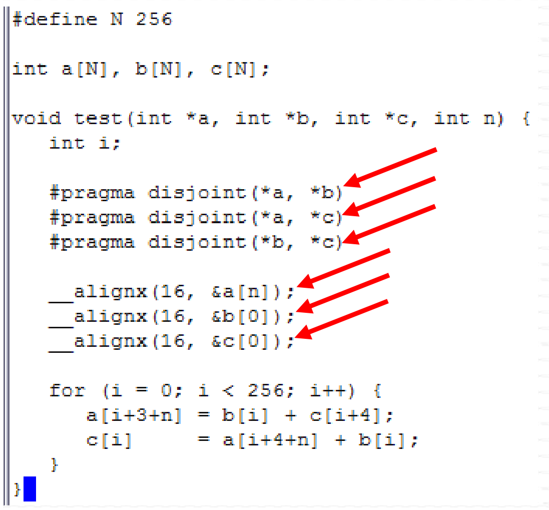
```
#define N 256

int a[N], b[N], c[N];

void test(int *a, int *b, int *c, int n) {
   int i;

   for (i = 0; i < 256; i++) {
      a[i+3+n] = b[i] + c[i+4];
      c[i]     = a[i+4+n] + b[i];
   }
}
```

Figure 16 simd_after.c



```
#define N 256

int a[N], b[N], c[N];

void test(int *a, int *b, int *c, int n) {
   int i;

   #pragma disjoint(*a, *b)
   #pragma disjoint(*a, *c)
   #pragma disjoint(*b, *c)

   __alignx(16, &a[n]);
   __alignx(16, &b[0]);
   __alignx(16, &c[0]);

   for (i = 0; i < 256; i++) {
      a[i+3+n] = b[i] + c[i+4];
      c[i]     = a[i+4+n] + b[i];
   }
}
```

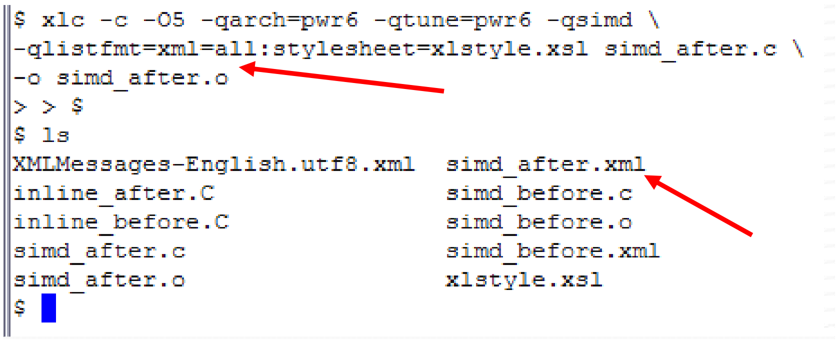
Looking at the source code of simd_before.c, the user can determine that integer pointers a, b, and c, as arguments of function test, may have data dependence, causing the potential optimization to be omitted by the compiler. In addition, data in integer arrays a, b, and c may not align with a 16-byte boundary, causing them not to be vectorized by the compiler.

With these analyses in mind, the user can now modify the code by removing the data dependence among a, b, and c; and aligning a, b, and c with a 16-byte boundary. simd_after.c contains the code after these changes.

4. Compile simd_after.c with qarch=pwr6, -qtune=pwr6, -qsimd and –qlistfmt flags

   Command:
   ```
   xlc -c -O5 -qarch=pwr6 -qtune=pwr6 -qsimd \
   -qlistfmt=xml=all:stylesheet=xlstyle.xsl simd_after.c \
   -o simd_after.o
   ```

Figure 17 Compile 4

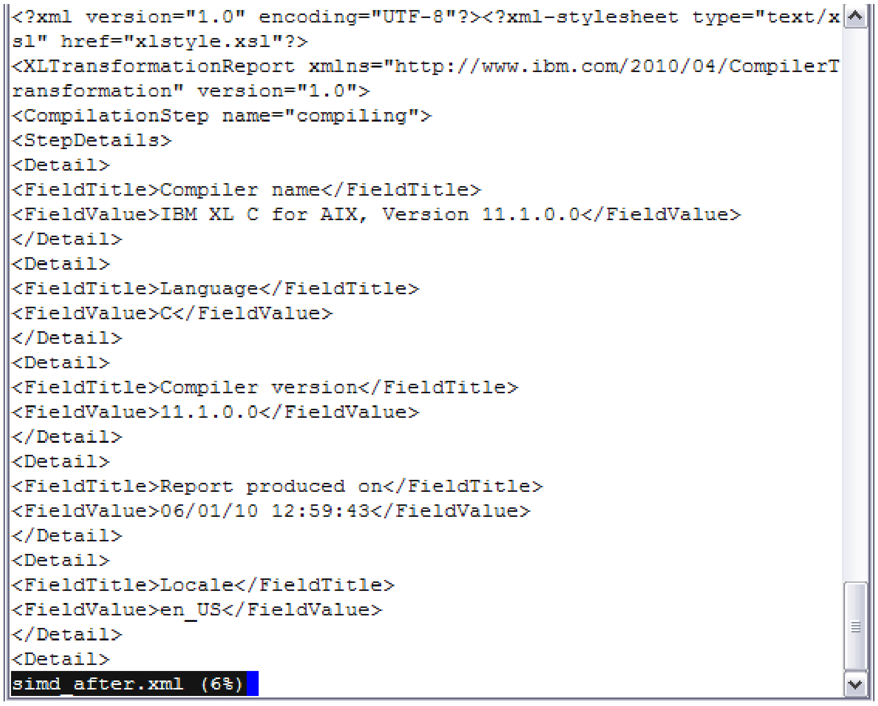


```
$ xlc -c -O5 -qarch=pwr6 -qtune=pwr6 -qsimd \
-qlistfmt=xml=all:stylesheet=xlstyle.xsl simd_after.c \
-o simd_after.o
> > $
$ ls
XMLMessages-English.utf8.xml   simd_after.xml
inline_after.C                 simd_before.c
inline_before.C                simd_before.o
simd_after.c                   simd_before.xml
simd_after.o                   xlstyle.xsl
$
```

5. View the simd_after.xml using more command

   Command:
   ```
   more simd_after.xml
   ```

Figure 18 More XML 4

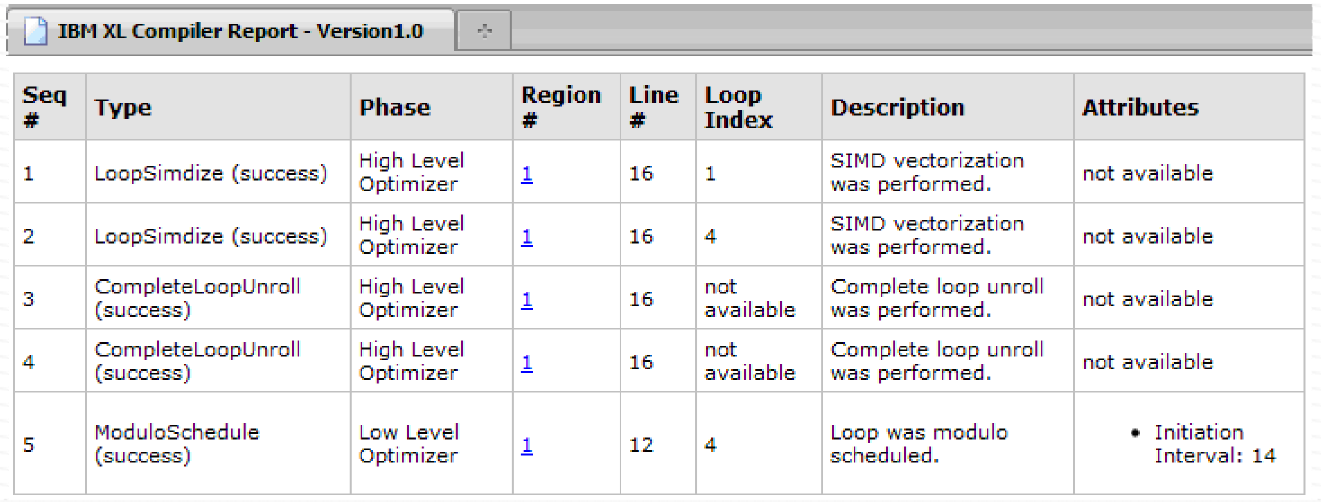


```
<?xml version="1.0" encoding="UTF-8"?><?xml-stylesheet type="text/x
sl" href="xlstyle.xsl"?>
<XLTransformationReport xmlns="http://www.ibm.com/2010/04/CompilerT
ransformation" version="1.0">
<CompilationStep name="compiling">
<StepDetails>
<Detail>
<FieldTitle>Compiler name</FieldTitle>
<FieldValue>IBM XL C for AIX, Version 11.1.0.0</FieldValue>
</Detail>
<Detail>
<FieldTitle>Language</FieldTitle>
<FieldValue>C</FieldValue>
</Detail>
<Detail>
<FieldTitle>Compiler version</FieldTitle>
<FieldValue>11.1.0.0</FieldValue>
</Detail>
<Detail>
<FieldTitle>Report produced on</FieldTitle>
<FieldValue>06/01/10 12:59:43</FieldValue>
</Detail>
<Detail>
<FieldTitle>Locale</FieldTitle>
<FieldValue>en_US</FieldValue>
</Detail>
<Detail>
simd_after.xml (6%)
```

Notice the XML document that is generated. The sandbox is limited and thus is not able to launch this file in a web browser. In your own environment, you should be able to view this report, formatted for easier readability, in a browser as shown below.

The generated XML report for the updated source code now shows that SIMD vectorization and complete unroll are both performed by the compiler.

Figure 19 XML Report 4



Seq #	Type	Phase	Region #	Line #	Loop Index	Description	Attributes
1	LoopSimdize (success)	High Level Optimizer	1	16	1	SIMD vectorization was performed.	not available
2	LoopSimdize (success)	High Level Optimizer	1	16	4	SIMD vectorization was performed.	not available
3	CompleteLoopUnroll (success)	High Level Optimizer	1	16	not available	Complete loop unroll was performed.	not available
4	CompleteLoopUnroll (success)	High Level Optimizer	1	16	not available	Complete loop unroll was performed.	not available
5	ModuloSchedule (success)	Low Level Optimizer	1	12	4	Loop was modulo scheduled.	• Initiation Interval: 14

This concludes this tutorial “Explore Optimization Opportunities with XML Transformation Reports in IBM XL C/C++ and XL Fortran for AIX Compilers”.

What you have learned

In this exercise you learnt how to:

- Use IBM XL C/C++ for AIX compiler to build source code.
- Use optimization flags to optimize the program
- How to generate and use XML reports to identify and improve coding issues that hinder the compiler optimization

Conclusion

This tutorial demonstrated how to use the XML Transformation Reports feature available in the XL C/C++ V11.1/V12.1 and XL Fortran V13.1/14.1 for AIX compilers. It gave two examples to show that the XML reports generated by the compiler, provide useful and readable information about optimizations that the compiler has or has not performed, and the user could then update the code based on the messages received, leading to code that can be further optimized by the compiler.

Trademarks

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

UNIX is a registered trademark of The Open Group in the United States and other countries.

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

Resources

Optimization:

White Paper: Optimizing C code at optimization level 2
http://www.ibm.com/support/docview.wss?uid=swg27022103

Tutorial: Applying XL C Compiler Optimization on AIX: Optimization Level 2
http://www.ibm.com/support/docview.wss?uid=swg27022278

Command Line option manual
http://pic.dhe.ibm.com/infocenter/comphelp/v121v141/index.jsp?topic=/com.ibm.xlcpp121.aix.doc/compiler_ref/opt_optimize.html

Papers

"Optimizing C Code at Optimization Level 2"
http://www.ibm.com/support/docview.wss?uid=swg27022103

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

Community Cafe

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