IBM XL C/C++ Compilers
Insights on Improving Your Application
Table of Contents

Purpose of this document .............................................................................................................................2
Overview .......................................................................................................................................................2
Performance..................................................................................................................................................2
Figure 1: ....................................................................................................................................................3
Figure 2: ....................................................................................................................................................4
Exploiting the z/OS XL C/C++ V1R12 Compiler ...........................................................................................5
Hints and Tips ...............................................................................................................................................7
Useful Links and Contacts ............................................................................................................................9
Purpose of this document
This document is for a technical C/C++ application programmer audience and is intended to explain some of the newest performance characteristics of the z/OS® XL C/C++ V1R12 compiler, an optionally priced feature of z/OS V1R12. This document is not meant to be a comprehensive guide to tuning applications and it does not cover runtime/ Language Environment related tuning either. For further information on tuning, refer to the additional links section.

Overview
IBM generally announces a new release of the z/OS XL C/C++ compiler with every release of z/OS. Each release of the compiler delivers a new set of features developed to address our clients’ needs, of which numerous enhancements are made specifically to improve performance. z/OS V1R12 and z/OS XL C/C++ V1R12 were announced July 22 and were generally available September 24, 2010. In this latest release, z/OS XL C/C++ introduces support for innovative advances in optimization technology as well as support for the IBM zEnterprise™ System 196 (z196). Implemented separately or together, these improvements can provide different performance characteristics for your applications. These improvements apply equally to both the C and C++ languages.

Please note that this document focuses on z/OS XL C/C++ compiler performance. Enterprise COBOL and Enterprise PL/I compilers may also benefit from later releases of IBM System z® hardware and software, but that is beyond the scope of this document. Please contact your IBM representative for technical information on improving COBOL and PL/I application performance.

Performance
Improving the performance of a program can be a complicated, challenging endeavor. Many factors can affect the running time or throughput of a given program, including network latency, disk latency, memory bandwidth constraints, runtime library dependencies and computational efficiency, to name just a few. This document focuses on performance improvements implemented to increase the computational efficiency delivered by the z/OS XL C/C++ V1R12 compiler for the zEnterprise system.

In order to measure the performance improvement delivered by the z/OS XL C/C++ V1R12 compiler, IBM assembled, for internal use, two suites of computationally intensive benchmarks. The applications in the first suite are characterized by a substantial use of integer arithmetic; the applications in the second suite engage primarily in floating point calculations. The benchmarks in both suites are designed to capture the behavior of a range of programs, and as such their response to any given compiler improvement can vary widely.
IBM reports the mean average improvement across these benchmarks. For z/OS XL C/C++ V1R12, IBM uses the compiler options described in the following section ("Exploiting the z/OS XL C/C++ V1R12 Compiler"). Depending on the options specified, or how similar or dissimilar a given program is to one of the programs in IBM’s benchmark suites, the performance improvements from the z/OS XL C/C++ compiler can, unsurprisingly, vary from the reported mean average. The improvement presented in Figures 1 and 2 are based on internal IBM lab measurements and results will vary from application to application.

Figure 1 illustrates not only the performance gains of the z196 relative to IBM System z10®, but also the contributions of the z/OS XL C/C++ V1R12 compiler on top of these gains, for both the integer and floating point IBM internal benchmarks. The boost from the z196 hardware alone on IBM’s internal benchmarks is impressive. Performance was improved by over 54% for the internal integer benchmarks, and over 120% for the internal floating point ones. On top of that, the z/OS XL C/C++ V1R12 compiler’s z196 exploitation improved performance by an additional 10% for the IBM internal benchmarks.

![Figure 1: Performance Improvement Due To z196 Hardware And z/OS V1R12 C/C++ Exploitation Of z196 Instructions](image-url)
Figure 2 summarizes the performance improvements available in the z/OS XL C/C++ V1R12 compiler as compared to the z/OS XL C/C++ V1R11 compiler, again for both the integer and floating point IBM internal benchmarks. Users upgrading from z/OS XL C/C++ compilers older than z/OS XL C/C++ V1R11 could see an even greater relative improvement after upgrading their z/OS XL C/C++ compiler.

![Figure 2: z/OS XL C/C++ Performance Improvement On z196 z/OS XL C/C++ V1R12 vs. V1R11](image)

- **V1R11**: z10 instr./tune - Executed on z196, compiled with V1R11 compiler using z10 instruction set tuned to improve performance on z10 (z196 instructions and tuning aren't available with V1R11 compiler.)
- **V1R12**: z196 instr./tune - Executed on z196, compiled with V1R12 compiler using z196 instructions set with special tuning to improve performance on z196

Reported improvements are based on internal IBM lab measurements of benchmarks compiled with options designed to maximize performance. Results achieved will vary from application to application, depending on a variety of factors including source code and compiler options specified. The reported improvement is the geometric mean of the results observed across a variety of benchmarks.
Exploiting the z/OS XL C/C++ V1R12 Compiler

There are many options available to tune an application for peak performance. The options used to build the benchmarks used in Figure 2 are listed below, followed by a brief description of each option and its potential value. In general, the highest level of optimization will produce the best overall result. However, individual cases vary so it is important to understand how the application might be best optimized when choosing compiler options to improve performance.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARCH(9)</td>
<td>Target hardware</td>
</tr>
<tr>
<td>TUNE(9)</td>
<td>Target micro-architecture</td>
</tr>
<tr>
<td>OPTIMIZE(3)</td>
<td>Optimization level</td>
</tr>
<tr>
<td>ILP32</td>
<td>31-bit addressing mode</td>
</tr>
<tr>
<td>XPLINK</td>
<td>eXtra Performance Linkage</td>
</tr>
<tr>
<td>HGPR</td>
<td>High-word GPR</td>
</tr>
<tr>
<td>HOT</td>
<td>High Order Transformation</td>
</tr>
<tr>
<td>PDF</td>
<td>Profile Directed Feedback</td>
</tr>
<tr>
<td>IPA</td>
<td>Intra-Procedural Analysis</td>
</tr>
</tbody>
</table>

ARCH(9)

The ARCH option determines which architectural features the z/OS XL C/C++ compiler can assume will be available in the hardware. For instance, zEnterprise introduces new instructions such as the conditional register move (LOCR); the z/OS XL C/C++ V1R12 compiler can utilize this powerful instruction if ARCH(9) is specified. Similarly, the prefetch (PFD) instruction is available on both z10 and zEnterprise systems; it may be used by the compiler if ARCH(8) or ARCH(9) is specified, but not if ARCH(7) or below is used.

ARCH(5) is the lowest supported architecture setting and should be used when the target hardware level is unknown. However, since each increase in ARCH level allows the z/OS XL C/C++ compiler to use increasingly powerful instructions, the highest level should be used where feasible. For zEnterprise systems, it is ARCH(9).

Note: programs will not run on a lower architecture processor than what is specified in the ARCH option.

TUNE(9)

While the ARCH option determines which instructions can be generated, the TUNE option determines how the instructions will be used. The z/OS XL C/C++ compiler may, for example, adjust the sequencing of generated instructions to allow them to flow through the processor with maximum efficiency. While the ARCH level must not exceed the oldest version of the hardware on which the program will execute, the TUNE level may be higher and will probably be most effective if set to match the hardware on which the program will most frequently run.
If there is no clear hardware preference, the highest available TUNE level should be used, i.e. TUNE(9).

**OPTIMIZE(3)**
There are three settings for the OPTimize option: 0, 2 or 3. OPTimize 0 disables all optimizations (as does –g, which implies OPT(0)). OPT(2) strikes a balance between aggressive optimization and compilation time. OPT(3) performs additional optimizations over OPT(2), with a greater emphasis on aggressive optimization over minimizing compilation time. The z/OS V1R12 XL C/C++ User’s Guide describes the differences between options in more detail. Using OPTimize 3 is recommended where possible.

**IP32**
31-bit programs may run faster than 64-bit programs particularly if they manipulate a lot of addresses, since there is half as much data to move. If the program needs more than 2 Gb of addressable space this option cannot be used.

**XPLINK**
XPLINK linkage is the default for 64-bit programs, but is optional for 31-bit. If at all possible, XPLINK should be used as it is significantly faster than the default linkage for 31-bit programs. However, there still exist many libraries built with non-XPLINK linkage. Calls between XPLINK and non-XPLINK code are more expensive than calls between code using the same linkage convention. If, therefore, non-XPLINK code is called frequently, XPLINK may not always provide the best overall performance.

**HGPR**
This option allows the use of 64-bit instructions in 31-bit mode, and is, therefore, only relevant with IP32. It almost always improves the performance of 31-bit code.

**HOT**
The HOT option enables aggressive loop optimization and may significantly improve performance, particularly if there are computationally intensive loops in the program. It also finds additional opportunities to use the ARCH(9) LOCR instruction.

**PDF**
Profile-Directed Feedback is a two pass process. In the first pass the compiler instruments the generated code so a training run of the application can detect which parts of the code are most frequently executed. In the second pass the compiler optimizes the code based on the profiling information from the training run. PDF benefits all ARCH levels and can provide significant performance improvements because the z/OS XL C/C++ compiler can aggressively optimize frequently executed paths through the program, providing maximum performance where it will count the most.
IPA
The IPA option enables inter-procedural analysis. There are three levels of IPA: 0, 1 and 2. Level 0 provides some reorganization of the layout of functions within the program, and can group hot procedures together to maximize use of the hardware's instruction cache. Level 1 enables cross-file inlining as well as a host of other optimizations. Level 2 enables very aggressive loop optimizations, similar to HOT, but with interprocedural knowledge. Level 2 can result in relatively longer link times, so level 1 is generally recommended unless there are computationally intensive loops in the program, in which case level 2 is preferred.

In addition to the performance improvements, recent releases of IBM z/OS XL C/C++ compilers include many new features such as Metal C RENT support, improved debug support, increased support of C++0x and reduced memory consumption.

Hints and Tips
The previous section summarized the options used to improve the performance of a suite of C/C++ benchmarks. This section offers suggestions of how best to incorporate these options into your C/C++ build environment. Note that the improvements shown in Figure 2 apply specifically to C/C++ code.

Generally speaking, the more aggressive (higher) the optimization level, the better the performance of the compiled C/C++ program. Because the z/OS XL C/C++ compiler is working harder, build times can take longer, so it is up to each user to determine the best tradeoff between increased application performance versus increased build time.

Additionally, more aggressive optimization by the z/OS XL C/C++ compiler can expose latent programming bugs. The most challenging problems involve misuse of the ANSI aliasing rules, which prohibit casting data of one type to a pointer of an incompatible type. If a program is ANSI compliant, the compiler has a much freer hand when rearranging the code, leading to substantially faster programs. If the compiler is told the source is compliant (via the ANSIALIAS option) when, in fact, the code violates the aliasing rules, the compiler may incorrectly move instructions, resulting in faulty code. If you are unsure whether your program is ANSI compliant, you should specify NOANSIALIAS, although this will restrict the z/OS XL C/C++ compiler and reduce the potential performance improvement.

If the program has never been compiled with IPA, PDF or HOT, it is recommended that the program is developed in the following phases:

Development Phase
Use aggressive ARCH/TUNE options (will not affect problem diagnosis)

Use XPLINK (will not affect problem diagnosis)
Use OPTIMIZE (2) and NOANSIALIAS, and if/when bugs appear, temporarily switch to OPTIMIZE(0) to debug them. Use ANSIALIAS if possible. If program works with OPT (2) and NOANSIALIAS, but fails with OPT(2) with default ANSIALIAS, then specify NOANSIALIAS, or run the z/OS XL C/C++ compiler with CHECKOUT(CAST)/INFO=ALS and fix any ANSI violations.

Use HGPR (will not affect problem diagnosis)

**Tuning Phase I**

Use OPTIMIZE(3)

Use ANSIALIAS if possible (enabled by default). If program works with OPT(3) and NOANSIALIAS, but fails with OPT(3) with default ANSIALIAS, then specify NOANSIALIAS (or fix any ANSI violations in your code)

See also CHECKOUT(CAST) and INFO=ALS.

If OPT(3) works, use HOT

**Tuning Phase II**

Assuming OPT(3) and HOT do not expose problems, compile with IPA level 0.

Once level 0 works, compile/link with PDF. If PDF is not desired, skip to Phase III

REPORT can help when diagnosing problems exposed by these optimization levels.

**Tuning Phase III**

If Phase II was skipped, start with IPA level 0 first. If no problems are exposed with Level 0, use Level 1. Level 1 provides better optimization, but takes longer than level 0. If not skipped, try PDF IPA with level 1.

If further optimization is required, experiment with level 2. Level 2 can significantly improve loop dominated programs, but does further lengthen the link phase. Determine if the improved performance over Level 1 justifies the increased link time. REPORT can help when diagnosing problems exposed by these optimization levels.
## Useful Links and Contacts

<table>
<thead>
<tr>
<th>Application</th>
<th>URL</th>
</tr>
</thead>
</table>
IBM XL C/C++ Compilers

Copyright IBM Corporation 2010
IBM Systems and Technology Group
Route 100
Somers, New York 10589
U.S.A.
Produced in the United States of America,
11/2010
All Rights Reserved

IBM, IBM logo, Redbooks, System z, zEnterprise and z/OS are trademarks or registered trademarks of the International Business Machines Corporation.

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

Performance is based on measurements and projections using standard IBM benchmarks in a controlled environment. The actual throughput that any user will experience will vary depending upon considerations such as the amount of multiprogramming in the user’s job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve throughput improvements equivalent to the performance ratios stated here.