IBM XL C/C++ compilers features

April 2018

Chapter 1. Introduction

This paper details what's new in the following members of the IBM® XL C/C++ compiler family: IBM XL C for AIX®, XL C/C++ for AIX, and XL C/C++ for Linux.

Compiler features vary slightly by operating system platform, and platform-specific features are described in the appropriate sections. The IBM XL C/C++ compilers share the common features described below unless otherwise noted.

IBM XL C for AIX, XL C/C++ for AIX, and XL C/C++ for Linux are part of a multiplatform XL compiler family derived from a common code base optimized to run on IBM Power Architecture®. These are industry leading optimizing compiler products that support IBM Power® systems. IBM XL C/C++ fully exploits POWER6®, POWER7®, POWER8®, and POWER9 architectures.

For more information about the benchmarks for IBM Power Systems™, see:

- www.ibm.com/systems/power (IBM Power Systems)
- www.spec.org
- www.tpc.org/tpcc/

Starting from V13.1.1, IBM XL C/C++ for Linux for little endian distributions runs on IBM Power Systems and is based on the Clang open source framework. The compiler combines the Clang front-end infrastructure with the advanced optimization technology from IBM in the compiler back end. It enables straightforward migration of applications to IBM Power Systems while delivering optimal performance.

IBM XL C/C++ for Linux, V13.1.6 for little endian distributions supports the following new features and enhancements:

- Additional OpenMP 4.5 features
- OpenMP Technical Reports support
- Enhancements of offloading computations to the NVIDIA GPUs
- POWER9 technology support
- Debugging enhancements
- Additional POWER9 built-in functions
- Additional compiler options and pragma directives

Note that the version that follows XL C/C++ for Linux, V13.1.6 is V16.1. IBM XL C/C++ for Linux, V16.1 for little endian distributions enhances the support for OpenMP 4.5 standard.

You can run XL C/C++ for Linux, V16.1 for little endian distributions on the following Linux little endian distributions for POWER9 technology based servers.

- Ubuntu Server 16.04
- Ubuntu Server 18.04
- SUSE Linux Enterprise Server 12 (SLES 12)
- SUSE Linux Enterprise Server 12 Service Pack 3 (SLES 12 SP3)
- Red Hat Enterprise Linux 7.3 (RHEL 7.3)

- Red Hat Enterprise Linux 7.4 (RHEL 7.4)
- Red Hat Enterprise Linux 7.4 for Power Little Endian (POWER9)
- Red Hat Enterprise Linux 7.5 (RHEL 7.5)
- Red Hat Enterprise Linux 7.5 for Power Little Endian (POWER9)
- Community Enterprise Operating System 7 (CentOS 7)

To compile and link programs that contain code to be offloaded to the NVIDIA GPUs with IBM XL C/C++ for Linux, V16.1, you must ensure the following operating system, hardware, and software requirements are met.

- Use any IBM Power Systems™ server that has one or more NVIDIA GPUs installed and is supported by your Linux operating system distribution and the NVIDIA CUDA Toolkit.
- Use a system that satisfies the installation requirements of the CUDA Toolkit. See the NVIDIA CUDA Toolkit website for more information.
- Install CUDA Toolkit 9.2.

For more information, see: https://www.ibm.com/us-en/marketplace/xl-cpp-linux-compiler-power.

IBM XL C for AIX, V13.1.3 and XL C/C++ for AIX, V13.1.3 support IBM Power Systems capable of running following operating systems:

- IBM AIX V6.1 TL 2 Service Pack 5 or later
- AIX V7.1
- AIX V7.2
- IBM i V7.1 PASE V7.1
- IBM i V7.2 PASE V7.2

For more information, see https://www.ibm.com/us-en/marketplace/xl-c-aix-compiler-power and https://www.ibm.com/us-en/marketplace/xl-cpp-aix-compiler-power.

Other members of the IBM XL C/C++ compiler family include XL C/C++ for Linux on z Systems™, XL C/C++ for z/VM®, and z/OS® XL C/C++, which are not in the scope of this whitepaper.

- **XL C/C++ for Linux on z Systems**

  IBM XL C/C++ for Linux on z Systems is an advanced, high-performance compiler that can be used for developing complex, computationally intensive C/C++ programs for Linux on z Systems. For more information, see: http://ibm.biz/xlcpp-loz.
- **XL C/C++ for z/VM**

  IBM XL C/C++ for z/VM is an advanced optimizing compiler for the z/VM platform. For more information, see: https://www.ibm.com/us-en/marketplace/xl-cpp-compiler-zvm.
- **z/OS XL C/C++**

  The XL C/C++ compiler feature for z/OS is an optional priced feature of the z/OS operating system. For more information, see: https://www.ibm.com/us-en/marketplace/xl-cpp-compiler-zos.

IBM XL C/C++ compilers support C and C++ international standards and industry specifications, facilitating application portability across hardware platforms and operating systems. The compilers support a large array of common language features.

On Linux, the increased compatibility with GNU C/C++ gives you the flexibility to build different parts of your application with either the IBM or GNU compiler, and still bind the parts together into a single application. One common use of this functionality is to build an application with IBM XL C/C++ that interacts with the GNU-built dynamic libraries, without recompiling the library source code. Applications built with this functionality can integrate with GNU assembler, and also provide full support for debugging through **gdb**, the GNU debugger.

IBM XL C/C++ compilers on AIX and Linux also offer support for the IBM XL Fortran compilers on AIX and Linux through interlanguage calls.

IBM XL C/C++ offers developers the opportunity to create and optimize 32-bit and 64-bit applications for the AIX and big endian Linux platforms, and 64-bit applications for the little endian Linux platform. On operating systems and architectures supporting the VMX instruction set, the IBM XL C/C++ compilers allow you to take advantage of the AltiVec programming model and APIs. They also allow you to improve the performance of your data and CPU intensive applications by exploiting the cutting edge IBM XL C/C++ automatic SIMD vectorization technology.

Information on IBM XL C and XL C/C++ compilers is available at:

https://www.ibm.com/us-en/marketplace/ibm-c-and-c-plus-plus-compiler-family

Chapter 2. Standards conformance

IBM XL compilers strive to maximize the performance of scientific, technical, and commercial applications on server platforms. Multiple operating system availability ensures cross-platform portability, augmented by standards compliance.

Starting from XL C for AIX, V13.1, XL C/C++ for AIX, V13.1, and XL C/C++ for Linux, V13.1, conformance to the following language standards and industry specifications is provided:

- C89, C99, and selected features of the C11 standard
- C++98, C++03, and selected features of the C++11 standard
- Full support for OpenMP V3.1
- Partial support for OpenMP V4.0
- AltiVec
- IEEE POSIX 1003.2

Supported C11 features

- `_Noreturn` function specifier
- Anonymous structures (C only)
- Anonymous unions (C only)
- Complex type initializations
- Generic selection (C only)
- Static assertions
- `typedef` redeclarations

Supported C++11 features (C++ only)

- Auto type deduction
- C99 `long long`
- C99 preprocessor features adopted in C++11
- Decltype
- Defaulted and deleted functions
- Delegating constructors
- Explicit conversion operators
- Explicit instantiation declarations
- Extended friend declarations
- Forward declaration of enumerations
- Generalized constant expressions
- Inline namespace definitions
- `nullptr`
- Reference collapsing
- Right angle brackets
- Rvalue references
- Scoped enumerations
- `static_assert`
- Trailing comma allowed in `enum` declarations
- Trailing return type

- Variadic templates

Starting from XL C/C++ for Linux, V13.1.6 for little endian distributions, the compiler conforms to the following language standards and industry specifications:

- C89, C99, and C11 standards
- C++98, C++03, C++11 and selected features of the C++14 standard
- Full support for OpenMP V3.1
- Partial support for OpenMP V4.0
- Partial support for OpenMP V4.5
- AltiVec
- IEEE POSIX 1003.2

Supported C++14 features

- Binary integer literals
- Digit separators
- Polymorphic lambda expressions
- Relaxing constraints on `constexpr` functions
- Return type deduction for normal functions
- The `deprecated` attribute
- Variable templates

Chapter 3. Key features

Offloading computations to the NVIDIA GPUs

The combination of the IBM POWER® processors and the NVIDIA GPUs provides a platform for heterogeneous high-performance computing that can run several technical computing workloads efficiently. The computational capability is built on top of massively parallel and multithreaded cores within the NVIDIA GPUs and the IBM POWER processors. You can offload parallel operations within applications, such as data analysis or high-performance computing workloads, to GPUs.

Programming with supported OpenMP 4.5 device constructs

Starting from XL C/C++ for Linux, V13.1.5 for little endian distributions, you can offload compute-intensive parts of an application and associated data to the NVIDIA GPUs by using the supported device constructs. For more information, see "OpenMP support" on page 8.

You must specify the **-qoffload** option to enable the support for offloading OpenMP target regions to NVIDIA GPUs. For **-qoffload** to take effect, you must also specify the **-qsmp** option to enable support for OpenMP target regions.

Starting from XL C/C++ for Linux, V13.1.6 for little endian distributions and XL Fortran for Linux, V15.1.6 for little endian distributions, you can use the **-qtgtarch** option to specify the real or virtual GPU architectures where the code can run, overriding the default GPU architecture. This allows the compiler to take maximum advantage of the capabilities and machine instructions which are specific to a GPU architecture, or common to a virtual architecture.

Starting from XL C/C++ for Linux, V13.1.6 for little endian distributions, you can use the **XLSMPOPTS=target={mandatory | default | disabled}** environment variable to control which device to execute target regions on. You can also use the supported runtime functions, for example, to query the target environment or to manage device memory.

Using the compiler with NVCC

The NVIDIA CUDA C++ compiler (NVCC) from the NVIDIA CUDA Toolkit partitions C/C++ source code into host and device portions. Starting from XL C/C++ for Linux, V13.1.5 for little endian distributions, you can use the compiler as the host compiler for the POWER processor with NVCC. For more information, see the *NVIDIA CUDA on IBM POWER8: Technical overview, software installation, and application development* downloadable from http://www.redbooks.ibm.com/redpapers/pdfs/redp5169.pdf.

System prerequisites

To compile and link programs that contain code to be offloaded to the NVIDIA GPUs with XL C/C++ for Linux, V13.1.5 or higher for little endian distributions, you must ensure the operating system, hardware, and software requirements are met. For more information, see the *XL C/C++ Installation Guide*.

Parallel programming

IBM XL C for AIX, IBM XL C/C++ for AIX, and IBM XL C/C++ for Linux compilers provide parallel programming through AltiVec/VMX, OpenMP, automatic parallelization, and autosimdization.

OpenMP support

Starting from V13.1.3, XL C and XL C/C++ compilers include full support for the OpenMP API V3.1 specification and partial support for the OpenMP API V4.0 and OpenMP API V4.5 specifications for shared memory parallel programming as follows:

- OpenMP V4.0 features
  - Atomic update, atomic capture, and atomic swap
  - The **OMP_DISPLAY_ENV** environment variable
  - The `omp_get_proc_bind` function (Linux for little endian distributions only)
  - The **OMP_PLACES** environment variable (Linux for little endian distributions only)
- OpenMP V4.5 functions (Linux for little endian distributions only)
  - `omp_get_num_places`
  - `omp_get_partition_num_places`
  - `omp_get_partition_place_nums`
  - `omp_get_place_num_procs`
  - `omp_get_place_proc_ids`
  - `omp_get_place_num`
- Environment variables that are extended to control the thread affinity policy (Linux for little endian distributions only)
  - **OMP_DYNAMIC**
  - **OMP_DISPLAY_ENV**
  - **OMP_PROC_BIND**
  - **OMP_THREAD_LIMIT**

Starting from XL C/C++ for Linux, V13.1.5 for little endian distributions, the compiler provides the following support for the OpenMP API V4.5. Some features are useful for offloading computations to the NVIDIA GPUs.

- New directives
  - **omp declare target**
  - **omp distribute**
  - **omp distribute parallel for**
  - **omp target**
  - **omp target data**
  - **omp target enter data**
  - **omp target exit data**
  - **omp target update**
  - **omp teams**
  - Combined constructs
- New functions
  - OpenMP execution environment functions
    - `omp_get_default_device`

    - `omp_get_initial_device`
    - `omp_get_num_devices`
    - `omp_get_num_teams`
    - `omp_get_team_num`
    - `omp_is_initial_device`
    - `omp_set_default_device`
  - OpenMP device memory functions
    - `omp_target_alloc`
    - `omp_target_associate_ptr`
    - `omp_target_disassociate_ptr`
    - `omp_target_free`
    - `omp_target_is_present`
    - `omp_target_memcpy`
- New environment variables
  - **OMP_DEFAULT_DEVICE = *n***
  - **XLSMPOPTS = TARGET = {MANDATORY | OPTIONAL | DISABLE}**

Starting from XL C/C++ for Linux, V13.1.6 for little endian distributions, the compiler adds the following support for the OpenMP API V4.5.

- New directives
  - **omp simd**
  - **omp for simd**
  - **omp distribute simd**
  - **omp distribute parallel for simd**
  - Combined constructs
- Updated directives
  - **omp ordered**
  - **omp target**
  - **omp target data**
  - **omp target enter data**
  - **omp target exit data**
  - **omp target update**
  - **omp task**
- Updated functions
  - `omp_target_alloc`
- Other enhancements

  You can map a lambda functor in OpenMP target regions.

Starting from XL C/C++ for Linux, V13.1.6 for little endian distributions, the compiler adds the following support for OpenMP Technical Reports.

- Implicit declare target
- Pointer attachment for aggregate members

Starting from XL C/C++ for Linux, V13.1.6 for little endian distributions, the compiler provides support for OpenMP interoperability with CUDA C/C++ and CUDA Fortran.

- You can call kernels written in CUDA C/C++ or CUDA Fortran in your OpenMP programs from the host.
- You can use the OpenMP **use_device_ptr** clause to pass OpenMP mapped variables to CUDA kernels that are launched from the host.
- You can use the OpenMP **is_device_ptr** clause to access CUDA device attribute variables or to pass device addresses directly to target regions.

XL C/C++ for Linux, V16.1 for little endian distributions enhances the support for OpenMP API V4.5 as follows:

- New directives
  - **omp cancel**
  - **omp cancellation point**
  - **omp declare simd**
  - **omp taskgroup**
  - **omp taskloop**
  - **omp taskloop simd**
- Updated directives
  - **omp task**
- New functions
  - `omp_get_cancellation`
- Other enhancements

  You can use the OMP_CANCELLATION environment variable to enable or disable the cancellation model.

OpenMP provides a simple and flexible interface for parallel application development. OpenMP is comprised of three components: compiler directives, runtime library functions, and environment variables. Applications that conform to the OpenMP specification are easily ported to other platforms from desktop to super computer that support the specification.

OpenMP supports applications that run both as parallel programs (multiple threads of execution and a full OpenMP support library) and as sequential programs (directives are ignored and a stub library is linked).

For information about using OpenMP for application parallelization, refer to the Chapter "Parallelizing your programs", *Optimization and Programming Guide* for XL C and C/C++ for AIX and Linux compilers.

For information about OpenMP runtime library functions, see the OpenMP Application Program Interface specification at www.openmp.org.

Thread-local storage (TLS)

TLS has been included in IBM XL C/C++ for Linux since V8.0. Starting from V10.1, IBM XL C for AIX and XL C/C++ for AIX support thread-specific variables through the thread-local storage (TLS) feature.

In multithreaded applications, it might be useful to maintain thread-specific static data. Thread-local storage is a GNU extension for this purpose. TLS is adapted by many vendors and is similar to using the POSIX **getthreadspecific** and **setthreadspecific** functions, but TLS may allow for better performance.

TLS is enabled by the **__thread** storage class specifier, or the **threadprivate** directive in OpenMP. **-qtls** compiler flag enables recognition of the **__thread** storage class specifier. Thread-local variables are static lifetime memory variables with a separate storage location for each thread. Use of thread-local storage prevents unintended sharing of static data between threads. A simple example demonstrating a practical use of thread-local storage is the C error code variable **errno**.

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.

AltiVec support

IBM XL C/C++ for AIX and IBM XL C/C++ for Linux support the AltiVec programming model through non-orthogonal language extensions. These language extensions can be used on operating systems and hardware supporting the VMX instruction set. The IBM implementation of the AltiVec Programming Interface specification is an extended syntax that allows type qualifiers and storage class specifiers to precede the keyword `vector` (or alternately, `__vector`) in a declaration.

Although not strictly required by the AltiVec Programming Interface specification, the `vector` keyword is recognized in a declaration context only when used as a type specifier (and when you compile the application with **-qaltivec**). The other AltiVec keywords, `pixel` and `bool` (for C), are recognized as valid type specifiers only when used in a vector declaration context. This approach has an important advantage: it allows your application to continue to use *vector* and *pixel* as variables and function names. To ensure maximum portability, use the underscore versions of the specifiers `vector` and `pixel` (`__vector` and `__pixel`) in declarations. For Altivec data types, the name mangling schema that XL C/C++ for Linux, V13.1.1 for little endian distributions uses is binary compatible with a new name mangling schema that is specified in the GNU C compiler with the **-fabi-version=4** option.

VMX support is available on XL C/C++ V12.1 or later AIX compilers where the target environment is running AIX V5.3, AIX V6.1, AIX 7.1, and AIX 7.2 on architectures that support the Single Instruction Multiple Data (SIMD) instruction set.

Shared memory parallelization

XL C/C++ supports application development for multiprocessor system architectures. You can use any of the following methods to develop your parallelized applications with XL C/C++:

- Directive-based shared memory parallelization (OpenMP, SMP)
- Instructing the compiler to automatically generate shared memory parallelization
- Message passing based shared or distributed memory parallelization (MPI)
- POSIX threads (Pthreads) parallelization
- Low-level UNIX parallelization using **fork()** and **exec()**

The parallel programming facilities are based on the concept of threads. Parallel programming exploits the advantages of multiprocessor systems, while maintaining a full binary compatibility with existing uniprocessor systems. This means that a multithreaded program that works on a uniprocessor system can take

advantage of a multiprocessor system without recompiling. For more information, see "Parallelizing your programs" in the *XL C/C++ Optimization and Programming Guide.*

POWER9 technology exploitation

Starting from XL C/C++ for Linux, V13.1.5 for little endian distributions, the compiler introduces the following support for the POWER9 technology.

- Compiler options to target the POWER9 architecture

  The **-mcpu=pwr9** or **-mcpu=power9 (-qarch=pwr9)** suboption enables the compiler to generate code that exploits new POWER9 instructions, which improve the program performance on the POWER9 architecture. With the **-mtune=pwr9** or **-mtune=power9 (-qtune=pwr9)** suboption, optimizations are tuned for the POWER9 architecture.
- Built-in functions for the POWER9 architecture
- MASS libraries

  The vector library `libmassvp9.a` and the SIMD library `libmass_simdp9.a` that contain functions tuned for the POWER9 architecture.

Optimization capabilities

One of the key strengths of IBM XL C/C++ is optimization. These compilers offer the benefit of optimization technology that has been evolving at IBM since the late 1980s, combining extensive hardware knowledge with a comprehensive understanding of compiler technology and what users look for in a compiler when building end-user applications. The optimizations can decrease execution time and make your applications run faster, producing code that is highly tuned for execution on Power Architecture platforms. Improving optimization is a key goal of the IBM compiler team, and one that will continue to be a major focus with each iteration of the IBM XL C/C++ compilers.

The optimizer includes five base optimization levels: **-O0**, **-O2**, **-O3**, **-O4**, and **-O5**. These levels allow you to choose from minimal optimization to intense program analysis that provides benefits even across programming languages. Optimization analyses range from local basic block to subprogram to file-level to whole-program analysis. The higher the optimization level, the more intense the program analysis becomes as increasingly sophisticated optimization techniques are applied to your code.

At each optimization level, the optimizer performs transformations that result in performance improvements, while still executing your code the way it was written. At higher levels, the optimizer can trade numeric precision for execution speed. If this effect is not desired, you can specify compiler options such as **-qstrict** to prevent such trade-offs. You can use other options such as **-qsmallstack** or **-qcompact** to bias optimization decisions in favor of smaller stack space or program size.

The IBM XL C/C++ compilers do not limit your optimization choices unnecessarily. All of the optimization capabilities, including those discussed above, can be combined. You choose the levels and types of optimizations best suited to your application and build constraints, putting ultimate control of how your application builds and runs firmly in your hands.

For more information on optimization, see the *Code Optimization with the IBM XL Compilers on Power architectures* whitepaper at www.ibm.com/support/docview.wss?uid=swg27005174 and the IBM XL C/C++ *Optimization and Programming Guide*.

Decimal floating-point support for XL C/C++

Decimal floating point arithmetic offers greater computational performance and precision in business and financial applications where numeric data I/O is usually performed in decimal form. Data conversions from decimal type to binary floating-point type and back are avoided, as there are inherent rounding errors accumulated during data conversions.

Table 1. Decimal floating-point compiler options

Options	Description
-qdfp | **-qnodfp**	Specifying **-qdfp** enables compiler support for decimal floating-point data types and literals.
-qfloat= dfpemulate | **nodfpemulate**	Specifying **-qfloat=dfpemulate** instructs the compiler to use software emulation when handling decimal floating-point computations.
-y	There are suboptions specific to decimal floating-point arithmetic for the **-y** option to control rounding of constant expressions.

Notes:

- Compiler support for decimal floating-point operations on Linux requires Advanced Tool Chain 7.0 or higher.
- XL C/C++ for Linux for little endian distributions does not support decimal floating-point.

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, see "Compiler messages and listings" in the *XL C/C++ Compiler Reference*.

Enhanced Unicode and NLS support

As recommended by the C Standard committee, the C compiler extends C99 to add new data types to support UTF-16 and UTF-32 literals. The data types are *u-literals* and *U-literals*. To enable support for UTF literals in your source code, you must compile with the option **-qutf** enabled. The C++ compiler also supports these new data types for compatibility with C.

Note: The **-qutf** option is available only on AIX and Linux for big endian distributions.

IBM is a corporate member of the Unicode Consortium. For more information about Unicode, see www.unicode.org.

Boost C++ library support

Boost C++ libraries are open source libraries that take you beyond the C++ Standard Library. Boost makes C++ programming more elegant, robust, and productive. The Boost license grants permission to copy, use, and modify the software for any commercial or non-commercial use. With the non-restrictive licensing, these libraries are used directly by many commercial applications. Many of the libraries are planned for inclusion in the next version of the C++ Standard Library.

Boost libraries allow you to be more productive through software reuse. The ability to compile and execute the Boost Libraries properly demonstrates IBM's support of the latest C++ idioms and paradigms, specifically **generic programming** and **template metaprogramming**.

Boost C++ libraries are coded by the leading C++ experts in the world, many of whom are long time members of the C++ Standard Committee. They use Boost as a test bed for cutting edge C++ programming techniques and codify discoveries and best practices without the long delay that it takes for a library to be formally accepted into the C++ Standard. However, the Boost community subjects each submission to rigorous peer review. This free sharing of knowledge, exposes a submission to a larger audience which helps C++ evolve and grow.

The IBM XL C/C++ compilers have attained a high degree of compatibility with Boost and continue to support Boost as new releases appear. Each version of the compiler is fully tested on one version of Boost, usually the latest. The following table shows the Boost support in each version of the compiler.

Table 2. IBM XL C++ compiler and Boost version supported

IBM XL C++ Compiler Version	Boost version supported
16.1 (Linux for little endian distributions only)	1.59.0
13.1.6 (Linux for little endian distributions only)	1.59.0
13.1.5 (Linux for little endian distributions only)	1.59.0
13.1.4 (Linux for little endian distributions only)	1.59.0
13.1.3 (Linux for little endian distributions only)	1.59.0
13.1.3 (AIX)	1.55.0
13.1.2	1.55.0
13.1.1 (Linux for little endian distributions only)	1.55.0
13.1	1.55.0
12.1	1.47.0
11.1	1.40.0

Patch files are available that modifies the Boost C++ libraries so that they can be built and used with XL C/C++ applications. The patch or modification file does not extend or otherwise provide additional functionality to the Boost C++ libraries. To download the patch file, see:

http://www.ibm.com/support/docview.wss?uid=swg27006911

The above link also provides information on Boost library regression test results.

For more information on portable C++ source libraries from Boost, see:

www.boost.org

IBM Mathematical Acceleration Subsystem (MASS) libraries

IBM XL C/C++ compilers ship the IBM Mathematical Acceleration Subsystem (MASS) libraries of mathematical intrinsic functions specifically tuned for optimum performance on the IBM Power architecture. The MASS libraries are thread-safe, include scalar, SIMD, and vector functions, and offer improved performance over the equivalent intrinsic functions in the standard math library provided with your AIX or Linux on POWER. The MASS functions can be used with either Fortran or C/C++ applications.

A version of the MASS scalar library is provided for each operating system: AIX, Linux on POWER big endian, and Linux on POWER little endian. Vector and SIMD versions of the MASS libraries are provided, tuned for POWER7 (AIX and Linux big endian) processors, POWER8 (AIX, Linux big endian, and Linux little endian) processors, and POWER9 (Linux little endian) architecture. These support programs running on the specific processor or higher.

The MASS SIMD and vector libraries allow you to compute mathematical functions for multiple inputs with (often significant) performance gains over repeated calls to a scalar function. SIMD MASS functions input and output a vector data type whose size matches the size of the supported processor's vector registers, while vector MASS supports arguments that are vectors of arbitrary length.

Basic Linear Algebra Subprograms (BLAS)

There are four BLAS high-performance algebraic functions shipped with IBM XL C/C++ in the `libxlopt` library. The functions are as follows:

- `sgemv` (single-precision) and `dgemv` (double-precision), which compute the matrix-vector product for a general matrix or its transpose.
- `sgemm` (single-precision) and `dgemm` (double-precision), which perform combined matrix multiplication and addition for general matrices or their transposes.

Because the BLAS routines are written in Fortran, all parameters are passed to them by reference, and all arrays are stored in column-major order.

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.

C++ templates

Templates are an area of the C++ language that provides a great deal of flexibility for developers. The ISO C++ standard defines the language facilities and features for templates.

The IBM XL C++ compiler provides several methods to compile templates:

- Simple layout method. This results in code bloat and longer compile time, but it is easy to use and requires no specific structuring by programmers.
- Automatic instantiation using **-qtempinc**. This requires user code structuring but it addresses the long compile time problem inherent in the simple layout method.
- Automatic instantiation using **-qtemplateregistry**. This requires no user code restructuring and addresses both the long compile time and code bloat issues.

The instantiation mechanisms are the external mechanisms that allow C++ implementations to create instantiations correctly. These mechanisms may be constrained by requirements of the linker and other software building tools.

IBM XL C++ compilers have two queried instantiation mechanisms, **-qtempinc** and **-qtemplateregistry**. One of the differences between **-qtempinc** and **-qtemplateregistry** is that **-qtempinc** delays the instantiation until link time, and the **-qtemplateregistry** does the instantiation in the first compilation unit that uses it.

-qtempinc and **-qtemplateregistry** compiler options are mutually exclusive.

Here is how you get the various instantiation models for the XL C++ compiler:

Greedy instantiation
default is **-qtmplinst=auto -qnotemplateregistry -qnotempinc** or **-qtmplinst=always**

Queried instantiation
-qtemplateregistry or **-qtempinc** (for example **-qtmplinst=auto**)

Manual instantiation
-qtmplinst=none with explicit instantiations in your code.

Notes:

- XL C/C++ for Linux for little endian distributions supports only the greedy instantiation model.
- XL C/C++ for Linux for little endian distributions does not support the **-qtempinc**, **-qtemplateregistry**, **-qtmplinst=auto**, or **-qtmplinst=always** option.

Utilization tracking and reporting

On Linux for big endian distributions and AIX, the utilization tracking and reporting feature is a lightweight and simple mechanism for tracking the compiler utilization within your organization. It is disabled by default. You can use this feature to detect whether your organization's use of the compiler exceeds your compiler license entitlements.

When utilization tracking is enabled, each invocation of the compiler is recorded in a compiler utilization file. You can run the utilization reporting tool to generate a report from one or more of these files to get a picture of the overall usage of the compiler within your organization. The **urt** command can be used to control how the report is generated. In particular, the report indicates whether the compiler usage complies with the number of Concurrent User licenses that you have purchased.

The utilization tracking and reporting feature is easy to set up and manage, and utilization tracking does not impact the usage or performance of the compiler.

On Linux for little endian distributions and AIX, you can also enable IBM Software License Metric (SLM) Tags logging in the compiler so that IBM License Metric Tool (ILMT) can track compiler license usage.

For detailed information about the utilization tracking and reporting feature, see the topics about tracking compiler usage in the *XL C/C++ Compiler Reference*.

Chapter 4. Compatibility and porting

Porting from open source and other platforms

The cross-platform portability of GNU C and GNU C++ has ensured GNU a place in the open source community. GNU has excelled in educational and compiler research arenas as a test bed for new language syntax. IBM XL compilers are built on a platform of reliability, customer service, and cutting-edge optimization. In recent years, the IBM XL compilers have been evolving to gain some of the additional flexibility and portability of the GNU compilers, while still retaining the strengths that have built the IBM XL C/C++ compiler's reputation in the industry.

GNU source compatibility

XL C/C++ for Linux for little endian distributions is built with Clang front end and IBM optimizing back end components. It provides improved GCC compatibility and language standards support for easier migration and enhanced capability as well as the IBM optimization technology.

On Linux platform, the compilers use the GNU C and C++ headers, and the resulting application is linked with the C and C++ runtime libraries provided by the GNU compiler shipped with the operating system. IBM ships an implementation of some header files with the compiler product to override the corresponding GNU header files. These header files are functionally equivalent to the corresponding GNU implementation. Other IBM headers are wrappers that include the corresponding GNU header files.

XL C/C++ supports a subset of the GNU compiler command options to facilitate porting applications developed with GNU C and GNU C++ compilers. Where possible, the XL C/C++ compiler maps GNU options to their XL C/C++ compiler option counterparts before invoking the XL C/C++ 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 might have. This support is available when the **gxlc** or **gxlc++** invocation command is used together with select GNU compiler options on AIX. Starting from V13.1.1, XL C/C++ for Linux for little endian distributions provides a greater level of GNU source compatibility. It supports the use of **gcc** and **g++** compiler options and therefore the **gxlc** and **gxlc++** invocation commands are not required or included.

GNU binary compatibility

The IBM XL C/C++ for Linux compilers achieve a high degree of binary compatibility with GNU-built objects, archives, and shared objects. The compiler achieves this by adhering to the system ABI and calling conventions, and by closely following the GNU behavior where alignment modifiers like the attributes aligned and packed are used.

C++ interoperability is somewhat more difficult to achieve due to differing conventions for name mangling, object model, and exception handling. However, the GNU C++ compiler, since V3.2, has adopted a common vendor C++ ABI that defines a way to allow interoperability of C++ object model, name mangling, and exception handling. This common C++ ABI is supported in the IBM XL C++

compilers. IBM XL C/C++ for Linux, V13.1 for big endian distributions has been fully tested with GNU C/C++ 4.4.7 on RHEL 6.4, GNU C/C++ 4.8.2 on RHEL 7.0, and GNU C/C++ 4.3.4 on SLES 11, and offers a high degree of binary compatibility in addition to source compatibility. IBM XL C/C++ for Linux, V16.1 for little endian distributions has been fully tested with the following versions:

- GNU C/C++ 4.8.3 on SLES 12
- GNU C/C++ 4.8.5 on RHEL 7.3
- GNU C/C++ 4.8.5 on RHEL 7.4
- GNU C/C++ 4.8.5 on RHEL 7.5
- GNU C/C++ 4.8.5 on SLES 12 SP3
- GNU C/C++ 5.3.1 on Ubuntu 16.04
- GNU C/C++ 7.3 on Ubuntu 18.04

The XL C++ compiler for Linux also has an option to display the class layouts, the virtual function tables entries as well as all the intermediate object model tables such as the construction virtual function table, and the virtual function table. These help you to ensure binary compatibility through verification of internal table layouts, and significantly enhance the debugging of incompatibility problems.

Command-line compatibility utilities

When you are porting GNU makefiles to IBM XL C/C++ for Linux, the **gxlc** and **gxlc++** invocation commands are available to translate a GNU compiler invocation command into the corresponding XL C/C++ for Linux command where applicable, and invoke the XL C/C++ for Linux compiler. This facilitates the transition to XL C/C++ for Linux while minimizing the number of changes to makefiles built with a GNU compiler.

Note: Starting from V13.1.1, XL C/C++ for Linux for little endian distributions supports the use of **gcc** and **g++** compiler options and therefore the **gxlc** and **gxlc++** command-line compatibility utilities are not required or included.

To fully exploit the capabilities of IBM XL C/C++, you should use the XL C/C++ invocation commands and their associated options.

Support for third-party C++ runtime libraries

The IBM XL C++ compiler on AIX can compile C++ applications so that the application supports only the core language, thus enabling it to link with C++ runtime libraries from third-party vendors. The following archive files enable this functionality.

Table 3. Core Language libraries

Library name	Content
lib*C*core.a	Contains exception handling, RTTI, static initialization, new and delete operators. Does not contain any of the following libraries: Input/Output, Localization, STL Containers, Iterators, Algorithms, Numerics, Strings.
libCcore.a	The core language version of the C++ runtime library, libC.a.
libC128core.a	The core language version of libC128.a.
libhCcore.a	The core language version of libhC.a.

Invocation commands have been added to facilitate using these libraries:

- xlc++core
- xlCcore

Equivalent special invocations:

- xlc++core_r, xlc++core_r7, xlc++core128, xlc++core128_r, xlc++core128_r7
- xlCcore_r, xlCcore_r7, xlC128core, xlC128core_r, xlC128core_r7

Explanation of suffixes for special invocations:

- **128-**suffixed invocations - All **128-**suffixed invocation commands are functionally similar to their corresponding base compiler invocations. They specify the **-qldbl128** option, which increases the length of `long double` types in your program from 64 to 128 bits. They also link with the 128-bit versions of the C and C++ runtime libraries.
- **_r** suffixed invocations - All **_r** suffixed invocations allow for threadsafe compilation and you can use them to link the programs that use multithreading. Use these commands if you want to create threaded applications.

The **_r7** invocations are provided to help migrate programs based on POSIX Draft 7 to POSIX Draft 10.

Compatibility of redistributable library libxlopt.a

The `libxlopt.a` library is compatible with XL C/C++ for AIX, V11.1, or later versions.

You can download and use the latest redistributable library for multiple applications compiled with XL C/C++ for AIX, V11.1 and later versions, on supported platforms.

For more information about the redistributable libraries, see "Redistributable libraries" in the *XL C/C++ Compiler Reference*.

Chapter 5. Tools supporting IBM XL C/C++ compilers

Debugging capabilities

You can instruct IBM XL C/C++ compilers to include debugging information in your compiled objects. The debugging information can be examined by any symbolic debugger to help you debug your programs.

For debugging capability on AIX, you have the choice of any symbolic debugger that supports the AIX XCOFF executable format including dbx, TotalView, DDT, and IBM Debugger for AIX. On Linux, you can use debuggers including DDT, gdb, or TotalView. TotalView also supports debugging OpenMP applications.

Additionally, the compilers offer the SNAPSHOT directive and the **-qoptdebug** and **-qkeepparm** options to assist you in debugging optimized code.

Note: The **-qoptdebug** option is supported by XL C for AIX, XL C/C++ for AIX, and XL C/C++ Linux for big endian distributions only.

Rational PurifyPlus

IBM Rational® PurifyPlus is a dynamic software analysis solution designed to help developers write faster, more reliable code. It includes the following capabilities packaged into a single product:

Memory debugging
Pinpoints memory errors that are hard to find, such as uninitialized memory access, buffer overflow, and improper freeing of memory.

Memory leak detection
Identifies memory blocks that no longer have a valid pointer.

Performance profiling
Highlights application performance bottlenecks and improves application understanding with a graphical representation of function calls.

Code coverage
Identifies untested code with line-level precision.

Rational PurifyPlus is supported on Windows, Linux, Solaris, and AIX. Rational PurifyPlus for Linux and UNIX supports AIX, Linux, and Solaris.

Note: Rational PurifyPlus supports XL C/C++ for AIX and XL C/C++ for Linux for big endian distributions.

For more information, see:

http://www.ibm.com/software/products/en/purifyplus

Rational Team Concert

IBM Rational Team Concert™ is a lean collaborative lifecycle management solution. It helps companies build better software and products with a complete lean development environment for teams, which includes complete agile, formal and hybrid planning and reporting, all on a common platform. Rational Team Concert

also supports geographically distributed teams with features such as communication in context, event feeds, integrated chat, and automated traceability, all beyond simple e-mail. Rational Team Concert can be easily adopted by component, as a unified solution, or surrounding existing tooling. Operating systems supported by Rational Team Concert include AIX, IBM i, Linux, Intel, Microsoft, Windows, and z/OS.

For more information, see:

http://www.ibm.com/software/rational/products/rtc

Chapter 6. Summary

IBM XL C/C++ compilers are stable and flexible, providing industry leading optimization techniques that can address your compiler needs for everything from small applications, to large, computationally intensive programs.

The extensive cross-platform availability of the IBM XL C/C++ compilers eases the porting process between AIX and Linux. Standards conformance and GNU compatibility improve portability of source code from GNU compilers to IBM XL compilers. On Linux, the binary compatibility feature allows direct linkage with objects, shared libraries, and archives built by either the GNU or IBM XL compilers. This allows you to take advantage of the features offered by both suites of compiler products.

IBM is also deeply involved in the High Performance Computing effort. Many of the TOP500 supercomputers are from IBM using IBM XL C/C++ compiler optimizations. The IBM XL C/C++ compiler team is deeply involved in parallel computing and supporting different parallel memory models. Other new features support customer requests and enable middleware applications. With XL C/C++ for Linux, V16.1 for little endian distributions, you can develop modern high-performance applications by exploiting the heterogeneous resources consisting of the IBM POWER processors and the NVIDIA GPUs.

Optimization through chip-specific instruction generation and tuning, parallelization, vectorization, interprocedural analysis, and profile-directed feedback offers an increase in performance without sacrificing stability or flexibility. Coupled with IBM's excellent service and support, IBM XL C/C++ compilers are robust, versatile, and capable of delivering mission critical applications.

Starting from V13.1.1, IBM also delivers XL C/C++ compilers for Linux for little endian distributions. XL C/C++ for Linux for little endian distributions leverages the Clang infrastructure from the open source community for a portion of its compiler front end. Clang is a component of the LLVM open source compiler and toolchain project and provides the C and C++ language family front end for LLVM. XL C/C++ combines the Clang front-end infrastructure with the advanced optimization technology in the IBM compiler back end. For additional information about Clang, see the LLVM website at: http://clang.llvm.org.

Chapter 7. Support information

Documentation

IBM XL C/C++ compiler product documentation is available online. You can access it from the following library page links:

- IBM XL C for AIX: http://www.ibm.com/support/docview.wss?uid=swg27036590
- IBM XL C/C++ for AIX: http://www.ibm.com/support/docview.wss?uid=swg27036618
- IBM XL C/C++ for Linux: http://www.ibm.com/support/docview.wss?uid=swg27036675
- IBM XL C/C++ for Linux on z Systems: http://www.ibm.com/support/docview.wss?uid=swg27044043
- IBM z/OS XL C/C++: http://www.ibm.com/support/docview.wss?uid=swg27036892
- IBM XL C/C++ for z/VM: http://www.ibm.com/support/docview.wss?uid=swg27037738

The IBM XL C/C++ compilers also include man pages for all utilities and compiler invocation commands.

An extensive collection of technical material and demos, support information, and features and benefits of IBM XL C/C++ can be found at the following URL:

https://www.ibm.com/us-en/marketplace/ibm-c-and-c-plus-plus-compiler-family

Premier customer service

The IBM XL C/C++ compilers come with IBM's premier service and support. The IBM Service and Support organization is made up of a team dedicated to providing you with responsive platform and cross-platform software support. For complex or code-related problems, IBM employs specialized service teams with access to compiler development experts. The vision of IBM Service and Support is to achieve a level of support excellence that exceeds customer expectations and differentiates IBM in the marketplace. You will always have access to the right level of IBM expertise when you need it. More information is available at:

http://www.ibm.com/support/entry/portal/support?brandind=rational

You can also find the latest updates for IBM XL C and XL C/C++ compilers at:

http://www.ibm.com/support/docview.wss?uid=swg21110831

Community Edition and purchasing

IBM XL C/C++ for Linux, V16.1 Community Edition for little endian distributions is available for download and deployment in April 2018. This compiler product is a no-charge, fully functional product for developers who do not require official IBM support.

The following link contains the XL C/C++ compiler product web pages. To download the Community Edition of the compilers, navigate to the download page.

https://www.ibm.com/us-en/marketplace/ibm-c-and-c-plus-plus-compiler-family

Purchasing information of IBM XL C/C++ is also available at the above website.

Contacting IBM

IBM welcomes your comments. You can send them to compinfo@cn.ibm.com.

Printed in USA