IBM XL C/C++ for Multicore Acceleration for Linux,
V10.1

Getting Started with XL C/C++

Version 10.1

GC23-8572-00

IBM XL C/C++ for Multicore Acceleration for Linux,
V10.1

Getting Started with XL C/C++

Version 10.1

GC23-8572-00

Note
Before using this information and the product it supports, read the information in "Notices" on page 25.

First edition

This edition applies to IBM XL C/C++ for Multicore Acceleration for Linux on Power Systems, V10.1 and IBM XL C/C++ for Multicore Acceleration for Linux on x86 Systems, V10.1 (Programs 5724-T42 & 5724-T43), and to all subsequent releases and modifications until otherwise indicated in new editions. Make sure you are using the correct edition for the level of the product.

Contents

About this document

This document contains overview and basic usage information for the IBM® XL C/C++ for Multicore Acceleration for Linux®, V10.1 compiler.

Who should read this document

This document is intended for C and C++ developers who are looking for introductory overview and usage information for XL C/C++. It assumes that you have some familiarity with command-line compilers, a basic knowledge of the C and C++ programming languages, and basic knowledge of operating system commands. Programmers new to XL C/C++ can use this document to find information on the capabilities and features unique to XL C/C++.

How to use this document

Unless indicated otherwise, all of the text in this reference pertains to both C and C++ languages. Where there are differences between languages, these are indicated through qualifying text and icons, as described in "Conventions." Additionally, unless indicated otherwise, text in this document pertains to compilation targeting both the PowerPC® Processing Unit (PPU) and Synergistic Processor Units (SPUs).

XL C/C++ provides several compiler invocation commands depending on source code language levels and whether you are compiling for the PowerPC Processor Unit (PPU) or Synergistic Processor Units (SPUs). However, for convenience, this document uses only the basic **ppuxlc**, **ppuxlc++**, **spuxlc**, and **spuxlc++** invocation commands to describe the actions of the C and C++ compiler.

While this document covers information on configuring the compiler environment, and compiling and linking C or C++ applications using the XL C/C++ compiler, it does not include the following topics:

- Compiler installation: see the *XL C/C++ Installation Guide* for information on installing XL C/C++.
- Compiler options: see the *XL C/C++ Compiler Reference* for detailed information on the syntax and usage of compiler options.
- The C or C++ programming languages: see the *XL C/C++ Language Reference* for information on the syntax, semantics, and IBM implementation of the C or C++ programming languages.
- Programming topics: see the *XL C/C++ Optimization and Programming Guide* for detailed information on developing applications with XL C/C++, with a focus on program portability and optimization.

Conventions

Typographical conventions

The following table explains the typographical conventions used in the IBM XL C/C++ for Multicore Acceleration for Linux, V10.1 information.

Table 1. Typographical conventions

Typeface	Indicates	Example
bold	Lowercase commands, executable names, compiler options, and directives.	The compiler provides basic invocation commands, **xlc** and **xlC** (**xlc++**), along with several other compiler invocation commands to support various C/C++ language levels and compilation environments.
italics	Parameters or variables whose actual names or values are to be supplied by the user. Italics are also used to introduce new terms.	Make sure that you update the *size* parameter if you return more than the *size* requested.
underlining	The default setting of a parameter of a compiler option or directive.	nomaf | maf
`monospace`	Programming keywords and library functions, compiler builtins, examples of program code, command strings, or user-defined names.	To compile and optimize myprogram.c, enter: `xlc myprogram.c -O3`.

Other conventions

In addition to typographical conventions, the following conventions are used:

- *(SPU only)* indicates functionality that only applies to code targeting the Synergistic Processor Unit (SPU), whether it is compiled using an `spu` or `cbe` prefixed compiler invocation command.
- *(PPU only)* indicates functionality that only applies to code targeting the Power Processor Unit (PPU), whether it is compiled using a `ppu` or `cbe` prefixed compiler invocation command.

Qualifying elements (icons)

Most features described in this information apply to both C and C++ languages. In descriptions of language elements where a feature is exclusive to one language, or where functionality differs between languages, this information uses icons to delineate segments of text as follows:

Table 2. Qualifying elements

Qualifier/Icon	Meaning
C only, or C only begins   C only ends	The text describes a feature that is supported in the C language only; or describes behavior that is specific to the C language.

Table 2. Qualifying elements (continued)

Qualifier/Icon	Meaning
C++ only, or C++ only begins C++ C++ C++ only ends	The text describes a feature that is supported in the C++ language only; or describes behavior that is specific to the C++ language.
IBM extension begins IBM IBM IBM extension ends	The text describes a feature that is an IBM extension to the standard language specifications.

Syntax diagrams

Throughout this information, diagrams illustrate XL C/C++ syntax. This section will help you to interpret and use those diagrams.

- Read the syntax diagrams from left to right, from top to bottom, following the path of the line.

  The ►►— symbol indicates the beginning of a command, directive, or statement.

  The —► symbol indicates that the command, directive, or statement syntax is continued on the next line.

  The ►— symbol indicates that a command, directive, or statement is continued from the previous line.

  The —►◄ symbol indicates the end of a command, directive, or statement.

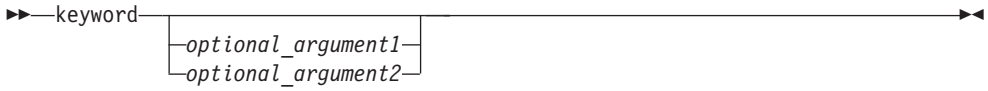
  Fragments, which are diagrams of syntactical units other than complete commands, directives, or statements, start with the |— symbol and end with the —| symbol.
- Required items are shown on the horizontal line (the main path):

- Optional items are shown below the main path:

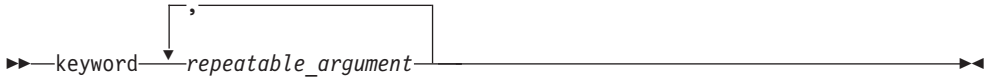
- If you can choose from two or more items, they are shown vertically, in a stack.

  If you *must* choose one of the items, one item of the stack is shown on the main path.

  If choosing one of the items is optional, the entire stack is shown below the main path.



- An arrow returning to the left above the main line (a repeat arrow) indicates that you can make more than one choice from the stacked items or repeat an item. The separator character, if it is other than a blank, is also indicated:

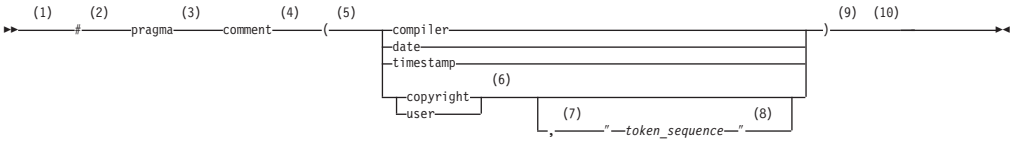


- The item that is the default is shown above the main path.

- Keywords are shown in nonitalic letters and should be entered exactly as shown.
- Variables are shown in italicized lowercase letters. They represent user-supplied names or values.
- If punctuation marks, parentheses, arithmetic operators, or other such symbols are shown, you must enter them as part of the syntax.

Sample syntax diagram

The following syntax diagram example shows the syntax for the **#pragma comment** directive.



Notes:

1 This is the start of the syntax diagram.

2 The symbol # must appear first.

3 The keyword `pragma` must appear following the # symbol.

4 The name of the pragma `comment` must appear following the keyword `pragma`.

5 An opening parenthesis must be present.

6 The comment type must be entered only as one of the types indicated: `compiler`, `date`, `timestamp`, `copyright`, or `user`.

7 A comma must appear between the comment type `copyright` or `user`, and an optional character string.

8 A character string must follow the comma. The character string must be enclosed in double quotation marks.

9 A closing parenthesis is required.

10 This is the end of the syntax diagram.

The following examples of the **#pragma comment** directive are syntactically correct according to the diagram shown above:

```
#pragma comment(date)
#pragma comment(user)
#pragma comment(copyright,"This text will appear in the module")
```

Examples in this information

The examples in this information, except where otherwise noted, are coded in a simple style that does not try to conserve storage, check for errors, achieve fast performance, or demonstrate all possible methods to achieve a specific result.Also, examples may use different compiler invocation commands interchangeably or simply indicate *invocation*. For detailed information on the commands available to invoke the compiler see "Invoking the compiler" in the *XL C/C++ Compiler Reference*.

The examples for installation information are labelled as either *Example* or *Basic example*. *Basic examples* are intended to document a procedure as it would be performed during a basic, or default, installation; these need little or no modification.

Related information

The following sections provide related information for XL C/C++:

IBM XL C/C++ information

XL C/C++ provides product information in the following formats:

- README files

  README files contain late-breaking information, including changes and corrections to the product information. README files are located by default in the XL C/C++ directory and in the root directory of the installation CD.
- Installable man pages

  Man pages are provided for the compiler invocations and all command-line utilities provided with the product. Instructions for installing and accessing the man pages are provided in the *IBM XL C/C++ for Multicore Acceleration for Linux, V10.1 Installation Guide*.
- Information center

  The information center of searchable HTML files is viewable on the Web at http://publib.boulder.ibm.com/infocenter/cellcomp/v101v121/index.jsp.
- PDF documents

  PDF documents are located by default in the /opt/ibmcmp/xlc/cbe/10.1/doc/en_US/pdf/ directory. The PDF files are also available on the Web at http://www.ibm.com/software/awdtools/xlcpp/multicore/library/.

  The following files comprise the full set of XL C/C++ product information:

Table 3. XL C/C++ PDF files

Document title	PDF file name	Description
IBM XL C/C++ for Multicore Acceleration for Linux, V10.1 Installation Guide, GC23-8574-00	install.pdf	Contains information for installing XL C/C++ and configuring your environment for basic compilation and program execution.

Table 3. XL C/C++ PDF files (continued)

Document title	PDF file name	Description
Getting Started with IBM XL C/C++ for Multicore Acceleration for Linux, V10.1, GC23-8572-00	getstart.pdf	Contains an introduction to the XL C/C++ product, with information on setting up and configuring your environment, compiling and linking programs, and troubleshooting compilation errors.
IBM XL C/C++ for Multicore Acceleration for Linux, V10.1 Compiler Reference, SC23-8570-00	compiler.pdf	Contains information about the various compiler options, pragmas, macros, environment variables, and built-in functions.
IBM XL C/C++ for Multicore Acceleration for Linux, V10.1 Language Reference, SC23-8573-00	langref.pdf	Contains information about the C and C++ programming languages, as supported by IBM, including language extensions for portability and conformance to nonproprietary standards.
IBM XL C/C++ for Multicore Acceleration for Linux, V10.1 Optimization and Programming Guide, SC23-8571-00	proguide.pdf	Contains information on advanced programming topics, such as application porting, library development, application optimization, and the XL C/C++ high-performance libraries.

To read a PDF file, use the Adobe® Reader. If you do not have the Adobe Reader, you can download it (subject to license terms) from the Adobe Web site at http://www.adobe.com.

More information related to XL C/C++ including redbooks, white papers, tutorials, and other articles, is available on the Web at:

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

Standards and specifications

XL C/C++ is designed to support the following standards and specifications. You can refer to these standards for precise definitions of some of the features found in this information.

- *Information Technology – Programming languages – C, ISO/IEC 9899:1990*, also known as *C89*.
- *Information Technology – Programming languages – C, ISO/IEC 9899:1999*, also known as *C99*.
- *Information Technology – Programming languages – C++, ISO/IEC 14882:1998*, also known as *C++98*.
- *Information Technology – Programming languages – C++, ISO/IEC 14882:2003(E)*, also known as *Standard C++*.
- *Information Technology – Programming languages – Extensions for the programming language C to support new character data types, ISO/IEC DTR 19769*. This draft technical report has been accepted by the C standards committee, and is available at http://www.open-std.org/JTC1/SC22/WG14/www/docs/n1040.pdf.
- *Draft Technical Report on C++ Library Extensions, ISO/IEC DTR 19768*. This draft technical report has been submitted to the C++ standards committee, and is available at http://www.open-std.org/JTC1/SC22/WG21/docs/papers/2005/n1836.pdf.

- *AltiVec Technology Programming Interface Manual*, Motorola Inc. This specification for vector data types, to support vector processing technology, is available at http://www.freescale.com/files/32bit/doc/ref_manual/ALTIVECPIM.pdf.

Other IBM information

- *IBM C/C++ Language Extensions for Cell Broadband Engine Architecture*, available at http://publib.boulder.ibm.com/infocenter/systems/topic/eiccg/eiccgkickoff.htm
- Specifications, white papers, and other technical information for the Cell Broadband Engine™ architecture are available at http://www.ibm.com/chips/techlib/techlib.nsf/products/Cell_Broadband_Engine.
- The Cell Broadband Engine resource center, at http://www.ibm.com/developerworks/power/cell/, is the central repository for technical information, including articles, tutorials, programming guides, and educational resources.
- The IBM Systems Information Center, at http://publib.boulder.ibm.com/infocenter/systems/index.jsp is a resource for technical information about IBM systems, including the Cell Broadband Engine solution.

Other information

- *Using the GNU Compiler Collection* available at http://gcc.gnu.org/onlinedocs

Technical support

Additional technical support is available from the XL C/C++ Support page at http://www.ibm.com/software/awdtools/xlcpp/support. This page provides a portal with search capabilities to a large selection of Technotes and other support information.

If you cannot find what you need, you can send e-mail to compinfo@ca.ibm.com.

For the latest information about XL C/C++, visit the product information site at http://www.ibm.com/software/awdtools/xlcpp.

How to send your comments

Your feedback is important in helping to provide accurate and high-quality information. If you have any comments about this information or any other XL C/C++ information, send your comments by e-mail to compinfo@ca.ibm.com.

Be sure to include the name of the information, the part number of the information, the version of XL C/C++, and, if applicable, the specific location of the text you are commenting on (for example, a page number or table number).

Chapter 1. Introducing XL C/C++

IBM XL C/C++ for Multicore Acceleration for Linux, V10.1 is an advanced, high-performance compiler that can be used for developing complex, computationally intensive programs.

This section discusses the features of the XL C/C++ compiler at a high level. It is intended for people who are evaluating the compiler, and for new users who want to find out more about the product.

Commonality with other IBM compilers

IBM XL C/C++ for Multicore Acceleration for Linux, V10.1 is part of a larger family of IBM C, C++, and Fortran compilers.

XL C/C++, together with XL Fortran, comprise the family of XL compilers.

These compilers are derived from a common code base that shares compiler function and optimization technologies for a variety of platforms and programming languages. Programming environments include IBM AIX®, IBM Blue Gene®/L™, IBM Blue Gene®/P™, the Cell Broadband Engine architecture, IBM i, selected Linux distributions, IBM z/OS®, and IBM z/VM®. The common code base, along with compliance with international programming language standards, helps support consistent compiler performance and ease of program portability across multiple operating systems and hardware platforms.

IBM XL C/C++ for Multicore Acceleration for Linux, V10.1

IBM XL C/C++ for Multicore Acceleration for Linux, V10.1 is the latest addition to the IBM XL family of compilers. It adopts proven high-performance compiler technologies used in its compiler family predecessors, and adds new features tailored to exploit the unique performance capabilities of processors compliant with the new Cell Broadband Engine architecture.

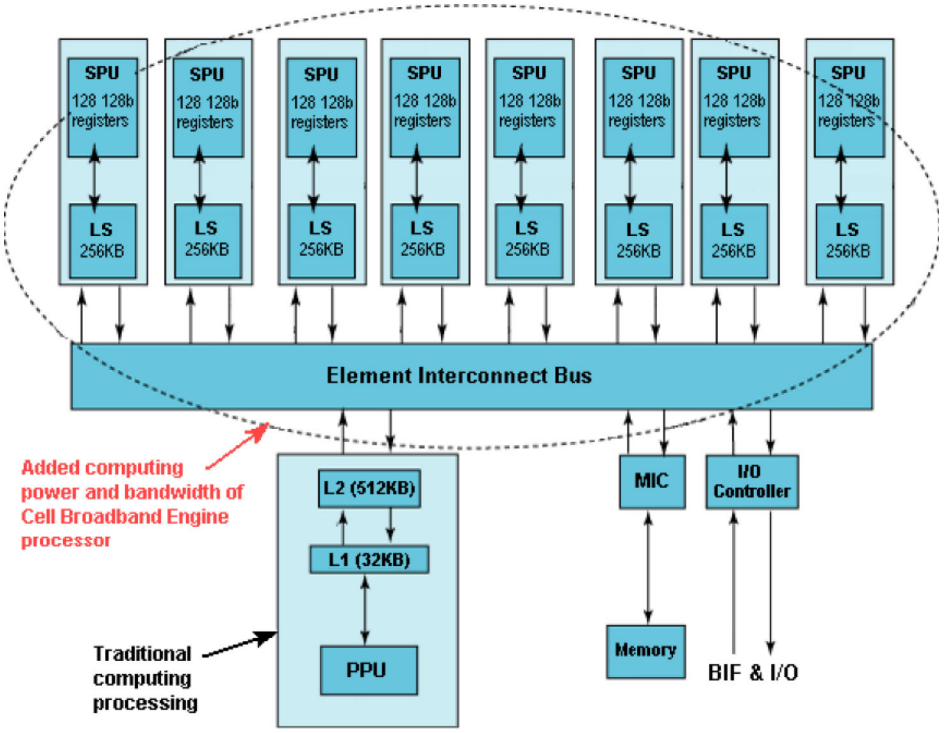
XL C/C++ is a cross-compiler. First, you compile your applications on an x86 system or IBM Power Systems server compilation host running Red Hat Enterprise Linux 5.2 (RHEL 5.2). Then you move the executable application produced by the compiler onto a Cell/B.E. system also running the RHEL 5.2 Linux distribution. The Cell/B.E. system will be the execution host where you will actually run your compiled application.

About the Cell Broadband Engine architecture

The Cell Broadband Engine architecture specification describes a new single-chip multiprocessor based upon the 64-bit Power Architecture® technology, but with unique features directed toward distributed processing and media-rich applications.

At the heart of the new multiprocessor is the Power Processor Unit (PPU). The PPU is a 64-bit processor fully compliant with the Power Architecture standard, and capable of running both operating systems and applications. The multiprocessor also incorporates a set of eight high-performance SIMD Synergistic Processor Units (SPUs) into its design. The SPUs are optimized for running

computationally intensive applications, operate independently of each other, and can access memory shared between all SPUs and the PPU.



In operation, the PPU runs the operating system and performs high-level application control, while the SPUs divide and perform an application's computational work between them.

For more information on the Cell Broadband Engine architecture, see *"Cell Broadband Engine Architecture from 20,000 feet"* at http://www.ibm.com/developerworks/power/library/pa-cbea.html.

A highly configurable compiler

You can use a variety of compiler invocation commands and options to tailor the compiler to your unique compilation requirements.

Compiler invocation and linking commands for separate PPU and SPU programs

If you have written separate PPU and SPU programs, you must compile PPU and SPU program code in separate steps.

Several versions of PPU-specific compiler invocation commands are provided. In most cases, you should compile using the basic **ppuxlc**, **ppuxlC**, or **ppuxlc++** compiler invocation commands, but other variants are also provided to help you meet special compilation needs.

SPU-specific invocation commands are also provided with **spuxlc**, **spuxlC**, **spuxlc++**, and their variants.

The IBM Software Developer Kit for Multicore Acceleration V3.1 (SDK 3.1) includes the **ppu-embedspu** command to link compiled PPU and SPU objects together.

For detailed information about compiler invocation commands provided with XL C/C++, see "Invoking the compiler" in the *XL C/C++ Compiler Reference*.

Compiler invocation and linking commands for combined PPU and SPU programs

XL C/C++ can compile programs that contain both PPU and SPU code. If you have written the SPU portions of your code using OpenMP directives, you can compile PPU and SPU program code in one step.

You should compile using the basic invocation commands, **cbexlc**, **cbexlC**, or **cbexlc++** for these programs.

For more information about this technology see "Single-source compiler technology" on page 9.

Compiler options

You can choose from a large selection of compiler options to control compiler behavior. Different categories of options help you to debug your applications, optimize and tune application performance, select language levels and extensions for compatibility with non-standard features and behaviors supported by other C or C++ compilers, and perform many other common tasks that would otherwise require changing the source code.

XL C/C++ lets you specify compiler options through a combination of environment variables, compiler configuration files, command line options, and compiler directive statements embedded in your program source.

For more information about XL C/C++ compiler options, see "Compiler options reference" in the *XL C/C++ Compiler Reference*.

Language standards compliance

The compiler supports the following programming language specifications for C/C++:

- ISO/IEC 9899:1999 (C99)
- ISO/IEC 9899:1990 (referred to as C89)
- ISO/IEC 14882:2003 (referred to as Standard C++)
- ISO/IEC 14882:1998, the first official specification of the language (referred to as C++98)

In addition to the standardized language levels, XL C/C++ supports language extensions, including:

- OpenMP V2.5 extensions to support portable parallelized programming
- Language extensions to support vector programming
- Language extensions to support SPU programming
- A subset of GNU C and C++ language extensions

See "Language levels and language extensions" in the *XL C/C++ Language Reference* for more information about C/C++ language specifications and extensions.

The IBM XL C/C++ compiler contains a separate SPU cross-compiler that supports the standards defined in the following documents:

- IBM C/C++ Language Extensions for Cell Broadband Engine Architecture V2.6
- SPU Application Binary Interface Specification V1.8

- SPU Instruction Set Architecture V1.2

IBM XL C/C++ supports all language extensions described in the ″C/C++ Language Extensions for Cell Broadband Engine Architecture V2.6″ specification except for the following:

- The Shift and Rotate Intrinsics spu_sr, spu_sra, spu_srqw, spu_srqwbyte and spu_srqwbytebc as listed in section 2.10
- The __builtin_expect_call builtin function call
- The recommended vector printf format controls as specified in section 8.1.1 due to library restrictions
- The C99 complex math library as specified in section 8.1.1 due to library restrictions
- In C++ code, the XL C/C++ compiler currently supports mapping between SPU and VMX intrinsics as defined in section 5 only.

IBM XL C/C++ supports the ″IBM Software Development Kit (SDK) for Multicore Acceleration V3.1 Programmer's Guide″ specification except for the following:

- The XL C/C++ compiler currently allows __ea variable declarations, but not variable definitions.

Compatibility with GNU

XL C/C++ supports a subset of the GNU compiler command options to facilitate porting applications developed with **gcc** and **g++** compilers.

This support is available when the **ppugxlc**, **ppugxlC**, **ppugxlc++**, **spugxlc**, **spugxlC**, or **spugxlc++** invocation commands are used together with select GNU compiler options. Where possible, the compiler maps GNU options to their XL C/C++ compiler option counterparts before invoking the compiler.

These invocation commands use a plain text configuration file to control GNU-to-XL C/C++ option mappings and defaults. You can customize this configuration file to better meet the needs of any unique compilation requirements you may have. See ″Reusing GNU C/C++ compiler options with gxlc and gxlc++″ in the *XL C/C++ Compiler Reference* for more information.

XL C/C++ uses GNU C and GNU C++ header files together with the GNU C and C++ runtime libraries to produce code that is binary-compatible with that produced by the GNU Compiler Collection (GCC). Portions of an application can be built with XL C/C++ and combined with portions built with GCC to produce an application that behaves as if it had been built solely with GCC.

To achieve binary compatibility with GCC-compiled code, a program compiled with XL C/C++ includes the same headers as those used by a GNU compiler residing on the same system. To ensure that the proper versions of headers and runtime libraries are present on the system, the prerequisite GCC compiler must be installed before installing XL C/C++.

Some additional noteworthy points about this relationship are:

- IBM built-in functions coexist with GNU C built-ins.
- Compilation of C and C++ programs uses the GNU C and GNU C++ header files.
- Compilation uses the GNU assembler for assembler input files.
- Compiled C code is linked to the GNU C runtime libraries.

- Compiled C++ code is linked to the GNU C and GNU C++ runtime libraries.
- Debugging uses the **ppu-gdb** and **spu-gdb** debuggers provided with the IBM Software Developer Kit for Multicore Acceleration V3.1 (SDK 3.1).

Source-code migration and conformance checking

XL C/C++ helps protect your investment in your existing C/C++ source code by providing compiler invocation commands that instruct the compiler to compile your application code to a specific language level.

You can also use the **-qlanglvl** compiler option to specify a given language level, and the compiler will issue warnings, errors, and severe error messages if language or language extension elements in your program source do not conform to that language level.

See "qlanglvl" in the *XL C/C++ Compiler Reference* for more information.

Libraries

XL C/C++ includes a runtime environment containing a number of libraries.

Mathematical Acceleration Subsystem library

The Mathematical Acceleration Subsystem (MASS) library consists of scalar and vector mathematical intrinsic functions tuned specifically for optimum performance on supported processor architectures and SIMD mathematical intrinsic functions tuned specifically for the SPUs. You can choose a MASS library to support high-performance computing on a broad range of processors, or you can select a library tuned to support a specific processor family.

The MASS library functions support both 32-bit and 64-bit compilation modes, are thread-safe, and offer improved performance over the default `libm` math library routines. They are called automatically when you request specific levels of optimization for your application. You can also make explicit calls to MASS library functions regardless of whether optimization options are in effect or not.

See "Using the Mathematical Acceleration Subsystem" in the *XL C/C++ Optimization and Programming Guide* for more information.

Basic Linear Algebra Subprograms (PPU only)

The Basic Linear Algebra Subprograms (BLAS) set of high-performance algebraic functions are shipped in the libxlopt library. These functions let you:

- Compute the matrix-vector product for a general matrix or its transpose.
- Perform combined matrix multiplication and addition for general matrices or their transposes.

For more information about using the BLAS functions, see "Using the Basic Linear Algebra Subprograms" in the *XL C/C++ Optimization and Programming Guide*.

Other libraries

The following are also shipped with XL C/C++:

- C++ Runtime Library contains support routines needed by the compiler.

Tools and utilities

There are many tools and utilities that are included with XL C/C++.

new_install
After you install IBM XL C/C++ for Multicore Acceleration for Linux, V10.1, running this utility will configure the compiler for use on your system.

xlc_configure
Use this utility if you need to update your compiler configuration file following SDK updates or if you want to create customized compiler configuration files.

cleanpdf command (PPU only)
A command related to profile-directed feedback (PDF), **cleanpdf** removes all profiling information from the directory to which profile-directed feedback data is written.

resetpdf command (PPU only)
The current behavior of the cleanpdf command is the same as the **resetpdf** command, and is retained for compatibility with earlier releases on other platforms.

ppugxlc and ppugxlc++ utilities
When compiling PPU code, you can use these invocation methods to translate a GNU C or GNU C++ invocation command and associated options into a corresponding **ppuxlc** or **ppuxlc++** compiler invocation. These utilities help minimize the number of changes to your existing GNU compiler makefiles to help you make the transition to using the XL C/C++ compiler. See in the *XL C/C++ Compiler Reference* for more information.

spugxlc and spugxlc++ utilities
When compiling SPU code, you can use these invocation methods to translate a GNU C or GNU C++ invocation command and associated options into a corresponding **spuxlc** or **spuxlc++** compiler invocation. These utilities help minimize the number of changes to your existing GNU compiler makefiles to help you make the transition to using the XL C/C++ compiler. See in the *XL C/C++ Compiler Reference* for more information.

Automated program analysis and transformations

Significant performance improvements are possible with relatively little development effort because the compiler is capable of performing sophisticated program analysis and transformation of your program code. For example, the compiler can:

Automatically generate code overlays for the SPUs
Specifying **-qipa=overlay** instructs the compiler to automatically generate code overlays for the SPUs that allow two or more code segments to be loaded at the same physical address as they are needed. This feature lets developers create SPU programs that would otherwise be too large to fit in the local memory store of the SPUs. In addition, the compiler also provides the **-qipa=overlayproc** and **-qipa=nooverlayproc** compiler options to give developers direct control over generation of code overlays on specified procedures.

See Using automatic code overlays in the *XL C/C++ Optimization and Programming Guide* for more information.

Perform automatic SIMD vectorization of your program code

When the **-qhot=simd** compiler option is in effect, the compiler takes certain operations that are performed in a loop on successive elements of an array, and converts them into a call to a vector instruction. This call calculates several results at one time, which is faster than calculating each result sequentially. Applying this suboption is useful for applications with significant image processing demands. **-qhot=simd** is the default with **-O3** for the SPU.

Not all loops can be successfully vectorized. However, specifying the **-qreport** compiler option together with **-qhot=simd** will cause the compiler to generate diagnostic information that can help you improve the efficiency of your loops.

See **-qhot** and **-qreport** in the *XL C/C++ Compiler Reference* for more information.

Use interprocedural analysis (IPA) to optimize across program files

IPA can result in significant performance improvements. You can specify interprocedural analysis on the compile step only or on both compile and link steps in "whole program" mode . Whole program mode expands the scope of optimization to an entire program unit, which can be an executable or shared object.

See **-qipa** in the *XL C/C++ Compiler Reference* for more information.

Program optimization

XL C/C++ provides several compiler options that can help you control the optimization or performance of your programs.

With these options, you can perform the following tasks:

- Select different levels of compiler optimizations.
- Control optimizations for loops, floating point, and other types of operations.

XL C/C++ also provides optimization features specifically tailored to exploit the unique performance capabilities of Cell Broadband Engine processors, including specialized data types and highly optimized built-in functions that you can use in your application code to perform common computational needs.

Optimizing transformations can give your application better overall execution performance. C/C++ provides a portfolio of optimizing transformations tailored to various supported hardware. These transformations offer the following benefits:

- Reducing the number of instructions executed for critical operations
- Restructuring generated object code to make optimal use of the Cell Broadband Engine architecture
- Improving the usage of the memory subsystem

Note: For code targeting the SPU, we recommend compiling and linking with the **-O5** or **-qopt=5** compiler options to get the maximum performance from your application.

For more information, see these related topics:

- "Optimizing your applications" in the *XL C/C++ Optimization and Programming Guide*
- "Optimizing and tuning options" in the *XL C/C++ Compiler Reference*

- "Compiler built-in functions" in the *XL C/C++ Compiler Reference*
- To read an article about optimizing performance, search the Power Architecture technical library at www.ibm.com/developerworks/views/power/library.jsp for "cell broadband tips".

Diagnostic listings

The compiler output listing can provide important information to help you develop and debug your applications more efficiently.

Listing information is organized into optional sections that you can include or omit. For more information about the applicable compiler options and the listing itself, refer to "Compiler messages and listings" in the *XL C/C++ Compiler Reference*.

Symbolic debugger support

You can instruct XL C/C++ to include debugging information in your compiled objects. That information can be examined by the debuggers provided by the IBM Software Developer Kit for Multicore Acceleration V3.1 (SDK 3.1) to help you debug your programs.

Chapter 2. What's new for IBM XL C/C++ for Multicore Acceleration for Linux, V10.1

This section describes new added features and enhancements in IBM XL C/C++ for Multicore Acceleration for Linux, V10.1.

Operating system support

IBM XL C/C++ for Multicore Acceleration for Linux, V10.1 now supports Red Hat Enterprise Linux 5.2 (RHEL 5.2).

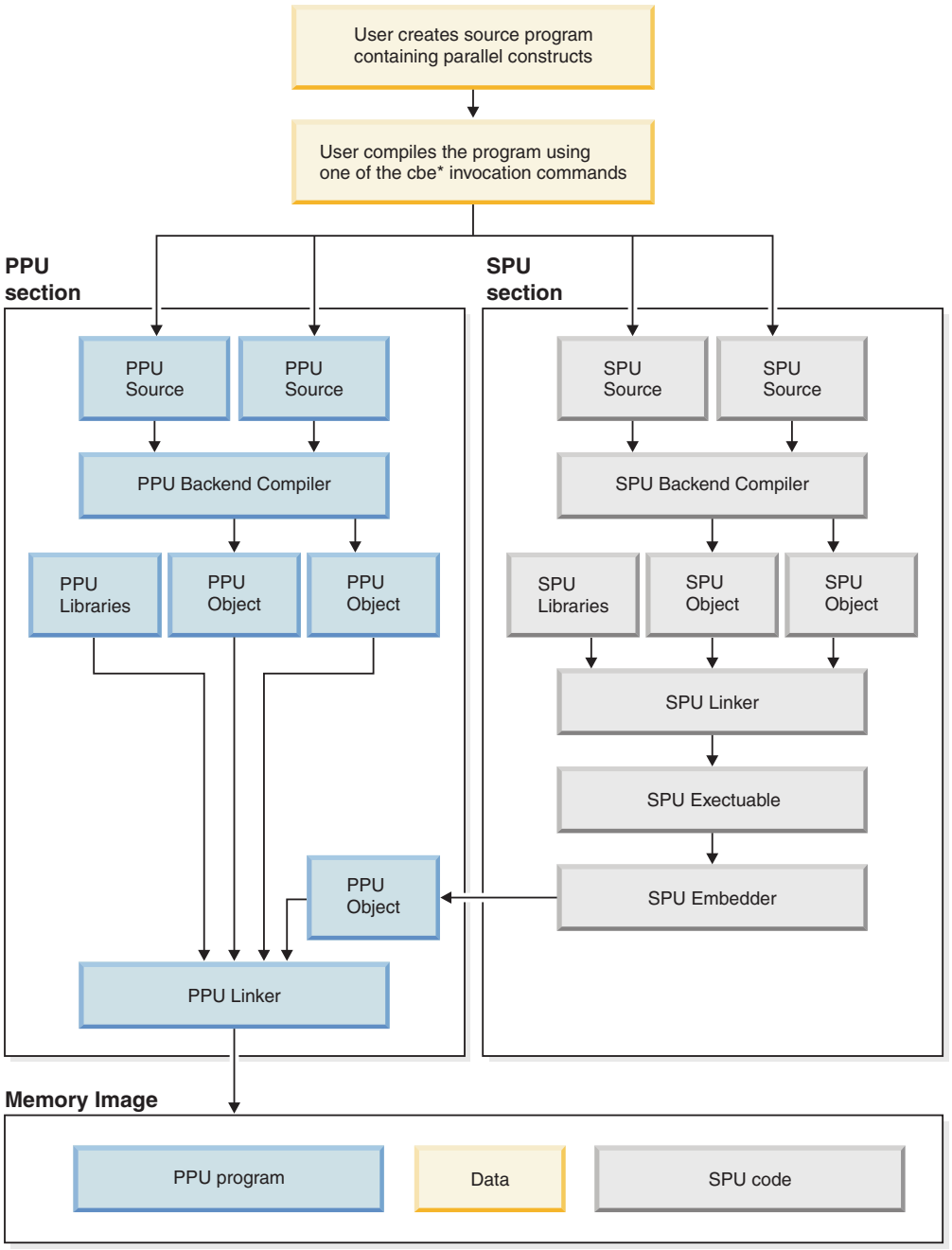
Single-source compiler technology

In this release, XL C/C++ adds another compiler invocation allowing programmers to include both PPU-specific and SPU-specific portions of their code in one application program.

Earlier compilers for the Cell Broadband Engine architecture are considered *dual-source* compilers. The compiler provides both PPU- and SPU-specific invocations to compile the different code segments. You write, compile, and link code segments destined to run on the PPU separately from code segments destined for the SPUs.

In contrast, a *single-source* compiler can compile and link both PPU and SPU code segments with a single compiler invocation. OpenMP API V2.5 directives can be used to program the portions of code targeting the SPU so that code destined for the PPU does not need to be written and compiled separately from code destined for the SPUs. You can compile and link PPU and SPU code segments together using a single compiler invocation.

You develop and compile your applications on an IBM Power System or Intel® x86 system running the RHEL 5.2 Linux operating system. When complete, you move your compiled application to a system based on the Cell Broadband Engine architecture, where that application will run.



For some background information on how the single-source compiler works to compile code optimized specifically for use on the Cell Broadband Engine architecture, see the *"Generation of Parallel Code"* section in the article *"Using advanced compiler technology to exploit the performance of the Cell Broadband Engine architecture"* which can be found online at http://www.research.ibm.com/journal/sj/451/eichenberger.html.

For more information, see:

- "Using OpenMP directives" in the *XL C/C++ Optimization and Programming Guide*
- "Invoking the compiler" on page 20
- www.openmp.org

C++0x

This release introduces support for a new version of the standard for the C++ programming language - specifically C++0x. This standard has not yet been officially adopted but we are beginning to support some of its features.

Specifically, in this release:

- we add a new language level
- we introduce new integer promotion rules for arithmetic conversions with `long long` data types
- the C++ preprocessor now supports C99 features

New language level - extended0x

The default **-qlanglvl** compiler option remains **extended** when invoking the C++ compiler.

A new suboption has been added to the **-qlanglvl** option in this release. **-qlanglvl=extended0x** is used to allow users to try out early implementations of any features of C++0x that are currently supported by XL C/C++.

C99 long long under C++

With this release of XL C/C++ V10.1, compiler behavior changes when performing certain arithmetic operations with integral literal data types. Specifically, the integer promotion rules have changed.

Previously, in C++ (and as an extension to C89), when compiling with **-qlonglong**, an unsuffixed integral literal would be promoted to the first type in this list into which it fitted:

```
int
long int
unsigned long int
long long int
unsigned long long
```

Starting with this release and when compiling with **-qlanglvl=extended0x**, the compiler now promotes unsuffixed integral literal to the first type in this list into which it fits:

```
int
long int
long long int
unsigned long long
```

Note: Like our implementation of the C99 Standard in the C compiler, C++ will allow promotions from `long long` to `unsigned long long` if a value cannot fit into a `long long` type, but can fit in an `unsigned long long`. In this case, a message will be generated.

The macro `__C99_LLONG` has been added for compatibility with C99. This macro is defined to 1 with **-qlanglvl=extended0x** and is otherwise undefined.

For more information, see ″Integral and floating-point promotions″ in the *XL C/C++ Language Reference*.

Preprocessor changes

The following changes to the C++ preprocessor make it easier to port code from C to C++:

- Regular string literals can now be concatenated with wide-string literals.
- The `#line <integer>` preprocessor directive has a larger upper limit. It has been increased from 32767 to 2147483647 for C++ .
- C++ now supports `_Pragma` operator.
- These macros now apply to C++ as well as C:
  - __C99_MACRO_WITH_VA_ARGS (also available with **-qlanglvl=extended**)
  - __C99_MAX_LINE_NUMBER (also available with **-qlanglvl=extended**)
  - __C99_PRAGMA_OPERATOR
  - __C99_MIXED_STRING_CONCAT

Note: Except as noted, these C++ preprocessor changes are only available when compiling with **-qlanglvl=extended0x**.

For additional information about the language standards supported by XL C/C++, see ″Language levels and extensions″ in the *XL C/C++ Language Reference*.

Other XL C/C++ language-related updates

Vector data types

Vector data types can now use some of the operators that can be used with base data types such as:.

- unary operators
- binary operators
- relational operators

__ea type qualifier (C only)

The `__ea` type qualifier support has been added to allow programs written for the SPU to access variables stored in the PPU address space. For additional information on using this type qualifier see, ″__ea type qualifier (C only)″ in the *XL C/C++ Language Reference*.

Performance and optimization

Some features and enhancements can assist with performance tuning and optimization of your application.

Enhancements to -qstrict

Many suboptions have been added to the **-qstrict** option to allow more fine-grained control over optimizations and transformations that violate strict program semantics. In previous releases, the **-qstrict** option disabled all transformations that violate strict program semantics. This is still the behavior if you use **-qstrict** without suboptions. Likewise, in previous releases **-qnostrict** allowed transformations that could change program semantics. Since higher level

of optimizations may require relaxing strict program semantics, the addition of the suboptions allow you to relax selected rules in order to get specific benefits of faster code without turning off all semantic verification.

There are 16 new suboptions that can be used separately or by using a suboption group. The groups are:

all Disables all semantics-changing transformations, including those controlled by the other suboptions.

ieeefp
Controls whether individual operations conform to IEEE 754 semantics. This is valid for code targeting the PPU or for double precison values on the SPU.

order Controls whether or not individual operations can be reordered in a way that may violate program language semantics.

precision
Controls optimizations and transformations that may affect the precision of program results.

exceptions
Controls optimizations and transformations that may affect the runtime exceptions generated by the program.

For detailed information about these suboptions, refer to ″-qstrict″ in the *XL C/C++ Compiler Reference*.

Performance-related compiler options and directives

The entries in the following table describe new or changed compiler options and directives.

Information presented here is a brief overview. For detailed information about these and other performance-related compiler options, refer to ″Optimization and tuning options″ in the *XL C/C++ Compiler Reference*.

Table 4. Performance-related compiler options and directives

Option/directive	Description
-qstrict	Many new suboptions have been added to give you more control over the relaxation of program semantic rules in order to gain some performance benefits.
-qreport	The listing now contains information about how many streams are created for each loop and which loops cannot be SIMD vectorized due to non-stride-one references. You can use this information to improve the performance of your applications.
-qsmp	**-qsmp** allows the programmer to write SPU portions of their program using OpenMP directives. For more information, see “Single-source compiler technology” on page 9.

For additional information about performance tuning and program optimization, refer to ″Optimizing your applications″ in the *XL C/C++ Optimization and Programming Guide*.

New or changed compiler options and directives

Compiler options can be specified on the command line or through directives embedded in your application source files. See the *XL C/C++ Compiler Reference* for detailed descriptions and usage information for these and other compiler options.

Table 5. New or changed compiler options and directives

Option/directive	Description
-qstrict	Many suboptions have been added to the **-qstrict** option to allow more control over optimizations and transformations that violate strict program semantics. See "Performance and optimization" on page 12 for more information.
-qcheck	You can now use the following suboptions for **-qcheck** for both SPU and PPU code: `bounds`, `divzero`, and `nullptr`.
-qshowmacros	When used in conjunction with the **-E** option, the **-qshowmacros** option replaces preprocessed output with macro definitions. There are suboptions provided to control the emissions of predefined and user-defined macros more precisely.
-qreport	When used together with compiler options that enable automatic parallelization or vectorization, the **-qreport** option now reports the number of streams in a loop and produces information when loops cannot be SIMD vectorized due to non-stride-one references.
-qsmp	**-qsmp** allows the programmer to write SPU portions of their program using OpenMP directives. For more information, see "Single-source compiler technology" on page 9.
-qtimestamps	This option can be used to remove timestamps from generated binaries.
-qtls (PPU only)	The thread local storage support has been enhanced to include `__attribute__((tls-model("`*string*`")))` where *string* is one of `local-exec`, `initial-exec`, `local-dynamic`, or `global-dynamic`.
-qinfo	The suboptions `als` and `noals` have been added to the **qinfo** option to report (or not report) possible violations of the ANSI aliasing rule.
-qea32, **-qea64** (SPU only)	The compiler option **-qea32** should be used if you are linking your SPU program to a 32-bit PPU object, and **-qea64** should be used if you are linking your SPU program to a 64-bit PPU object.
-qswcache_size (SPU only)	Effective address space support uses a software cache in combination with a cache manager library to maximize the speed with which variables stored in the PPE address space can be retrieved by the SPU. You can customize the size of the software cache by using the **-qswcache_size** compiler option.
-qsmallstack	You can reduce SPU stack size of your program by using the **-qsmallstack=size** option and compiling with a single-source compiler invocation, such as **cbexlc** or **cbexlc++**.

Table 5. New or changed compiler options and directives (continued)

Option/directive	Description
-qatomic_updates (SPU only)	Using **-qatomic_updates** ensures that the updates from SPU to PPU are done with no interleaving from other processors, so concurrent changes to adjacent bytes in the cache line are not lost. You can control how the PowerPC Processor Unit (PPU) memory is updated by the SPU software cache manager by using the -qatomic_updates and -qnoatomic_updates compiler options.

Chapter 3. Setting up and customizing XL C/C++

Setting up the IBM XL C/C++ for Multicore Acceleration for Linux, V10.1 compiler on your compilation host entails the following main steps:

1. Installing the IBM Software Developer Kit for Multicore Acceleration V3.1 (SDK 3.1) development tools on your compilation host.
2. Installing the XL C/C++ compiler and runtime environment on your compilation host.

Note: XL C/C++ is a cross compiler. The completed applications will run on BladeCenter® servers that contain processors built on the Cell Broadband Engine Architecture such as IBM BladeCenter QS21 and IBM BladeCenter QS22. You will also need to install the SDK 3.1 and the compiler runtime onto your execution host.

For complete prerequisite and installation information for XL C/C++, refer to "Before installing" in the *XL C/C++ Installation Guide*.

Chapter 4. Developing applications with XL C/C++

C/C++ application development consists of repeating cycles of editing, compiling, linking, and running.

Notes:

1. Before you can use the compiler, you must first ensure that XL C/C++ and the IBM Software Developer Kit for Multicore Acceleration V3.1 (SDK 3.1) are properly installed and configured. For more information see the *XL C/C++ Installation Guide*.
2. To learn about writing C/C++ programs, refer to the *XL C/C++ Language Reference*.

The compiler phases

A typical compiler invocation executes some or all of these activities in sequence. For link time optimizations, some activities will be executed more than once during a compilation. As each program runs, the results are sent to the next step in the sequence.

1. Preprocessing of source files
2. Compilation, which may consist of the following phases, depending on what compiler options are specified:
   a. Front-end parsing and semantic analysis
   b. High-level optimization
   c. Low-level optimization
   d. Register allocation
   e. Final assembly
3. Assemble the assembly (**.s**) files, and the unpreprocessed assembler (**.S**) files after they are preprocessed
4. Object linking to create an executable application

To see the compiler step through these phases, specify the **-v** compiler option when you compile your application. To see the amount of time the compiler spends in each phase, specify **-qphsinfo**.

Editing C/C++ source files

To create C/C++ source programs, you can use any text editor available to your system.

Source programs must be saved using a recognized file name suffix. See the "XL C/C++ input and output files" on page 23 for a list of suffixes recognized by XL C/C++.

For a C or C++ source program to be a valid program, it must conform to the language definitions specified in the *XL C/C++ Language Reference*.

Compiling with XL C/C++

XL C/C++ is a command-line compiler. Invocation commands and options can be selected according to the needs of a particular C/C++ application.

Compiling applications for a Cell/B.E. processor can involve multiple steps, depending on the complexity of your application. For a typical application containing code written for the PPU and the SPU in separate programs, you may need to:

1. Compile application code targeted to the PPU.
2. Compile application code targeted to the SPU, then embed the compiled SPU code into PPU code.
3. Perform the final PPU link.

For more information on this process, see "Embedding compiled SPU code into compiled PPU code" on page 24

If you have written the code targeting the SPU and PPU in the same program, using OpenMP pragmas for the SPU portions, you can compile and link the application in one step using one of the single-source compiler invocation commands, that is, those prefixed with **cbe** such as, **cbexlc** or **cbexlc++**.

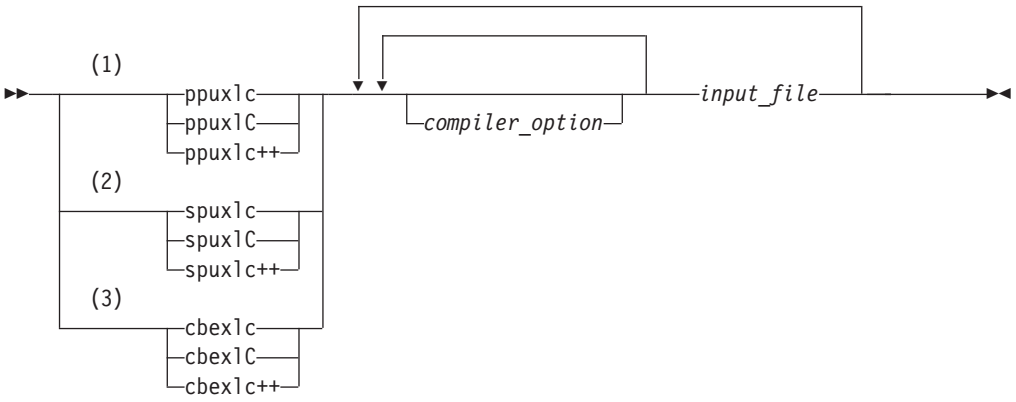
Invoking the compiler

The compiler invocation commands perform all necessary steps to compile C or C++ source files and link the object files and libraries into an executable program.

Compile your application code using a compiler invocation command appropriate to the type of code you are compiling. As with previous versions, XL C/C++ provides one set of compiler invocation commands for compiling programs targeting only the PPU and another set for compiling code to run only on the SPU application code. In this release, there is another set of invocation commands for compiling applications that contain code that has some portions targeting the PPU and that also has portions of the application that target the SPUs using OpenMP directives.

If you choose to create separate programs for PPU code and SPU code you must compile and link code targeted to the PPU using PPU-specific compiler invocations. Similarly, SPU-specific code must be compiled and linked using SPU-specific compiler invocations. You then must embed the SPU object in the PPU object (see "Embedding compiled SPU code into compiled PPU code" on page 24 for more information). Compiled PPU executable objects can be run on the execution host, and will load compiled SPU executable objects at run time as required.

To compile a source program, use the basic invocation syntax shown below:



Notes:

1 Basic invocations to compile C and C++ code targeting only the PPU.

2 Basic invocations to compile C and C++ targeting only the SPU.

3 Basic invocations to compile C and C++ programs containing both PPU and SPU code.

For new application work, you should compile with **ppuxlc**, **spuxlc**, **cbexlc**, **ppuxlc++**, **spuxlc++**, **cbexlc++**, or a thread safe counterpart. You can use **ppuxlc++**, **cbexlc++** or **spuxlc++** to compile either C or C++ program source, but compiling C++ files with **ppuxlc**, **cbexlc** or **spuxlc** may result in link or run time errors because libraries required for C++ code are not specified when the linker is called by the C compiler.

Additional invocation commands are available to meet specialized compilation needs, primarily to provide explicit compilation support for different levels and extensions of the C or C++ language. See "Invoking the compiler" in the *XL C/C++ Compiler Reference* for more information about compiler invocation commands available to you, including special invocations intended to assist developers migrating from a GNU compilation environment to XL C/C++.

Compiling parallelized XL C/C++ applications

XL C/C++ provides thread safe compiler invocation commands that you can use when compiling parallelized applications for use in multiprocessor environments.

These invocations are similar to their corresponding base compiler invocations, except that they link and bind compiled objects to thread safe components and libraries. The generic XL C/C++ thread safe compiler invocations include:

- cbexlc (for C programs where the SPU portions are written with OpenMP directives)
- cbexlc++, cbexlC (for C++ programs where the SPU portions are written with OpenMP directives)
- ppuxlc++_r, ppuxlC_r
- ppuxlc_r

XL C/C++ provides additional thread safe invocations to meet specific compilation requirements. See "Invoking the compiler" in the *XL C/C++ Compiler Reference* for more information.

For more information on parallelized applications see ″Parallelizing your programs″ in the *XL C/C++ Optimization and Programming Guide*.

Specifying compiler options

Compiler options perform a variety of functions, such as setting compiler characteristics, describing the object code to be produced, controlling the diagnostic messages emitted, and performing some preprocessor functions.

You can specify compiler options:

- On the command-line with command-line compiler options
- In your source code using directive statements
- In a makefile
- In the stanzas found in a compiler configuration file
- Or by using any combination of these techniques

It is possible for option conflicts and incompatibilities to occur when multiple compiler options are specified. To resolve these conflicts in a consistent fashion, the compiler usually applies the following general priority sequence to most options:

1. Directive statements in your source file *override* command-line settings
2. Command-line compiler option settings *override* configuration file settings
3. Configuration file settings *override* default settings

Generally, if the same compiler option is specified more than once on a command-line when invoking the compiler, the last option specified prevails.

Note: Some compiler options do not follow the priority sequence described above.

For example, the **-I** compiler option is a special case. The compiler searches any directories specified with **-I** in the vac.cfg file before it searches the directories specified with **-I** on the command-line. The option is cumulative rather than preemptive.

See the *XL C/C++ Compiler Reference* for more information about compiler options and their usage.

You can also pass compiler options to the linker, assembler, and preprocessor. See ″Compiler options reference″ in the *XL C/C++ Compiler Reference* for more information about compiler options and how to specify them.

Reusing GNU C/C++ compiler options with gxlc and gxlc++

XL C/C++ includes various features to help you transition from GNU C/C++ compilers to XL C/C++ including gxlc and gxlc++.

Each of the gxlc and gxlc++ utilities accepts GNU C or C++ compiler options and translates them into comparable XL C/C++ options. Both utilities use the XL C/C++ options to create an **xlc** or **xlc++** invocation command, which is then used to invoke the compiler. These utilities are provided to help you reuse makefiles created for applications previously developed with GNU C/C++. However, to fully exploit the capabilities of XL C/C++, you should use the XL C/C++ invocation commands and their associated options.

The actions of gxlc and gxlc++ are controlled by the configuration file `gxlc.cfg`. The GNU C/C++ options that have an XL C/C++ counterpart are shown in this

file. Not every GNU option has a corresponding XL C/C++ option. gxlc and gxlc++ return warnings for input options that were not translated.

The gxlc and gxlc++ option mappings are modifiable. For information on using the gxlc or gxlc++ configuration file, see "Reusing GNU C/C++ compiler options with gxlc and gxlc++" in the *XL C/C++ Compiler Reference* .

XL C/C++ input and output files

These file types are recognized by XL C/C++.

For detailed information about these and additional file types used by the compiler, see "Types of input files" in the *XL C/C++ Compiler Reference* and "Types of output files" in the *XL C/C++ Compiler Reference*.

Table 6. Input file types

Filename extension	Description
.c	C source files
.C, .cc, .cp, .cpp, .cxx, .c++	C++ source files
.i	Preprocessed source files
.o	Object files
.s	Assembler files
.S	Unpreprocessed assembler files

Table 7. Output file types

Filename extension	Description
a.out	Default name for executable file created by the compiler
.d	Make dependency file
.i	Preprocessed source files
.lst	Listing files
.o	Object files
.s	Assembler files

Linking your compiled applications with XL C/C++

By default, you do not need to do anything special to link an XL C/C++ program. The compiler invocation commands automatically call the linker to produce an executable output file.

For example, running the following command:

```
ppuxlc++ file1.C file2.o file3.C
```

compiles `file1.C` and `file3.C` to produce the object files `file1.o` and `file3.o`, then all object files (including `file2.o`) are submitted to the linker to produce one executable.

Compiling and linking in separate steps

To produce object files that can be linked later, use the **-c** option.

```
ppuxlc++ -c file1.C              # Produce one object file (file1.o)
ppuxlc++ -c file2.C file3.C      # Or multiple object files (file1.o, file3.o)
ppuxlc++ file1.o file2.o file3.o  # Link object files with default libraries
```

It is usually best to execute the linker through the compiler invocation command, because it passes additional **ppu-ld** or **spu-ld** options and library names to the linker automatically.

For more information about compiling and linking your programs, see:

- "Linking" in the *XL C/C++ Compiler Reference*
- Documentation provided with the IBM Software Developer Kit for Multicore Acceleration V3.1 (SDK 3.1)

Embedding compiled SPU code into compiled PPU code

If you have written separate program segments for PPU- and SPU-targeted code, instead of using OpenMP directives for the SPU portions of your program, you need to embed the SPU object file into the PPU object file.

The following string of compiler commands shows how you might compile an application with both PPU and SPU program segments, and then embed the compiled SPU application code into the compiled PPU application code.

```
spuxlc++ -c spu_file.C
spuxlc++ -o spu_file spu_file.o
ppu-embedspu -m32 spu_file spu_file spu_file-embed.o
ppuxlc++ -c ppu_file.C
ppuxlc++ ppu_file.o spu_file-embed.o
```

XL C/C++ compiler diagnostic aids

XL C/C++ issues diagnostic messages when it encounters problems compiling your application. You can use these messages and other information provided in compiler output listings to help identify and correct such problems.

For more information about listing, diagnostics, and related compiler options that can help you resolve problems with your application, see the following topics in the *XL C/C++ Compiler Reference*:

- "Compiler messages and listings"
- "Error checking and debugging options"
- "Listings, messages, and compiler information options"

Debugging compiled applications

You can use a symbolic debugger to debug applications compiled with XL C/C++.

Specifying the **-g** or **-qlinedebug** compiler options at compile time instructs the XL C/C++ compiler to include debugging information in compiled output. For more information debugging options, see "Error checking and debugging" in the *XL C/C++ Compiler Reference*.

You can then use any symbolic debugger to step through and inspect the behavior of your compiled application.

Optimized applications pose special challenges when debugging. When debugging highly optimized applications, you should consider using the **-qoptdebug** compiler option. For more information about optimizing your code, see "Optimizing your applications" in the *XL C/C++ Optimization and Programming Guide*.

Notices

This information was developed for products and services offered in the U.S.A. IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:
INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

Lab Director
IBM Canada Ltd. Laboratory
8200 Warden Avenue
Markham, Ontario L6G 1C7
Canada

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrates programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. You may copy, modify, and distribute these sample programs in any form without payment to IBM for the purposes of developing, using, marketing, or distributing application programs conforming to IBM's application programming interfaces.

Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows:

© (your company name) (year). Portions of this code are derived from IBM Corp. Sample Programs. © Copyright IBM Corp. 1998, 2008. All rights reserved.

Trademarks and service marks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. These and other IBM trademarked terms are marked on their first occurrence in this information with the appropriate symbol (® or ™), indicating US registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. A complete and current list of IBM trademarks is available on the Web at http://www.ibm.com/legal/copytrade.shtml.

Adobe, the Adobe logo, PostScript, and the PostScript logo are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, and/or other countries.

Intel is a trademark or registered trademark of Intel Corporation or its subsidiaries in the United States and other countries.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

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

Cell Broadband Engine is a trademark of Sony Computer Entertainment, Inc. in the United States, other countries, or both and is used under license therefrom.

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

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

Index

Special characters

A

B

C

D

E

F

G

I

L

M

O

P

R

S

T

U

V

X

Program Number: 5724-T42 & 5724-T43

Printed in USA

GC23-8572-00