

XL C/C++ Enterprise Edition for AIX



XL C/C++ 入门

版本 7.0

注意！

在使用本资料及其支持的产品之前，请确保阅读第 69 页的『声明』中的信息。

第一版（2004 年 9 月）

本版本适用于 XL C/C++ Enterprise Edition for AIX® 的 V7.0.0（产品号 5724-I11）及所有后续发行版和修订版，直到在新版本中另有声明为止。

IBM 欢迎您提出宝贵意见。您可通过因特网将它们发送至以下地址：

ctsrcf@cn.ibm.com

请包含本书的标题和订单号码以及与您意见相关的页码或主题。如果需要答复，请务必提供您的电子邮件地址。

当您发送信息给 IBM 后，即授予 IBM 非专有权，IBM 可以它认为合适的任何方式使用或分发此信息，而无须对您承担任何责任。

© Copyright International Business Machines Corporation 2004. All rights reserved.

目录

关于本书.	v
突出显示约定	v
如何阅读语法图	v

XL C/C++ 概述.	1
命令行 C 和 C++编译器	1
库	1
标准 C++ 库	2
IBM Mathematics Acceleration Subsystem 库	2
IBM Distributed Debugger	2
其它工具和实用程序	2
本地语言支持	3
文档和联机帮助	3

版本 7 中的新增内容	7
性能和优化	7
机器体系结构和硬件	7
用于 POWER5 处理器的新内置函数.	7
新增的 XL C/C++ 编译指示 (pragma)	9
新的优化实用程序.	9
IBM Mathematics Accelerated Subsystem (MASS) 库	9
SMP 线程绑定	10
遵循业界标准	10
易于使用	12
新增的 XL C/C++ 选项	13

定制编译环境	15
环境变量	15
为路径创建符号链接	15
配置文件	16

控制编译过程	17
调用编译器	17
对象模型	18
输入和输出文件的类型	18
缺省行为	19

编译器选项入门	21
编译器消息	21
返回码	22
编译器消息格式	22
通过 gxc 和 gxc++ 重用 GNU C 和 C++ 编译器选项	22
gxc 和 gxc++ 语法	23
GNU C 和 C++ 至 XL C/C++ 选项映射.	23
配置选项映射.	27
选项摘要: C 编译器	29
基本转换	30

特殊处理和控制	31
与链接和库相关的选项.	32
选项摘要: C++ 编译器	32

优化入门	33
选择的用于优化的编译器选项	33
优化 pragma 入门	35

移植注意事项	37
语言固有的可移植性问题	37
编译时错误的诊断	38
32 位和 64 位应用程序开发	39
AIX 上的 32 位开发环境和 64 位开发环境.	40
AIX 上的目标和库	41
AIX 上共享目标与库之间的差别	42
AIX 上共享目标与静态目标之间的差别	42
链接时和装入时	43
链接时错误的诊断	43
运行时错误的诊断	44
共享内存并行化	45
OpenMP 伪指令	46
AIX 上的线程	46
与 GNU C 和 C++ 可移植性相关的功能.	52
函数属性	53
变量属性	54
类型属性	54
GNU C 和 C++ 断言	55
与 GNU C 和 C++ 相关的其它扩展	55

附录 A. 语言支持	57
与 ISO/IEC 国际标准的兼容性	57
ISO/IEC 14882:2003(E) 国际标准兼容性	57
ISO/IEC 9899:1990 国际标准兼容性	57
ISO/IEC 9899:1999 国际标准支持	57
增强的语言级别支持	59

附录 B. OpenMP 一致性和支持	61
OpenMP 伪指令	61
OpenMP 数据作用域属性子句	63
OpenMP 库函数	63
OpenMP 环境变量	65
OpenMP 实现定义的行为	66
调整 OpenMP 程序.	66

声明	69
编程接口信息	70
商标和服务标记	71
业界标准	71

关于本书

XL C/C++ Enterprise Edition for AIX 是一个基于标准的、进行优化的命令行编译器，用于在 PowerPC 体系结构上运行的 AIX 操作系统。该编译器是一个专业编程工具，用于以扩展 C 和 C++ 编程语言创建和维护 32 位和 64 位应用程序。

本书向您介绍 XL C/C++ 编译器。它描述了各种编译器调用以及定制编译环境和控制编译过程的方法。本书包含编译器可以执行的转换类型的描述、输入和输出的可接受的文件类型、编译器选项的分类摘要和用于移植现有应用程序的注意事项。本书还简要介绍了如何优化应用程序的性能。编译器的优化能力使您能够利用 PowerPC 处理器的多层体系结构的方法。

AIX 和 Linux[®] 是互补的操作系统。为用 XL C/C++ 开发的应用程序创建的 makefile（制作文件）可以很容易地修改并移植到 Linux 平台。本书的目的是帮助您用 XL C/C++ 开发和维护程序并在编译、链接和运行时获得提高的性能。

本文档假定您熟悉 C 和 C++ 编程语言、AIX 操作系统和 ksh shell。

突出显示约定

粗体	标识命令、关键字和系统预定义了其名称的其它项。
<i>斜体</i>	标识其实际名称或值由程序员提供的参数。斜体还用于首次提及的新术语。
示例	标识特定数据值的示例、与可能看到的显示内容相似的文本的示例以及程序代码的某些部分、来自系统的消息或您实际应输入的信息的示例。

示例用于进行教学，并不尝试最大程度地减少运行时间、节约存储空间或检查错误。这些示例并不示范可能使用的所有语言构造。某些示例只是代码片段，如果不加上其它的代码，编译将不能进行。

如何阅读语法图

- 沿着线条的走向，从左至右、从上至下阅读语法图。

▶—— 符号指示命令、伪指令或语句的开始。

——▶ 符号指示命令、伪指令或语句语法在下一行继续。

▶—— 符号指示命令、伪指令或语句是从上一行继续的。

——▶◀ 符号指示命令、伪指令或语句结束。

语法单元图不同于完整的命令、伪指令和语句，它以 ▶—— 符号开始，以 ——▶ 符号结束。

注：在下图中，statement 表示 C 或 C++ 命令、伪指令或语句。

- 必需的项出现在水平线上（主路径）。
▶▶—statement—*required_item*—▶▶

- 可选的项出现在主路径之下。
▶▶—statement—
 └─*optional_item*—

- 如果可以从两个或更多项中进行选择，则将它们垂直堆叠在一起。
如果必须选择这些项的其中一项，则堆叠中有一项出现在主路径上。
▶▶—statement—*required_choice1*—▶▶
 └─*required_choice2*—

如果选择这些项目之一是可选的，则整个堆叠显示在主路径之下。

▶▶—statement—
 └─*optional_choice1*—
 └─*optional_choice2*—

缺省项出现在主路径之上。

▶▶—statement—
 └─*default_item*—
 └─*alternate_item*—

- 主线上向左边返回的箭头指示可以重复的项。
▶▶—statement—
 └─*repeatable_item*—

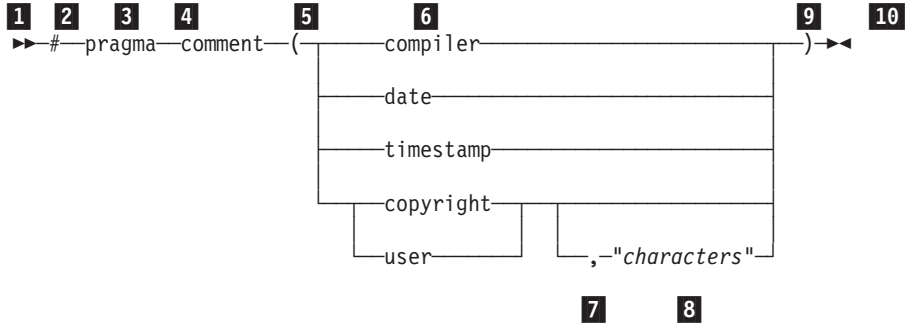
堆叠上方的重复箭头指示您可以从堆叠的项中选择多项或重复单个选项。

- 关键字以非斜体字母的形式出现，应严格按所显示的输入（例如：extern）。

变量以斜体小写字母的形式出现（例如：*identifier*）。它们表示用户提供的名称或值。

- 如果显示了标点符号、圆括号、算术运算符或其它这样的符号，则必须将它们作为语法的一部分输入。

以下语法图示例显示了 **#pragma comment** 伪指令的语法。有关 **#pragma** 伪指令的信息，请参阅 *XL C/C++ Language Reference*。



- 1 这是语法图的开始。
- 2 符号 # 必须首先出现。
- 3 关键字 pragma 必须跟在 # 符号之后。

- 4 编译指示的名称 `comment` 必须跟在关键字 `pragma` 之后。
- 5 左圆括号必须存在。
- 6 输入的注释类型必须仅为以下所指示的类型之一: `compiler`、`date`、`timestamp`、`copyright` 或 `user`。
- 7 逗号必须出现在注释类型 `copyright` 或 `user` 与可选字符串之间。
- 8 字符串必须跟在逗号之后。必须用双引号将字符串括起。
- 9 需要右圆括号。
- 10 这是语法图的结束。

按照以上显示的图表, **#pragma comment** 伪指令的下列示例在语法上是正确的:

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

XL C/C++ 概述

IBM XL C/C++ Enterprise Edition V7.0 是一个基于标准的、进行优化的命令行编译器，用于在 PowerPC 体系结构上运行的 AIX 操作系统。该编译器是一个专业编程工具，用于以扩展 C 和 C++ 编程语言创建和维护 32 位和 64 位应用程序。编译器代表一种成熟技术，它的发展重点在于性能和平台间可移植性并且包含取自其它平台上的发行版的灵活性、功能和改进。

本产品是 IBM VisualAge C++ Professional for AIX V7.0 后续发行版。IBM 已将 VisualAge C++ 品牌更改为 XL C/C++。

除了编译器本身之外，XL C/C++ 附带提供了可帮助您高效地创建程序的库、实用程序和工具。编译器文档包括了有关编译器调用以及所有命令行实用程序的可搜索帮助系统、PDF 书籍和手册页。

命令行 C 和 C++编译器

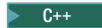
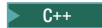
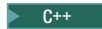
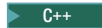
XL C/C++ 提供了一些基本编译器调用命令以供选择，这些命令支持 C 和 C++ 语言的各种版本级别。每个调用命令自动为语言级别、其它相关语言功能的选项以及任何相关的预定义宏设置编译器子选项。在大多数情况下，应使用 **xlc** 命令来编译 C 源文件，使用 **xlc** 命令来编译 C++ 源文件（或同时有 C 和 C++ 源文件时）。

提供了基本命令的变体以支持特殊环境和文件系统的需求。变体通过将后缀附加至基本命令组成。例如，使用具有后缀 **_r** 的命令确保编译器使用可重入且线程安全版本的函数和库。

另外，**gxlc** 和 **gxlc++** 实用程序是专门的编译器调用形式。

库

XL C/C++ 附带提供了下列库。

- “SMP 运行时库”支持显式并行处理和自动并行处理。
- 已调整的数学内部函数的 IBM Mathematics Acceleration Subsystem (MASS) 库（32 位方式和 64 位方式）。
- “内存调试运行时”用于诊断内存泄漏。
-  “标准 C++ 库”（包括“标准 C 库”和“标准模板库”）可以用来创建符合“标准 C++”的代码。
-  “C++ 运行时库”包含编译器所需要的支持例程。
-  “USL 复数数学类库”包含用于处理复数的类。提供此库是为了供旧的应用程序使用。对于新的应用程序，应使用“标准 C++ 库”。
-  “UNIX 系统实验 (USL) I/O 流类库”包含 C++ 的输入和输出功能的流类。提供此库是为了供旧的应用程序使用。对于新的应用程序，应使用“标准 C++ 库”。

-  拆分程序库提供用于拆分 C++ 编译器创建的链接名称的例程和类。

标准 C++ 库

XL C/C++ Enterprise Edition for AIX 附带提供了修改后版本的“Dinkum C++ 库”，该库是“标准 C++ 库”的一致实现。“标准 C++ 库”由 51 个头文件组成，包括构成“标准模板库”（STL）的 13 个头文件。此外，“标准 C++ 库”与“标准 C 库”中的 18 个头文件一起工作。这些头文件中的函数执行基本的服务，如输入和输出。它们也用于高效地实现频繁使用的操作。

相关参考

- *Standard C++ Library Reference* 中的 C++ Library Overview

IBM Mathematics Acceleration Subsystem 库

从版本 7 开始，XL C/C++ 附带提供了已调整的数学内部函数的 IBM Mathematics Acceleration Subsystem (MASS) 库（32 位方式和 64 位方式）。MASS 库都是线程安全的并通过相应的 libm 例程提高了性能。此外，不需要更改代码就可以使用 MASS 库。

IBM Distributed Debugger

XL C/C++ 附带提供了 IBM Distributed Debugger，后者是一个客户机 / 服务器应用程序，该程序可用于检测和诊断程序中的错误。该客户机 / 服务器设计使得可以调试在能通过网络连接访问的系统上运行的程序和工作站上的程序。

调试器服务器（也称为调试引擎）与要调试的程序在同一系统上运行。此系统可以是工作站或能通过网络访问的系统。如果调试在工作站上运行的程序，则执行本地调试。如果调试在能通过网络连接访问的系统上运行的程序，则执行远程调试。

Distributed Debugger 客户机是一个图形用户界面，可以在该界面发出调试引擎所使用的命令来控制程序的执行。例如，可以设置断点、单步执行代码和检查变量的内容。Distributed Debugger 用户界面可用于从单个调试器会话调试多个应用程序，这些应用程序可能是用不同语言编写的。调试的每个程序显示在独立的程序页面中。显示的信息的类型取决于所连接的调试引擎。

缺省情况下，将 Distributed Debugger 安装到 /usr/idebug 目录。要启动 Distributed Debugger，可在命令行输入 idebug。

其它工具和实用程序

CreateExportList 命令

创建一个文件，它包含在给定的一组目标文件中找到的所有全局符号的列表。

c++filt 名称拆分实用程序

当 XL C/C++ 编译 C++ 程序时，它编码所有函数名称和某些其它标识符来包括类型和作用域信息。此编码过程称为混成。此实用程序将混成的名称转换为其原始的源代码名称。

linkxlc 命令

链接 C++ .o 和 .a 文件。在未安装 XL C/C++ 编译器的系统上使用此命令进行链接。

makeC++SharedLib 命令

允许在未安装 XL C/C++ 编译器的系统上创建 C++ 共享库。

cleanpdf 命令

与基于概要的反馈相关的命令，用于管理 PDFDIR 目录。从指定的目录、PDFDIR 目录或当前目录除去所有概要分析信息。

mergepdf 命令

与基于概要的反馈（PDF）相关的命令，在将两个或多个 PDF 记录合并为一个记录时，它提供了权衡它们的重要性的能力。PDF 记录必须从相同的可执行文件派生。

resetpdf 命令

resetpdf 命令的当前行为与 cleanpdf 命令相同，保留它是为了与其它平台上的较早发行版兼容。

showpdf 命令

显示在基于概要的反馈练习运行（在选项 -qpdf1 和 -qshowpdf 下的编译）中执行的所有过程的调用和块计数的命令。

gxlc 和 gxlc++ 实用程序

一些调用方法，它们将 GNU C 或 GNU C++ 调用命令转换为对应的 **xlc** 或 **xlc** 命令并调用 XL C/C++ 编译器。这些实用程序的目的是减少对用 GNU 编译器构建的现有应用程序的 makefile 的更改并使得向 XL C/C++ 的过渡更方便。

相关参考

- *XL C/C++ Programming Guide* 中的 “CreateExportList Command”
- *XL C/C++ Programming Guide* 中的 “c++filt Name Demangling Utility”
- *XL C/C++ Programming Guide* 中的 “linkxlc Command”
- *XL C/C++ Programming Guide* 中的 “makeC++SharedLib Command”

本地语言支持

XL C/C++ 支持 Unicode 标准、多字节字符、UTF-16 和 UTF-32 字符串文字、多个装入的语言环境和双向性。这些功能使创建国际应用程序成为可能或更加容易。

相关参考

- *XL C/C++ Language Reference* 中的 “The Unicode Standard”
- *XL C/C++ Compiler Reference* 中的 “National Languages Support”

文档和联机帮助

XL C/C++ Enterprise Edition for AIX 提供以下形式的产品文档：

- 自述文件。
- 可安装的手册页。
- 基于 HTML 的可搜索帮助系统。

- PDF 文档。

可以按如下方式找到或访问这些项:

自述文件	自述文件位于 <code>/usr/vacpp/</code> 目录和安装 CD 的根目录中。
手册页	为编译器调用和随产品提供的所有命令行实用程序提供了手册页。
基于 HTML 的帮助系统	可搜索帮助系统称为“信息中心”，由 HTML 文件组成。帮助系统可安装在专用内部网上或可在产品 Web 站点上联机查看。
PDF 文档	PDF 文件位于 <code>/usr/vacpp/doc/\$LANG/pdf</code> 目录中。它们可通过 Adobe Acrobat Reader 来查看和打印。如果您没有安装 Adobe Acrobat Reader，则您可从 http://www.adobe.com 下载它。

完整的 XL C/C++ PDF 文档库由以下文件组成:

install.pdf

XL C/C++ Installation Guide 包含关于安装编译器、启用手册页和安装可搜索 HTML 帮助系统的指示信息。

getstart.pdf

《XL C/C++ 入门》包含 XL C/C++ 组件的概述、新功能的说明、有关定制编译环境和进程的方法信息、按类别排列的编译器选项的摘要表、对性能优化和调整的简介和将应用程序移植到 AIX 平台的一般建议。

language.pdf

XL C/C++ Language Reference 包含关于 C 和 C++ 编程语言的 IBM 实现（包括实现所定义的用于移植最初用 GNU C 和 g++ 开发的应用程序的扩展）的信息。

compiler.pdf

XL C/C++ Compiler Reference 包含关于各种编译器选项、编译指示、宏以及内置函数（包括那些用于并行处理的函数）的信息。

stdlib.pdf

Standard C++ Library Reference 包含关于 XL C/C++ 附带提供的“标准 C++ 库”（包括“标准模板库”）的详细信息。

proguide.pdf

XL C/C++ Programming Guide 包含关于使用 XL C/C++ 来进行编程的信息，在其它出版物中没有涉及这些信息。

legacy.pdf

C/C++ Legacy Class Libraries Reference 包含关于“USL 复杂数学类库”和“USL I/O 流库”（与 XL C/C++ Enterprise Edition 一起提供以与前发行版的编译器兼容）的信息。

debug.pdf

IBM Distributed Debugger 包含 Distributed Debugger 工具的文档。

访问其它信息

要获取关于 XL C/C++ 的最新信息，请访问以下 URL 处的产品文档和支持页面。另外，IBM 技术支持组织开发的 IBM 红皮书包含基于来自实际经验的真实情况的技术信息。

- 网址为 <http://www.ibm.com/software/awdtools/vacpp/library> 的“信息中心”。
- 网址为 <http://www.ibm.com/software/awdtools/ccompilers> 的产品支持站点。

- 网址为 <http://www.redbooks.ibm.com> 的 IBM 红皮书。

您可能会发现下列红皮书对于 XL C/C++ 应用程序开发很有帮助:

- *AIX 5L Porting Guide*, SG24-6034-00。
- *Developing and Porting C and C++ Applications on AIX*, SG24-5674-01。
- *POWER4 Processor Introduction and Tuning Guide*, SG24-7041-00。
- *Scientific Applications in RS/6000 SP Environments*, SG24-5611-00。
- *Understanding IBM eServer pSeries Performance and Sizing*, SG24-4810-01。

版本 7 中的新增内容

XL C/C++ Enterprise Edition for AIX 中的新功能和增强分为三个类别：性能和优化、符合业界标准以及易于使用。

性能和优化

许多新的功能部件和增强功能都可归入优化和性能调整类别。

机器体系结构和硬件

对选项 **-qarch** 和 **-qtune** 的改进

编译器选项 **-qarch** 控制为指定的机器体系结构生成的特定指令。选项 **-qtune** 调整指令、调度和其它优化方法以在指定的硬件上增强性能。这些选项共同起作用以生成对于指定的体系结构可获得最佳性能的应用程序代码。熟练使用这些选项的组合是充分利用 IBM 处理器和硬件的关键。在本发行版中已增强这些选项的协作以增加对 POWER5 和 PowerPC 970 硬件平台的支持和使这些选项更易于使用。

对于 **-qarch** 指定的特定体系结构，用缺省 **-qtune** 子选项进行编译将生成对于该体系结构可获得最佳性能的代码。选项 **-qarch** 现在可指定一组体系结构；用 **-qtune=auto** 进行编译将生成可以在指定组中的所有体系结构上运行的代码，但指令序列将是在编译机器的体系结构上具有最佳性能的指令序列。

用于 POWER5 处理器的新内置函数

下列内置函数在所有 PowerPC 系统上都可用。在 POWER5 系统上，这些函数使用 POWER5 指令来利用 POWER5 硬件。在 *XL C/C++ Compiler Reference* 中描述了所有受支持的内置函数。

用于所有 PowerPC 系统的新内置函数

函数	描述
int __popcnt4(unsigned int);	返回对 32 位整数设置的位数 (=1)。
int __popcnt8 (unsigned long long);	返回对 64 位整数设置的位数 (=1)。
int __poppar4(unsigned int);	如果对 32 位整数设置了奇数位，则返回 1。否则返回 0。
int __poppar8 (unsigned long long);	如果对 64 位整数设置了奇数位，则返回 1。否则返回 0。
unsigned long __mfspr(const int);	返回指定的专用寄存器中的值。
void __mtspr(const int, unsigned long);	设置由 const int 指定的专用寄存器。
unsigned long __mfmsr();	返回机器状态寄存器。
void __mtmsr(unsigned long);	设置机器状态寄存器。

下列内置函数只在 POWER5 处理器上可用。

函数	描述
double __fre(double);	返回浮点倒数运算的结果。结果为 1/x 的双精度估算值。

函数	描述
<code>float __frsqrtes(float);</code>	返回倒数平方根运算的结果。结果为 x 的平方根倒数的单精度估算值。
<code>unsigned long __popcntb (unsigned long);</code>	对源代码操作数中的每个字节计 1 位并将该计数放入相应的字节的结果中。
<code>void __protected_unlimited_stream_set_go(unsigned int direction, const void* addr, unsigned int ID);</code>	建立使用标识 ID 的无限长度的受保护流。流标识应该在范围 0 至 15 之间。该流从 <code>addr</code> 处的高速缓冲行开始。该流按照 <code>direction</code> 的指定从递增内存地址或递减内存地址进行访存。对于递增内存地址（即正向）， <code>direction</code> 的值为 1；对于递减内存地址， <code>direction</code> 的值为 3。该流受到保护而不会被任何硬件检测到的流所替换。（在 PowerPC 970 和 POWER5 上可用。）
<code>void __protected_stream_set(unsigned int direction, const void* addr, unsigned int ID);</code>	建立使用标识 ID 的有限长度的受保护流。该流从 <code>addr</code> 处的高速缓冲行开始，然后按照 <code>direction</code> 的指定从递增内存地址或递减内存地址进行访存。该流受到保护而不会被任何硬件检测到的流所替换。
<code>void __protected_stream_count(unsigned int unit_cnt, unsigned int ID);</code>	设置 ID 所标识的有限长度受保护流的高速缓冲行数目。高速缓冲行的数目由参数 <code>unit_cnt</code> 指定并且应该在范围 0 至 1023 之间。
<code>void __protected_stream_go();</code>	开始预取所有有限长度受保护流。
<code>void __protected_stream_stop(unsigned int ID);</code>	停止预取 ID 所标识的受保护流。
<code>void __protected_stream_stop_all();</code>	停止预取所有受保护流。

用于浮点除法的新内置函数

在此发行版中附带了四个用于浮点除法的新内置函数。浮点除法算法的这些软件实现利用 PowerPC 体系结构，它们比在向量上下文中使用对应硬件指令明显要快。所有 PowerPC 处理器（包括 POWER5）都支持新的内置函数。

如果在源程序中编写了浮点除法，则在缺省情况下获取硬件除法指令，但是编译器可根据它认为哪种快一些来选择硬件或软件除法代码。新内置函数允许用户显式调用软件算法。当调用这些例程时，缺省舍入方式（舍入至最近似的数）必须生效。

用于浮点除法的内置函数

函数	描述
<code>double __swdiv_nochk(double, double);</code>	double 类型的浮点除法；不进行范围检查。自变量限制：不允许分子为无穷大，也不允许分母为无穷大、零或反向规格化数。
<code>double __swdiv(double, double);</code>	double 类型的浮点除法。没有自变量限制。
<code>float __swdivs_nochk(float, float);</code>	float 类型的浮点除法；不进行范围检查。自变量限制：不允许分子为无穷大，也不允许分母为无穷大、零或反向规格化数。
<code>float __swdivs(float, float);</code>	double 类型的浮点除法。没有自变量限制。

新增的 XL C/C++ 编译指示 (pragma)

XL C/C++ Compiler Reference 中详细描述了编译指示伪指令。

编译指示	描述
<code>#pragma unrollandfuse</code>	用于优化嵌套 for 循环的编译指示。指示编译器复制外部循环（它本身也是一个循环嵌套）的主体，并将副本融合成单个展开的循环嵌套。
<code>#pragma stream_unroll</code>	将包含在 for 循环中的一个流分成多个流。用于具有大量迭代计算和少量流的循环。
<code>#pragma block_loop</code>	指示编译器为循环嵌套中的特定 for 循环创建分块循环。将循环分块涉及将循环的迭代空间分成多个部分或块。将创建额外的外部循环，称为分块循环，它驱动每个块的原始循环。
<code>#pragma loopid</code>	用作用域唯一的标识来标记 for 循环。 <code>#pragma block_loop</code> 和其它 <code>pragma</code> 可使用该标识来控制此循环的转换，并通过使用 <code>-qreport</code> 选项提供有关循环转换的信息。该标识还可以用来标识分块循环。
<code>#pragma disjoint</code>	增加的 C++ 实现。
对 <code>#pragma unroll</code> 的扩展	循环展开是指复制循环主体以便减少完成循环所需的迭代次数。 <code>#pragma unroll</code> 伪指令指示编译器可展开紧跟在该伪指令后面的 for 循环。此编译指示的功能已经得到扩展，以允许将它应用于最里层和最外层的 for 循环。扩展的 <code>#pragma</code> 功能仍不能应用于具有备用入口点的 for 循环。

新的优化实用程序

此发行版包含与基于概要的反馈 (PDF) 编译过程有关的两个新实用程序。通过使用基于概要的反馈，编译器可以提供优化的反映可执行文件如何在许多不同的方案中运行的可执行文件。作为运行该检测的可执行文件的副作用，将在其中一个方案中产生一个 PDF 记录。这些记录组成了通过对比来定义典型程序行为的数据。

showpdf 命令提供了显示在基于概要的反馈练习运行中执行的所有过程的调用和块计数的能力。该实用程序要求用选项 `-qpdf1` 和 `-qshowpdf` 编译。

mergepdf 命令允许用户指定两个或多个 PDF 记录的相对重要性并将它们合并为一个记录。这允许用户用更高的执行计数（即，更长的运行时间）来补偿练习运行，否则练习运行就会支配概要数据。

IBM Mathematics Accelerated Subsystem (MASS) 库

从版本 7.0 开始，XL C/C++ 提供了已调整的数学内部函数的 IBM Mathematical Accelerated Subsystem (MASS) 库。MASS 标量库 `libmass.a` 包含 AIX 系统库 `libm.a` 中一组常用的加速数学内部函数。MASS 向量库 `libmassv.a`、`libmassvp3.a` 和 `libmassvp4.a` 包含可与 Fortran 或 C 应用程序配合使用的已调整且线程安全的内部函数。通用向量库 `libmassv.a` 包含将在 IBM pSeries 和 RS/6000 系列中所有计算机上运行的向量函数，而 `libmassvp3.a` 和 `libmassvp4.a` 各自包含分别已对 POWER3 和 POWER4 处理器专门调整的 `libmassv.a` 函数的子集。

SMP 线程绑定

共享内存并行化（SMP）是通过创建调度为由操作系统在内核线程上运行的用户线程实现的。对于某些工作负载，将线程绑定至处理器避免了线程迁移的成本，从而能提高性能。当前，SMP 运行时创建的线程未绑定至任何特定处理器，AIX 操作系统关注线程的调度。借助于 SMP 线程绑定功能，动态链接至 SMP 运行时的程序可将它们的线程绑定至用户指定的处理器。

SMP 线程绑定采用具有两个部分的选项，该选项在 XLSMPOPT 环境变量上设置。用户指定要绑定的第一个线程的 CPU 标识和要从当前处理器跳过（跨过）的处理器数目以绑定下一个线程。

当多个 OpenMP 在同一台机器上运行时，能够指定起始 CPU 标识会非常有益。如果每个 OMP 程序都从 CPU 0 开始绑定它的线程，就会出现不平衡，在这种情况下，会装入系统中的前几个 CPU，而根本不会使用余下的 CPU。允许用户指定步幅 2 就会允许非 HPC 系统将它的线程绑定至从 CPU 0 开始的偶数编号的 CPU 标识。绑定至偶数编号的 CPU 可向线程提供完全的二级高速缓存和带宽。

使用处理器绑定的程序应该了解“动态逻辑分区”（DLPAR）。有关 DLPAR 知识的更多信息，请参阅 *General Programming Concepts: Writing and Debugging Programs*，它是 AIX 系统文档的一部分。

遵循业界标准

本节描述了 XL C/C++ 实现来符合各种业界标准的新功能部件。

ISO/IEC 14882:2003(E) 编程语言 - C++

XL C/C++ 符合修订后的国际 C++ 标准 ISO/IEC 14882:2003(E) 编程语言 - C++。

从版本 7 开始，XL C++ 添加了对无序关联容器的支持，与 *Draft Technical Report on Standard Library Extensions*（TR1）相符。作为对“C++ 标准库”的扩展添加的散列函数和基于散列的新容器如下所示：

头文件	添加内容
标准头 <functional>	函数模板 <code>std::tr1::hash</code>
新的头 <unordered_set>	容器 <code>std::tr1::unordered_set</code> 容器 <code>std::tr1::unordered_multiset</code>
新的头 <unordered_map>	容器 <code>std::tr1::unordered_map</code> 容器 <code>std::tr1::unordered_multimap</code>

TR1 库是在嵌套名称空间 `std::tr1` 中声明的。新的 XL C++ TR1 库组件通过定义宏 `__IBMCPP_TR1__` 启用。

C、C++ 和 Fortran 的 OpenMP API V2.0 支持

OpenMP 应用程序接口（API）是可移植且可伸缩的编程模型，它为用 C、C++ 和 Fortran 开发多平台的共享内存并行应用程序提供标准接口。该规范由 OpenMP 组织定义，该组织由主要的计算机硬件和软件供应商组成，其中包括 IBM。XL C/C++ Enterprise Edition for AIX 符合“OpenMP 规范 2.0”，该编译器能识别并保持下列 OpenMP V2.0 元素的语义：

- #pragma omp 伪指令中多个子句的逗号定界符。
- num_threads 子句。
- copyprivate 子句。
- threadprivate 静态块作用域变量。
- 支持 C99 可变长度数组。
- 私有变量的冗余声明。
- 计时例程 omp_get_wtime 和 omp_get_wtick。

增强的 Unicode 和 NLS 支持

按照“C 标准”委员会的最新报告中的建议，C 编译器扩展了 C99 以添加新的数据类型来支持 UTF-16 和 UTF-32 文字。C++ 编译器也支持这些新的数据类型以与 C 兼容。




本发行版中的另外一个新功能是：如果应用程序在 AIX V5.2 操作系统上运行，则 C++ 运行时能够使用 AIX V5.2 操作系统的能力来装入多个语言环境。

对 Boost 库的支持

XL C++ 编译器提供了与 1.30.2 Boost 库的高度兼容性。创建这些库是为了提供一组可再用的开放式源代码 C++ 库，这些库适合标准化。有关更多信息，请参阅 Boost Web 站点，网址为 <http://www.boost.org>。

与 GNU C 和 C++ 相关的语言扩展

C99 的 GNU C 扩展和标准 C++ 的 GNU 扩展不是业界标准。但是，这些来自开放式源代码（Open Source）团体的非专有语言功能，在一定程度上已获得普遍使用。XL C/C++ 实现 GNU C 和 C++ 扩展的一部分。本发行版中添加了对下列 GNU C 功能的支持。

功能	注释
标号作为值	包括已计算的 goto 语句。此功能现在与 GNU C 实现完全兼容。
类型属性	属性 aligned 和属性 packed。 <div> 仅 C 支持。</div>
函数属性	属性 format、format_arg、always_inline 和 noinline。
备用关键字	对 __extension__ 的实现的内部更改。
嵌套函数	<div> 仅 C 支持。</div>
强制类型转型为联合类型	<div> 仅 C 支持。</div>
带有可变数目的自变量的宏	当未指定 __VA_ARGS__ 自变量时，使用一个代替 __VA_ARGS__ 的标识，并除去尾部逗号。
带有 C 表达式操作数的 gcc 内联汇编程序指令	仅部分支持。
GNU C 复杂类型	添加了 C++ 支持。
GNU C 十六进制浮点常数	添加了 C++ 支持。
C99 组合文字	添加了 C++ 支持。
长度为零的数组	添加了 C++ 支持。

功能	注释
可变长度数组	添加了 C++ 支持。

与第三方 C++ 运行时库的协调

C++ 编译器可以编译 C++ 应用程序以使应用程序只支持核心语言，从而使得应用程序能够与第三方供应商的 C++ 运行时库链接。下列归档文件启用了此功能。

lib*C*core.a	包含异常处理、RTTI、静态初始化、new 和 delete 操作符。不包含下列任何一个库：Input/Output、Localization、STL Containers、Iterators、Algorithms、Numerics 和 Strings。
libCcore.a	C++ 运行时库 libC.a 的核心语言版本。
libC128core.a	libC128.a 的核心语言版本。
libhCcore.a	libhC.a 的语言核心版本。

添加了下列调用命令以方便使用这些归档：

x1Ccore x1C128core	x1Ccore_r x1C128core_r	x1Ccore_r7 x1C128core_r7
-----------------------	---------------------------	-----------------------------

易于使用

新增的 C++ 编译器调用

为了实现在所有受支持平台之间的可移植性，添加了编译器调用 **xlc++**。该调用等价于所有平台上的 **xlc** 调用，建议采用。然而，**xlc** 仍完全受支持。

文档

XL C/C++ 附带有“信息中心”，该“信息中心”由可搜索的 HTML 文件组成。同以前的发行版相比，新的帮助系统每次的搜索引擎产生的搜索结果的关联性更强。“信息中心”可安装在内部网上并通过使浏览器指向 http://server_name:5312/help/index.jsp 来访问。也可可在 <http://www.ibm.com/software/awdtools/vacpp/library> 中联机查看产品帮助系统。

从版本 7 开始，为编译器调用命令和每个命令行实用程序提供了手册页。编译器调用的手册页替换了在先前发行版中提供的文本帮助文件。

模板注册表增强

C++ 编译器使用批处理模板实例化模式，该模式涉及模板实例化注册表。在本发行版中，编译器将版本化信息添加至所创建的模板注册表文件。编译器内部使用此信息来跟踪应该使用哪个版本的模板注册表文件格式。编译器选项 **-qtemplateresistry** 已被更改为在缺省情况下启用。

新增的 XL C/C++ 选项

XL C/C++ Compiler Reference 中详细描述了新增的和已更改的编译器选项。

选项	描述和注释
-qasm=gcc	允许部分支持带 C 表达式操作数的汇编程序指令。指示编译器识别 asm 关键字及其备用拼写并将 gcc 语法和语义用于该关键字。缺省选项是 -qnoasm 。
-qasm_as	指定用于调用备用汇编程序的路径和标志以便处理 asm 伪指令中的代码。此选项覆盖在编译器配置文件中定义的 as 命令的缺省设置。
-qdirectstorage	声明可在给定的编译单元中引用启用直写或禁止高速缓存的存储器。此选项的目的是为了避免由于 PowerPC 体系结构允许的不同存储器控制属性而引起的意外行为。缺省选项是 -qnodirectstorage 。
-qkeepparm	确保在寄存器中传递的函数参数将保存至堆栈，而不是可能移至不同内存位置来提高性能。缺省选项是 -qnokeepparm 。
-qnoprefetch	指示编译器不自动插入软件预取指令，从而允许用户关闭这方面的优化。缺省选项是 -qprefetch 。
-qnotrigraph	指示编译器不解释三字母词序列，而无论指定的语言级别如何。在 AIX 上，缺省选项是 -qtrigraph 。
-qroptr	指示编译器允许将只读指针从 .data 部分移至 .text 部分。 .text 部分始终是只读，永远不会被装入程序或应用程序修改。如果允许将应用程序的常量指针从 .data 部分移至 .text 部分（该部分在多个进程之间是共享的），则有可能减少内存用量。在 -qroptr 生效时产生的代码不能用于共享库。缺省选项是 -qnoroptr ，它让只读指针留在 .data 部分中。
-qsaveopt	指示编译器保存将源文件编译成相应的目标文件所使用的命令行选项。如果编译未产生 .o 文件，则该选项无效。缺省选项是 -qnosaveopt 。
-qshowpdf	当与 -qpdf1 一起指定时，编译器将额外的概要分析信息插入已编译的应用程序中以收集应用程序中所有过程的调用和块计数。运行已编译的应用程序会将调用和块计数记录到文件 ._pdf 中。然后可使用 showpdf 实用程序来检索 ._pdf 的内容。缺省选项是 -qnoshowpdf 。
-qsourcetype	控制输入文件名的解释。缺省行为是源文件的编程语言暗含在其文件名的后缀中。缺省选项是 -qsourcetype 。
-qweaksymbol	指示编译器为具有外部链接的内联函数、用 #pragma weak 指定为弱标识的标识或用 __attribute__((weak)) 指定为弱函数的函数生成弱符号。当编译包含外部内联函数的 C++ 程序时，该选项还可用于停止发出存在重复符号的链接程序的警告消息。缺省选项是 -qnweaksymbol 。
-qutf	启用对 UTF 文字语法的识别，该语法提供针对 Unicode 编码格式的基于 16 位和 32 位的字符串文字。
-qfltrap=nanq	指示编译器在代码中生成额外的指令以捕获 NaNQ (Not a Number Quiet)。目的是检测由浮点指令处理或生成的所有 NaNQ，包括有效操作所创建的那些 NaNQ。
-qipa=infrequentlabel	指定在一般程序运行期间可能很少被调用的一组标号。通过减少对这些标号调用的优化，编译器可使程序的其它部分运行更快。此选项仅适用于用户定义的标号。

选项	描述和注释
-qlanglvl=newexcp	-qlanglvl=newexcp 子选项确定未能分配存储器的分配函数抛出的异常的数据类型。当该子选项生效时，如果用非空 <code>throw</code> 表达式声明的分配函数未能分配存储器，则该函数抛出类 <code>std::bad_alloc</code> 或派生自 <code>std::bad_alloc</code> 的类的异常。此行为符合 C++ 标准。当未显式指定此子选项时，未能分配存储器的任何分配函数返回一个空指针，而无论它是否指定了异常。
-qshowinc 子选项	与 <code>-qsource</code> 一起使用，以便在程序源清单中有选择地显示用户头文件或系统头文件。缺省选项是 -qnoshowinc 。新的子选项允许对包括哪些头文件进行更明确的控制。个别子选项是 <code>all</code> 、 <code>usr</code> （包括用户头文件）、 <code>nouser</code> （不包括用户头文件）、 <code>sys</code> （包括系统头文件）和 <code>nosys</code> （不包括系统头文件）。可使用冒号定界符指定多个子选项。

定制编译环境

本节讨论 XL C/C++ 用来指定包含 `include` 文件和库的目录搜索路径的机制。这些机制是环境变量、符号链接和配置文件。

环境变量

编译环境的一部分是特殊文件（如库和包含文件）的搜索路径。编译器使用以下系统变量。

LIBPATH 指定动态装入库的目录路径。参照其它系统上的 `LD_LIBRARY_PATH` 变量。影响链接编辑器和系统装入程序。

如果在环境中设置了 `LIBPATH`，则该值由 `ld` 命令读取和使用。如果发生了链接操作（不管是直接发生的还是作为编译器调用的一部分发生的），则使用 `LIBPATH` 的值来搜索从属库，而变量的内容存储在生成的模块中。如果没有 `-L` 命令行选项，则 `LIBPATH` 的内容附加到缺省库搜索路径之前。

如果命令行的任何地方使用了 `-L` 选项，则会在链接时忽略 `LIBPATH`。按路径（在命令行上通过一个或多个 `-L` 选项指定）的出现顺序并置这些路径。组合路径说明附加到缺省库搜索之前并存储在结构化模块的装入程序部分中。

在运行时，`LIBPATH` 由系统装入程序根据是否涉及动态装入以不同的方法使用。

会在执行时检查 `LIBPATH` 环境变量并在程序的特权不同于调用程序的特权时清除该环境变量。用来与备用有效用户或组标识一起运行的过程应只从嵌入在应用程序中的可信位置查找从属模块。

MANPATH 指定产品手册页的目录路径。

NLSPATH 指定“本地语言支持”库的目录路径。

PATH 指定编译器的可执行文件的目录路径。

PDFDIR 指定创建概要数据文件的目录。未设置缺省值，编译器将概要数据文件放置在当前工作目录中。对于基于概要的反馈，建议将此变量设置为绝对路径。

TMPDIR 指定创建临时文件的目录。在进行高级别的优化时，临时文件可能需要大量的磁盘空间，缺省位置可能不适合。

为路径创建符号链接

不会自动在 `/usr/bin` 中安装 XL C/C++ 的命令行接口。如果想调用编译器时不需要指定完整路径，可以执行下列其中一个步骤：

- 为包含在 `/usr/vacpp/bin` 和 `/usr/vac/bin` 中的特定驱动程序创建至 `/usr/bin` 的符号链接。
- 将 `/usr/vac/bin` 和 `/usr/vacpp/bin` 添加至 `PATH` 环境变量。

配置文件

配置文件是纯文本文件，它指定每次运行编译器时将读取的选项。配置文件的名称以 `.cfg` 文件扩展名结束。

通过用 **-F** 选项调用编译器并指定全限定文件名，可以指示编译器使用特定配置文件。

编译器选项 **-I** *directory_name* 允许您将目录添加至配置文件中的搜索路径。配置文件自己在内部使用 **-I** 选项来设置它控制的目录路径。编译器在搜索命令行上的 **-I** 选项指定的目录之前搜索配置文件中的 **-I** 选项指定的目录。

请参阅 *XL C/C++ Compiler Reference* 以获取更多信息。

控制编译过程

整个编译过程由三个阶段组成：预处理、转换为目标代码和链接。缺省情况下，编译器调用命令调用编译过程的所有阶段以将程序从源代码转换为可执行的输出。如果在调用编译器时指定了输入和输出文件的文件名，则它根据输入和输出文件的文件名后缀（扩展名）确定起始阶段和结束阶段。

也可以在任何编译阶段通过使用适当的编译器选项来创建特定类型的输出文件。例如，用 **-E** 或 **-P** 选项调用 **xlc** 或 **xlC** 命令仅对输入文件执行预处理阶段。编译器调用根据输入文件名的扩展名确定是调用编译器、汇编程序还是链接程序。

相关参考

- *XL C/C++ Compiler Reference* 中的 **-qphsinfo** 编译器选项

调用编译器

XL C/C++ 提供了一些基本编译器调用命令以供选择，这些命令支持 C 和 C++ 语言的各种版本级别。每个调用命令自动为语言级别、其它相关语言功能的选项以及任何相关的预定义宏设置编译器子选项。在大多数情况下，应使用 **xlc** 命令来编译 C 源文件，使用 **xlC** 命令来编译 C++ 源文件（或同时有 C 和 C++ 源文件时）。

基本调用命令

xlc	cc	xlc++core
xlc++	c89	xlCcore
xlC	c99	

调用 **xlc++** 等价于调用 **xlC**。

提供了基本命令的变体以支持特殊环境和文件系统的要求。变体是通过将后缀附加至基本命令形成的。

后缀	描述
_r _r4 _r7	用于编译线程安全的应用程序。也称为可重入的编译器调用。
128 128_r 128_r4 128_r7	将 long double 类型的长度从 64 位增加到 128 位并与 C 和 C++ 运行时的 128 位版本链接。当附加至基本调用 c89 或 c99 时，将在后缀前加下划线（如 c99_128 ）。

另外，**gxlc** 和 **gxlc++** 实用程序是专门的编译器调用形式。

对象模型

对象模型描述 C++ 对象和帮助程序数据结构在存储器中的布局。它可能还会影响名称混成。在 AIX 平台上支持两种对象模型：compat 和 ibm。在编译器布置虚函数表的方式、为虚拟基本类提供的支持类型以及使用的名称混成模式这些方面，它们有所不同。compat 对象模型创建运行时模块，该模块将与用相同模型或用编译器的前版本编译的其它模块兼容。ibm 对象模型可提供改进的性能，尤其是当源代码包含具有许多虚拟基本类的类层次结构时。此对象模型趋向于创建较小的派生类和更快地访问虚函数表。

输入和输出文件的类型

编译器使用文件扩展名来确定适当的编译阶段并调用关联的工具。

编译器接受以下类型的文件作为输入：

接受的输入文件类型

文件类型描述	文件扩展名	示例
C 和 C++ 源文件	.c (小写 c) 表示 C 语言源文件； .C (大写 c)、.cc、.cp、.cpp、.cxx 和 .c++ 表示 C++ 源文件。	file_name.c file_name.C、file_name.cc、file_name.cpp、file_name.cxx 和 file_name.c++
已预处理的源文件	.i	file_name.i
目标文件	.o	hello.o
汇编程序文件	.s	check.s
归档文件	.a	v1r5.a
可装入的模块或共享库文件	.so	my_shrlib.so
IPA 控制文件 (-qipa=file_name)	未强制使用 file_name 的命名约定。	ipa.ctl

当调用编译器时可以指定以下类型的输出文件：

输出文件的类型

文件类型描述	示例
可执行文件	缺省情况下为 a.out
目标文件	file_name.o
可装入的模块或共享目标文件	file_name.so
汇编程序文件	file_name.s
已预处理的文件	file_name.i
清单文件	file_name.lst
目标文件	file_name.u

相关参考

- XL C/C++ Compiler Reference 中的 -qsourcetype 编译器选项

缺省行为

如果调用编译器时没有指定任何选项，则编译器的行为由下列缺省设置控制：

- 尝试读取并调用在配置文件中指定的选项。
- 使用缺省对齐 `-qalign=power` 来对齐结构。
- 在当前目录中生成名为 `a.out` 的未优化的可执行文件。

请参阅 *XL C/C++ Compiler Reference* 以获取更多信息。

编译器选项入门

编译器选项执行各种功能，如设置编译器特征、描述要生成的目标代码、控制发出的诊断消息和执行某些预处理器功能。可以在命令行、配置文件、源代码或这些技术的任何组合中指定编译器选项。未显式设置的大多数选项采用缺省设置。

当指定了多个编译器选项时，有可能发生选项冲突和不兼容性。为了以一致的方式解决这些冲突，除非另有指定，否则编译器应用以下优先级顺序：

1. 源文件覆盖
2. 命令行覆盖
3. 配置文件覆盖
4. 缺省设置

在多个命令行选项中，一般是最后指定的选项有效。

注：**-I** 编译器选项是一种特殊情况。编译器在搜索在命令行上用 **-I** 指定的目录之前，它首先搜索在 `vac.cfg` 文件中用 **-I** 指定的任何目录。该选项是累积的而不是优先的。


相关参考

- 请参阅 *XL C/C++ Compiler Reference* 以获取更多信息。

编译器消息

XL C/C++ 对诊断消息使用五个级别的分类模式。每个严重性级别与一个编译器响应相关联。并非每个错误都会停止编译。下表对严重性级别的缩写提供了相应的键以及相关关联的编译器响应。

严重性级别和编译器响应

字母	严重性	编译器响应
I	信息	编译继续进行。消息报告编译期间发现的情况。
W	警告	编译继续进行。消息报告有效但可能是意外的情况。
 C E	错误	编译继续进行并生成目标代码。存在编译器可以更正但程序可能不会产生期望结果的错误情况。
S	严重错误	编译继续进行，但未生成目标代码。存在编译器不能更正的错误情况。
U	不可恢复的错误	编译器停止。遇到不可恢复的错误。如果消息指示资源限制（例如，文件系统已满或调页空间已满），则提供其它资源并重新编译。如果消息指示需要不同的编译器选项，使用它们重新编译。如果消息指示内部编译器错误，应向 IBM 服务代表报告该消息。

编译器的缺省行为是使用选项 **-qnoinfo** 或 **-qinfo=noall** 进行编译。**-qinfo** 的子选项提供指定信息诊断的特定类别的能力。例如，**-qinfo=por** 将输出限制为与可移植性问题相关的那些消息。

注：在 C 中，未指定子选项的选项 **-qinfo** 等价于 **-qinfo=all**；在 C++ 中，未指定子选项的 **-qinfo** 等价于 **-qinfo=all:noppt**。

返回码

在编译结束时，在以下任何一种情况下，编译器将返回码设置为零：

- 未发出任何消息。
- 诊断到的所有错误的最高严重性级别小于 **-qhalt** 编译器选项的设置，并且错误数未达到 **-qmaxerr** 编译器选项所设置的限制。
- 未发出 **-qhaltonmsg** 编译器选项所指定的消息。

否则，编译器将设置 *XL C/C++ Compiler Reference* 中介绍的返回码之一。

编译器消息格式

缺省情况下，诊断消息具有下列格式：

`"file", line line_number.column_number: 15cc-nnn (severity) message_text`。

其中 15 是编译器产品标识，*cc* 是指示发出消息的编译器组件的一个两位数的代码，*nnn* 是消息号，*severity* 是严重性级别的字母。*cc* 的可能值有：

- 00 代码生成或优化消息
- 01 编译器服务消息
- 05 特定于 C 编译器的消息
- 06 特定于 C 编译器的消息
- 40 特定于 C++ 编译器的消息
- 86 特定于过程间分析（IPA）的消息

此格式与启用 **-qnosrcmsg** 选项进行编译时的格式相同。要获取另一种消息格式，使源代码行与诊断消息一起显示，则尝试用 **-qsrcmsg** 选项进行编译。启用此选项会指导编译器将它认为包含错误的源代码行、该源代码行下面指向该源代码行的特定点的某行（若有可能）和诊断消息打印至标准错误。

注：不会将消息用作其它程序的输入。消息格式和内容不打算成为编程接口，因此可能在发行版之间更改。

通过 **gxlc** 和 **gxlc++** 重用 GNU C 和 C++ 编译器选项

每个 **gxlc** 和 **gxlc++** 实用程序都接受 GNU C 或 C++ 编译器选项并将它们转换成相当的 XL C/C++ 选项。两个实用程序都使用 XL C/C++ 选项来创建 **xlc** 或 **xlc** 调用命令，然后实用程序使用这些命令来调用 XL C/C++。提供这些实用程序的目的是为了便于重用为先前用 GNU C 和 C++ 开发的应用程序创建的 **make** 文件。然而，为了充分利用 XL C/C++ 的能力，建议您使用 XL C/C++ 调用命令及其相关选项。

gxlc 和 **gxlc++** 的操作由配置文件 **gxlc.cfg** 控制。此文件中显示具有 XL C 或 XL C++ 对应选项的 GNU C 和 C++ 选项。并非每个 GNU 选项都具有对应的 XL C/C++ 选项。**gxlc** 和 **gxlc++** 会对未转换的输入选项返回一个警告。

gxlc 和 **gxlc++** 选项映射是可修改的。有关对 **gxlc** 和 **gxlc++** 配置文件进行添加或编辑的信息，请参阅第 27 页的『配置选项映射』。

示例

要使用 `gcc -ansi` 选项来编译 Hello World 程序的 C 版本，您可以使用：
`gxlc -ansi hello.c`

它转换为：
`xlc -F:c89 hello.c`

然后使用此命令来调用 XL C 编译器。

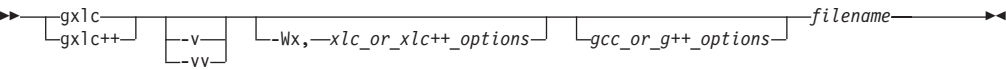
gxlc 和 gxlc++ 返回码

与其它调用命令一样，`gxlc` 和 `gxlc++` 返回输出，如清单、与编译有关的诊断消息、与 GUN 选项的不成功转换有关的警告和返回码。如果 `gxlc` 或 `gxlc++` 不能成功调用编译器，则它将返回码设置为以下其中一个值：

- 40** 检测到 `gcc` 或 `g++` 选项错误或不可恢复的错误。
- 255** 当进程运行时检测到错误。

gxlc 和 gxlc++ 语法

下图显示 `gxlc` 和 `gxlc++` 语法：



其中：

filename

是要编译的文件名称。

- v** 允许您验证将来用来调用 XL C/C++ 的命令。`gxlc` 或 `gxlc++` 在使用它所创建的 XL C/C++ 调用命令调用编译器之前显示该命令。
- vv** 允许您运行模拟。`gxlc` 或 `gxlc++` 显示它所创建的 XL C/C++ 调用命令，但不调用编译器。

-Wx, xlc_or_xlc++_options

将给定的 XL C/C++ 选项直接发送至 `xlc` 或 `xlc++` 调用命令。`gxlc` 或 `gxlc++` 将给定的选项添加至它正在创建的 XL C/C++ 调用，而不会尝试转换它们。将此选项与已知的 XL C/C++ 选项配合使用以提高该实用程序的性能。多个 *xlc_or_xlc++_options* 使用逗号定界符。

gcc_or_g++_options

是将转换为 `xlc` 或 `xlc++` 选项的 `gcc` 或 `g++` 选项。实用程序为它不能转换的任何选项发出一条警告。`gxlc` 和 `gxlc++` 当前可识别的 `gcc` 和 `g++` 选项列示在配置文件 `gxlc.cfg` 中。多个 *gcc_or_g++_options* 通过空格字符定界。








GNU C 和 C++ 至 XL C/C++ 选项映射

下表列示 `gxlc` 和 `gxlc++` 接受并转换的 GNU C 和 C++ 选项。将忽略指定为这些实用程序之一的输入的所有其它 GNU 选项或生成错误。如果 GNU 选项的否定格式存在，则 `gxlc` 和 `gxlc++` 还识别和转换否定格式。

已映射的选项：GNU C 和 C++ 至 XL C/C++

GNU C 和 C++ 选项	XL C/C++ 选项
-###	-#
-ansi	-F:c89













已映射的选项: GNU C 和 C++ 至 XL C/C++

GNU C 和 C++ 选项	XL C/C++ 选项
-B	-B
-C	-C
-c	-c
-Dmacro[=defn]	-Dmacro[=defn]
-E	-E
-e	-e
-fdollars-in-identifiers	-qdollar
 -fdump-class-hierarchy	-qdump_class_hierarchy
 -fexceptions	-qeh
 -ffor-scope	-qlanglvl=ansifor
 -fno-for-scope	-qlanglvl=noansifor
-ffunction-sections	-qfuncsect
-finline	-qinline
-finline-functions	-qinline
 -finline-limit=n	-qinline=limit=n
 -fkeep-inline-functions	-qkeepinlines
 -fno-gnu-keywords	-qnokeyword=typeof
 -fno-operator-names	-qnokeyword=and -qnokeyword=bitand -qnokeyword=bitor -qnokeyword=compl -qnokeyword=not -qnokeyword=or -qnokeyword=xor
-fpascal-strings	-qmacpstr
-fPIC	-qpik=large
-fpic	-qpik=small
 -frtti	-qrtti
-fshort-enums	-qenum=small
-fsigned-bitfields	-qbitfields=signed
-fsigned-char	-qchars=signed
-fstrict-aliasing	-qalias=ansi
-fsyntax-only	-qsyntaxonly
-funroll-all-loops	-qunroll=yes
-funroll-loops	-qunroll=yes
-funsigned-bitfields	-qbitfields=unsigned
-funsigned-char	-qchars=unsigned
-fwritable-strings	-qnoro
-g	-g
-g3	-g

已映射的选项: GNU C 和 C++ 至 XL C/C++

GNU C 和 C++ 选项	XL C/C++ 选项
-ggdb	-g
-gxcoff	-g
-Idir	-Idir
-Ldir	-Ldir
-llibrary	-llibrary
-M	-M
-MD	-M
-maix32	-q32
-maix64	-q64
-mcpu=403	-qarch=403
-mcpu=601	-qarch=601
-mcpu=602	-qarch=602
-mcpu=603	-qarch=603
-mcpu=604	-qarch=604
-mcpu=common	-qarch=com
-mcpu=power	-qarch=pwr
-mcpu=power2	-qarch=pwr2
-mcpu=powerpc	-qarch=ppc
-mcpu=powerpc64	-qarch=ppc64
-mcpu=rs64a	-qarch=rs64a
-mno-fused-madd	-qfloat=nomaf
-mfused-madd	-qfloat=maf
-mlong-double-64	-qnolonglong
-mlong-double-128	-qlonglong
-mpower	-qarch=pwr
-mpower2	-qarch=pwr2
-mpowerpc	-qarch=ppc
-mpowerpc-gfxopt	-qarch=pwrgr
-mpowerpc64	-qarch=ppc64
-mtune=403	-qtune=403
-mtune=601	-qtune=601
-mtune=602	-qtune=602
-mtune=603	-qtune=603
-mtune=604	-qtune=604
-mtune=common	-qtune=com
-mtune=power	-qtune=pwr
-mtune=power2	-qtune=pwr2
-mtune=powerpc	-qtune=ppc
-mtune=powerpc64	-qtune=ppc64
-mtune=rs64a	-qtune=rs64a

已映射的选项: GNU C 和 C++ 至 XL C/C++

GNU C 和 C++ 选项	XL C/C++ 选项
-nodefaultlibs	-qnolib
-nostartfiles	-qnocrt
-nostdinc	-qnostdinc
-nostdlib	-qnolib -qnocrt
-O	-O
-O0	-qnoopt
-O1	-O
-O2	-O2
-O3	-O3
-Os	-O2 -qcompact
-o	-o
-p	-p
-pg	-pg
-r	-r
-S	-S
-s	-s
 -std=c89	-F:c89
 -std=iso9899:1990	-F:c89
 -std=iso9899:199409	-F:c89
 -std=c99	-F:c99
 -std=c9x	-F:c99
 -std=iso9899:1999	-F:c99
 -std=iso9899:199x	-F:c99
 -std=gnu89	-qlanglvl=extc89
 -std=gnu99	-qlanglvl=extc99
 -std=gnu9x	-qlanglvl=extc99
 -std=c++98	-qlanglvl=strict98
 -std=gnu++98	-qlanglvl=extended
-time	-qphsinfo
-trigraphs	-qtrigraph
-Umacro	-Umacro
-u	-u
-Wformat	-qformat
-Wuninitialized	-qinfo=ini
-Wunreachable-code	-qinfo=eff

已映射的选项: GNU C 和 C++ 至 XL C/C++

GNU C 和 C++ 选项	XL C/C++ 选项
-Wa, <i>option</i>	-Wa, <i>option</i>
-Wl, <i>option</i>	-Wl, <i>option</i>
-Wp, <i>option</i>	-Wp, <i>option</i>
-w	-w
-x assembler	-qsourcecetype=assembler
-x c	-qsourcecetype=c
-x c++	-qsourcecetype=c++
-x none	-qsourcecetype=default
-Z	-Z

所有其它 GNU 选项都会被忽略并且会发出参考消息。

配置选项映射

gxlc 和 gxlc++ 实用程序使用配置文件 gxlc.cfg 来将 GNU C 和 C++ 选项转换为 XL C/C++ 选项。gxlc.cfg 中的每个条目描述实用程序应该如何将 GNU C 或 C++ 选项映射至 XL C/C++ 选项和如何处理它。

一个条目由一个处理指令标志字符串、一个 GNU C 选项字符串和一个 XL C/C++ 选项字符串组成。必须用空格将这三个字段分隔开。如果条目仅包含前两个字段并省略 XL C/C++ 选项字符串, 则第二个字段中的 GNU C 选项将被 gxlc 识别但将被忽略而不进行提示。

字符用来将注释插入配置文件。注释可单独放在一行或放在条目末尾。

以下语法用于 gxlc.cfg 中的条目:

```
abcd "gcc_or_g++_option" "xlc_or_xlc++_option"
```

其中:

- a** 允许您通过添加 no- 作为前缀来禁用该选项。值 y 表示是, 而 n 表示否。例如, 如果将标志设置为 y, 则可象 fno-common 一样禁用 finline, 于是该条目为:

```
ynn* "-finline" "-qinline"
```

如果给出了 -fno-inline, 则 gxlc 会将它转换为 -qnoinline。

- b** 通知实用程序 XL C/C++ 选项具有一个相关联的值。值 y 表示是, 而 n 表示否。例如, -finline-limit=n 映射至 -qinline=limit=n。在这种情况下, 标志设置为 y, 于是该条目为:

```
nyn* "-finline-limit" "-qinline=limit"
```

然后, gxlc 和 gxlc++ 将期望这些选项的值。

- c** 控制选项的处理。值可以为:

- n, 它指示实用程序处理列示在 gcc-option 字段中的选项。

- **i**，它指示实用程序忽略列示在 `gcc-option` 字段中的选项。`gxl` 和 `gxl++` 将生成一条消息表示该选项已被忽略，并继续处理给出的选项。
- **e**，它指示实用程序当遇到列示在 `gcc-option` 字段中的选项时停止处理。`gxl` 和 `gxl++` 还将生成错误消息。

例如，`gcc` 选项 `-I-` 不受支持，一定会被 `gxl` 和 `gxl++` 忽略。在这种情况下，标志设置为 **i**，于是该条目为：

```
nni*      "-I-"
```

如果 `gxl` 和 `gxl++` 遇到此选项作为输入，则它将不会处理此选项并将生成警告。

d 允许 `gxl` 和 `gxl++` 基于编译器的类型包括或忽略选项。值可以为：

- **c**，它指示 `gxl` 和 `gxl++` 仅对 **C** 转换选项。
- **x**，它指示 `gxl` 和 `gxl++` 仅对 **C++** 转换选项。
- *****，它指示 `gxl` 和 `gxl++` 对 **C** 和 **C++** 转换选项。

例如，`-fwritable-strings` 受这两个编译器的支持，并映射至 `-qnor`。条目为：

```
nnn*      "-fwritable-strings"      "-qnor"
```

"gcc_or_g++_option"

是表示受 GNU C V3.3 支持的 `gcc` 或 `g++` 选项的字符串。此字段是必需的并且必须用双引号引起来。

"xlc_or_xlc++_option"

是表示 **XL C/C++** 选项的字符串。此字段是可选的，如果提供，则必须用双引号引起来。如果保留为空白，则 `gxl` 和 `gxl++` 会忽略该条目中的 *gcc_or_g++_option*。

有可能创建将映射某一范围内的选项的条目。通过使用星号（*）作为通配符来完成此操作。例如，`gcc -D` 选项需要用户定义的名称并可带一个可选值。可能具有以下选项系列：

```
-DCOUNT1=100
-DCOUNT2=200
-DCOUNT3=300
-DCOUNT4=400
```

不必为此选项的每个版本创建条目，只需一个条目：

```
nnn*      "-D*"      "-D*"
```

其中星号将替换为 `-D` 选项后面的任何字符串。

反过来，您可使用星号来排除某一范围内的选项。例如，如果您想要 `gxl` 或 `gxl++` 忽略所有 `-std` 选项，则条目将为：

```
nni*      "-std*"
```

当在选项定义中使用星号时，选项标志 *a* 和 *b* 不适用于这些条目。

字符 `%` 与 GNU C 或 GNU C++ 选项配合使用以表示选项具有相关联的参数。这用来确保 `gxl` 或 `gxl++` 将忽略与忽略的选项相关联的参数。例如，`-include` 选项不受支持并使用一个参数。应用程序一定会忽略这两者。在这种情况下，条目为：

```
nni*      "-include %"
```

相关参考

- GNU Compiler Collection 联机文档, 网址为: <http://gcc.gnu.org/onlinedocs/>

选项摘要: C 编译器

本章附录提供了 C 编译器选项的摘要, 按类型分组。较高级别的分组包含选项子组。除了一个用于源代码的基本转换的子组之外, 还有一个子组包含用于代码的特殊处理或控制 (如添加专门的调试信息) 的选项。另一个子组与链接程序和库搜索路径的控制有关。在第 33 页的『优化入门』这一章的末尾总结了与性能和优化相关的选项。有关每个选项的描述、全部选项语法和用法的信息, 请参阅 *XL C/C++ Compiler Reference*。

基本转换

此分组中的选项在源代码的基本转换方面的通用性最好。编译器选项子组一般涉及到:

- 符合标准。
- 编译器驱动程序的编译方式或控制。
- 处理源代码以进行代码生成。
- 生成专门的诊断。
- 处理已编译的代码。

与源代码的基本转换相关的选项

符合标准	编译器驱动程序的编译方式或控制
-qgenproto 和 -qnogenproto -qlanglvl -qlibansi 和 -qnolibansi	-# -q32 -q64 -F -qpath -qproto 和 -qnoproto -qsourcetype
源代码生成	
-qalloca -qasm 和 -qnoasm -qasm_as -qattr 和 -qnoattr -B -C -qcpluscmt 和 -qnocpluscmt -D -qdbcs 和 -qnodbcs -qdigraph 和 -qnodigraph -qdirectstorage 和 -qnodirectstorage -E -qfuncsect 和 -qnofuncsect -qignprag -M -ma	-qmacpstr 和 -qnomacpstr -qmakedep -qmbcs 和 -qnombcs -P -qpascal 和 -qnopascal -qroptrs 和 -qnoroptrs -qsmallstack 和 -qnosmallstack -qsyntaxonly -t -qtabsize -qtrigraph 和 -qnotrigraph -U -qutf 和 -qnoutf -W -qweaksymbol 和 -qnoweaksymbol
诊断	已编译的代码
-qflag -qinfo 和 -qnoinfo -qmaxerr 和 -qnomaxerr -qphsinfo 和 -qnophsinfo -qprint 和 -qnoprint -qshowinc 和 -qnoshowinc -qsource 和 -qnosource -qsrcmsg 和 -qnosrcmsg -qsuppress 和 -qnosuppress -V -v -w -qwarn64 和 -qnowarn64 -qxcall 和 -qnoxcall	-qbitfields -c -qchars -qdataimported -qdatalocal -qdollar 和 -qnodollar -qexpfile -o -qprocimported -qprocllocal -qprocunknown -S -qstatsym 和 -qnostatsym -qtbtable -qupconv 和 -qnoupconv

特殊处理和控制

此分组中的选项提供转换过程的精确控制，通常比基本转换选项的适用性要差一些。此编译器选项分组中的主题一般涉及到：

- 数据对齐。
- 编译器驱动程序的编译方式或控制。

- 处理源代码以进行代码生成。
- 生成专门的诊断。
- 处理已编译的代码。

用于特殊处理、优化调整和调试的选项

数据对齐	并行化
-qalign -qenum	-qsmp 和 -qnosmp -qthreaded 和 -qnothreaded
浮点和数字功能	
大小 -qldbl128 和 -qnoldbl128 -qlongdouble 和 -qnolongdouble -qlonglit 和 -qnolonglit -qlonglong 和 -qnolonglong	浮点值的四舍五入 -qrndflt 和 -qnorndflt -y
单精度值 -qhsflt 和 -qnohsflt -qhssngl 和 -qnohssngl	其它浮点选项 -qfloat -qfltttrap 和 -qnofltttrap -qmaf 和 -qnomaf
调试	
-qcheck 和 -qnocheck -qdpcl 和 -qnodpcl -qdbxextra 和 -qnodbxextra -qextchk 和 -qnoextchk -qfullpath 和 -qnofullpath -g -qhalt -qheapdebug 和 -qnoheapdebug -qinitauto 和 -qnoinitauto	-qkeepparm 和 -qnokeepparm -qlinedebug 和 -qnolinedebug -qlist 和 -qnolist -qlistopt 和 -qnolistopt -qsaveopt 和 -qnosaveopt -qsymtab -qxref 和 -qnoxref

与链接和库相关的选项

此分组中的选项与编译过程的链接阶段相关。此分组还包含提供专门的方式来指定用于查找库和头文件的搜索路径的选项。这些编译器选项一般涉及到:

- 放置字符串文字和常量。
- 静态和动态链接和库。
- 指定搜索目录。

用于控制 ld 命令的选项

放置字符串文字和常量	静态和动态链接和库
-qkeyword 和 -qnokeyword -qro 和 -qnoro -qroconst 和 -qnoroconst	-b -brtl -e -G -qmkshrobj -qstdinc 和 -qnostdinc
搜索目录	其它链接程序选项
-I -L -l (小写 L) -qidirfirst 和 -qnoidirfirst -r -Z	-f -qinlglue 和 -qnoinglue

选项摘要: C++ 编译器

大多数 C 编译器选项可用于编译 C++ 程序。下表提供 AIX 平台上特定于编译 C++ 程序的其它编译器选项和不可用于编译 C++ 程序的 C 选项:

用于 C++ 程序的编译器选项

特定于 C++ 的选项	仅用于 C 的选项
-+ -qcinc -qeh 和 -qnoeh -qhaltonmsg -qkeepinlines 和 -qnokeepinlines -qnamemangling -qobjmodel -qoldpassbyvalue 和 -qnooldpassbyvalue -qpriority -qrtti 和 -qnortti -qstaticinline 和 -qnostaticinline -qtempinc 和 -qnotempinc -qtemplatercompile 和 -qnotemplatercompile -qtemplaterregistry 和 -qnotemplaterregistry -qtempmax -qtmplparse -qtwolink 和 -qnotwolink -qunique 和 -qnounique -qvftable 和 -qnovftable	-qalloca -qassert 和 -qnoassert -qc_stdinc -qcpluscmt 和 -qnocpluscmt -qdbxextra 和 -qnodbxextra -qgenproto 和 -qnogenproto -ma -macpstr 和 -nomacpstr -qproto 和 -qnoproto -qsrcmsg 和 -qnosrcmsg -qsyntaxonly -qupconv 和 -qnoupconv

优化入门

简单编译是将源代码转换或变换为可执行目标或共享目标。优化转换可以在运行时给予应用程序更好的整体性能。XL C/C++ 提供了一些针对 PowerPC 体系结构的优化转换的组合。这些转换能够：

- 减少对关键操作执行的指令数。
- 重构生成的目标代码以最佳地使用 PowerPC 体系结构。
- 改进内存子系统的使用。
- 利用体系结构的能力来处理大量的共享内存并行化。

它们的目的是使应用程序运行更快。

如果您了解存在哪些控制因素会影响写得好的代码的转换，则较少的开发工作便可获得显著的性能改善。编程模型（如 OpenMP）使得您能编写高性能代码。本节描述了一些优化方法，编译器可执行这些优化方法来帮助您在源代码的运行时性能、手工编码的宏优化、一般可读性和整体可移植性的折衷之间进行平衡。

经常在应用程序开发周期的后面几个阶段（如产品发行版构建）尝试优化。可能的话，在尝试对代码进行优化之前，应在没有优化的情况下测试并调试代码。开始进行优化应该表示您已选择对程序最有效的算法并已正确实现它们。在很大程度上，符合语言标准与代码可以优化到的程度直接有关。优化器是最终一致性测试！

优化由编译器选项、伪指令和编译指示控制。然而，编译器友好编程习惯语法与任何一个选项或伪指令一样对于提高性能很有用。不再需要也不建议过多用手工来优化代码（例如，手工展开循环）。不常见的构造会使编译器（和其他程序员）产生困惑，并会妨碍针对新机器进行的应用程序优化。

应注意并非所有优化对所有应用程序都有益。通常必须在增加编译时间（伴随着降低调试能力）与编译器执行的优化程度之间进行折衷。

相关参考

- *XL C/C++ Programming Guide* 中的 “Using optimization levels”
- *XL C/C++ Programming Guide* 中的 “Optimizing your applications”

选择的用于优化的编译器选项

下表描述了用于优化程序性能的基本编译器选项选择的特征。有关详尽的列表，请参阅 *XL C/C++ Programming Guide*。有关可用子选项的文档，请参阅 *XL C/C++ Compiler Reference* 或选项手册页。

表 1. 用于优化的基本编译器选项

选项	描述
-qnoopt	编译器执行非常有限的优化。这是缺省选项。在开始优化应用程序之前，请用 -qnoopt 确保已成功编译了它。
-O2	编译器执行广泛的低级优化，包括图着色、公共子表达式清除、死代码清除、代数简化、常量传播、目标机器的指令调度、循环展开和软件流水线操作。

表 1. 用于优化的基本编译器选项 (续)

选项	描述
-qarch -qtune -qcache	编译器利用应用程序运行的特定硬件和指令集的特征。使用 -qarch 命令来指定应对其生成应用程序代码的处理器体系结构系列。使用 -qtune 来使优化偏向在给定微处理器上的执行。使用 -qcache 来指定特定高速缓存或内存几何结构。
-qpdf1 -qpdf2	根据通常执行代码不同部分的频率的分析，编译器使用面向概要的反馈来优化应用程序。PDF 过程对于包含没有特定结构的分支的应用程序最有用。
-O3	与 -O2 相比，编译器执行更主动的优化：循环展开程度更深并消除了对隐式内存使用的限制。
-qhot	编译器执行高阶转换，这提供了其它循环优化并有选择地执行数组填充。此选项对于执行大量数字处理的科学应用程序最有用。
-qipa	编译器执行过程间分析以将整个应用程序作为一个单元来进行优化（整个程序分析）。此选项对于包含大量经常使用的例程的业务应用程序最有用。它对于具有高级抽象的 C++ 程序也很有用。在许多情况下，此选项明显增大编译时间。
-O4	此选项等价于 -O3 -qipa -qhot -qarch=auto -qtune=auto -qcache=auto 。如果编译用时太长，则尝试使用 -O4 -qnoipa 来进行编译。
-O5	此选项等价于 -O4 -qipa=level=2 。在 AIX 平台上，只要处理器是 PowerPC 970 并且操作系统支持 AltiVec 数据类型，此选项就还会打开 -qhot=vector -qhot=simd 。

优化 pragma 入门

pragma 伪指令是特定于实现的预处理器伪指令，它使得能够从程序源代码中划分出一组行作为一个部分以告知编译器有关该部分的一些信息。与名称相似的对应编译器选项相比，pragma 伪指令通常提供更细致的控制。与优化有关的 pragma 伪指令主要用于显式通知编译器以下两点：优化的机会（编译器自己可能检测不到）或编译器可作出的假设，这使得编译器作出优化源代码的决定变得容易。

XL C/C++ 提供了用于性能优化的 pragma 伪指令，这些伪指令在 XL Fortran 伪指令中有相应的项。与 Fortran 伪指令一样，优化 pragma 可分为三种描述性类别：命令式、断言式和惯例式。这些类别的区域在于编译器必须遵从 pragma 提供的指令或信息的程度。

所有 OpenMP 伪指令都是命令式的 pragma。要求编译器严格遵从 OpenMP 标准的实现运作。但是，不保证结果程序行为产生正确的结果。XL C/C++ 实现所有 OpenMP V2.0 pragma 伪指令。

断言式 pragma 通知编译器在转换 pragma 生效的源代码行时它可安全作出的假设。断言式 pragma 用于通知，断言没有强制编译器按某种方式运作的义务。XL C/C++ 提供了与优化有关的下列断言式 pragm。

选择的断言式 #pragma 伪指令

名称	描述
<code>isolated_call(function_list)</code>	断言对已命名函数的调用没有副作用。

选择的断言式 `#pragma` 伪指令

名称	描述
<code>disjoint(variable_list)</code>	断言没有一个已命名变量具有重叠的存储器区域。
<code>ibm independent_loop</code>	断言后面的循环没有循环带有的相关性。这种自由有助于局部性和并行转换。
<code>ibm independent_calls</code>	断言后面的循环中的调用不会导致循环带有的相关性。
<code>ibm permutation</code>	断言已命名数组的元素对后面的循环的每个迭代采用单一值。此断言在短小的代码中可能会很有用。
<code>ibm iterations(iteration_count)</code>	指定后面的循环的迭代次数，这有助于编译器确定并行化循环是否有益。 <code>pragma</code> 适用于单个循环而不是整个编译单元。
<code>execution_frequency(very_low)</code>	断言将不会频繁执行包含 <code>pragma</code> 的控制路径。
<code>leaves(function_list)</code>	断言对已命名函数的调用将不会返回。

惯例式 `pragma` 建议编译器以这样的方式转换源代码：通过它的工作，能够获得更好的性能。惯例式 `pragma` 工作方式与 **`inline`** 关键字相同：不要求编译器实现建议，但认为实现建议会很有利。XL C/C++ 提供了与优化有关的下列惯例式 `pragma`。

选择的惯例式 `#pragma` 伪指令

名称	描述
<code>ibm sequential_loop</code>	指导编译器在单个线程中执行后面的循环，即使指定了 <code>-qsmp=auto</code> 也是如此。
<code>ibm snapshot(variable_name)</code>	将调试断点设置为检查指定的变量的 <code>pragma</code> 点。
<code>stream_unroll</code>	将包含在 <code>for</code> 循环中的一个流分成多个流。用于具有大量迭代计算和少量流的循环。
<code>unroll</code>	指示编译器可展开紧跟在伪指令后面的 <code>for</code> 循环。该 <code>pragma</code> 可应用于最内层和外层 <code>for</code> 循环。
<code>unrollandfuse</code>	指示编译器复制 <code>for</code> 循环的外部循环（它本身也是一个循环嵌套）的主体，并将副本融合成单个展开的循环嵌套。

相关参考

- *XL C/C++ Compiler Reference* 中说明了所有 XL C/C++ `pragma`。

移植注意事项

本节描述了几个常见的研究领域，这些研究可以加快将基于 UNIX 的应用程序移植到 AIX 平台上。

移植应用程序以在另一个平台上运行涉及源平台和目标平台。在写任何代码之前，至少应该考虑以下问题：移植到目标平台会发生什么变化？真正的移植只涉及更改硬件和操作系统。

理论上，写得好的程序不依赖于特定于平台的相关性，它们遵循业界标准（例如，POSIX）并且符合标准语言定义且没有使用非标准的语言扩展，除了重新编译和调试之外只需要进行少量的额外工作就可以将它们很方便地移植到新的操作系统。当源平台是相当新的基于 UNIX 的操作系统时，可限制更改以便更符合业界标准或同一标准的更新版本。如果应用程序已在 Linux 系统上运行，则可选择重新编译该应用程序并以本机方式在 AIX 上运行它。许多应用程序无须进行更改就可以重新编译和运行。

而且，使用不同的符合标准的编译器编译应用程序可以消除源代码中由于在语言标准实现方面的差别而导致的瑕疵。结果是应用程序更健壮。

在移植过程中产生的问题可归类为内部和外部可移植性问题。内部可移植性问题与编程语言固有的隐性假设有关。例如，C 程序假设整数中的特定字节顺序、整数的一组相对大小和结构中字段的特定布局。内部可移植性与程序代码与硬件之间的关系有关。此类移植问题受程序员控制。

另一方面，外部可移植性问题与程序采用的接口的语义、传递至程序或从程序传递来的自变量和返回值的选择有关。它们与程序依赖于的库的系统调用以及程序调用的但在程序外部的代码有关。如果程序使用标准化的外部接口，则程序员可控制外部可移植性。

语言固有的可移植性问题

本节描述与移植到 AIX 平台相关的一些内部可移植性注意事项。

- 检查对 GNU C 和其它语言扩展的依赖程度。严格符合其 ISO 语言规范的应用程序将最具有可移植性。IBM XL C/C++ 支持对 C 和 C++ 的 GNU C 和 C++ 扩展的一部分。可能需要修改依赖于不受支持的扩展的代码。
- 检查如何解引用空指针。由于与硬件相关的一些特征，代码中的某些错误在某一平台上可能会检测不到。当将程序移植到另一个平台时，这种错误可能会显示出来。如果 AIX 是源平台，则在移植之前，可使用选项 `-qcheck=nullptr` 来帮助检测这些情况。

最低的 4K 内存（即地址 0 至 4K-1）在 AIX 上是可读的并包含零，但在 Linux 和 Mac OS X 平台上是不可读的，如果在这些平台上访问它会导致分段违例。例如：

```
if (strcmp(a, NULL) == 0) ...
```

在 Linux 和 Mac OS X 上将导致分段违例，但在 AIX 上则不会。

- 检查对齐。AIX 上支持的对齐类型与在 Linux 或 Mac OS X 平台上支持的对齐类型不同，虽然名称可能是一样的。如果要移植依赖于 **-qalign** 或 **#pragma align** 的特定值的程序，您可能需要更改该程序。
- 确保数据结构的可移植性。如果您使用一个平台上的应用程序生成数据并使用另一个平台上的应用程序读取该数据，则该数据的对齐可能与读取应用程序所期望的对齐不同。要避免发生此问题，确保您对结构中的数据布局使用独立于平台的机制。例如，如果您使用 **#pragma pack(1)** 和 **#pragma pack(pop)** 对括住某一结构，则在所有平台上对齐都将是相同的。
- 使用 **gxlc** 或 **gxlc++** 实用程序来转换 makefile 中的命令。并非所有 gcc 或 g++ 选项都具有 XL C/C++ 等价项。
- 对 32 位、64 位应用程序或 128 位支持使用不同的编译器调用方式。
- 在 Linux 或 Mac OS X 上，如果调用缺省全局操作符 **new**，且不能执行分配请求，则将抛出类型为 **std::bad_alloc** 的异常。在 AIX 上，如果分配失败，则全局操作符 **new** 的缺省行为是返回空指针。
- 确保使用模板的应用程序的可移植性。C++ 编译器提供了处理模板文件的两种不同方法，作为手工维护源代码中的模板的替代方法。每种方法都具有相关联的编译器选项。**-qtemplateregistry** 编译器选项保存所有模板的记录。建议使用此方法。也为您从另一个平台移植的应用程序提供了更旧的编译器选项 **-qtempinc**。然而，在 Mac OS X 平台上，建议不要使用编译器选项 **-qtempinc**。

相关参考

- 下列 IBM 红皮书包含与移植相关的信息。其它红皮书可在以下网址在线获取：
www.redbooks.ibm.com。
- *AIX 5L Porting Guide* (2001 年)。
 - *Developing and Porting C and C++ Applications on AIX* (2003)。

编译时错误的诊断

移植项目的基本建议是使用选项 **-qinfo=por** 编译应用程序。**-qinfo** 的此子选项添加特定于可移植性的诊断消息。选项 **-qinfo=warn64** 指示编译器发出特定于将应用程序移植到 64 位方式的诊断消息。这些消息有助于缩小调查范围或查明特定编码构造。

下表显示对于检测并更正编译时错误可能很有用的其它选项。

编译时错误的其它诊断选项

选项	描述
-qsrcmsg	将下列信息打印至标准错误：编译器认为包含错误的源代码行、指向该源代码行中的特定点的信息（在源代码行下面一行）以及诊断消息。
-qsource	请求返回编译器清单。清单的各个部分包括带有行号的源代码、指定的选项、编译时使用的所有文件的清单、按严重性级别分类的诊断消息的摘要、已读取的行数以及编译是否成功。可以通过分别指定 -qattr 和 -qxref 选项来生成清单的属性和交叉引用部分。目标部分（要求指定选项 -qlist ）显示编译器生成的伪汇编代码，用来在您怀疑由于代码生成错误导致程序不按期望执行时诊断执行时问题。

选项	描述
-qsuppress	阻止编译器发出特定消息。可以通过在用冒号分隔的列表中列示消息号来禁止发出多条消息。
-qflag	阻止向终端和清单文件发出指定的诊断消息。该选项使用缺省编译器消息格式的单字母严重性代码来指定某个级别，低于该级别的消息应被忽略。

32 位和 64 位应用程序开发

可以使用 XL C/C++ 来开发 32 位和 64 位应用程序。本节包含用于将 C 和 C++ 程序从 32 位方式移至 64 位方式的参考信息和其它可移植性注意事项。

如果只更改硬件和操作系统，则移植过程是真正的移植。作为迁移至 AIX 的过程的一部分，将 32 位应用程序移至 64 位编程模型意味着该过程不再是真正的移植，而是开发活动。当移植至 64 位环境时，具有下列特征中的任何一个的 32 位应用程序很可能需要更改：

- 直接读取和解释内核存储器。
- 使用 /proc 文件系统来访问 64 位过程。
- 使用只具有 64 位版本的库。
- 是设备驱动程序。
- 正在从与 AIX 具有互操作性问题的源平台移植。

在 64 位方式下进行开发允许应用程序利用更新和更快的 64 位硬件和操作系统来提高大型的、复杂的耗费大量内存的程序（例如，数据库和科学应用程序）的性能。受 32 位地址空间限制的应用程序（例如，数据库应用程序、Web 搜索引擎和科学计算应用程序）可受益于目标为 64 位方式的转换。64 位方式下的 I/O 绑定应用程序还可通过将数据保存在内存中而不是写至磁盘来提高性能，因为磁盘 I/O 通常比内存存取要慢。

能够直接在物理内存中处理较大的问题或许是 64 位机器最显著的性能优点。但是，某些应用程序在 32 位方式下编译比起在 64 位方式下重新编译来说执行性能更好一些。产生这种情况的一些原因是：

- 64 位程序大一些。程序越大，对物理内存的要求就越高。
- 64 位长整型除法比 32 位整数除法更耗时。
- 使用 32 位有符号整数作为数组下标的 64 位程序每当引用数组时都可能需要其它指令来执行符号扩展。

64 位应用程序的其它缺点是它们需要更多的堆栈空间来存放更大的寄存器。因指针大小更大，应用程序高速缓存占用量也更大。64 位应用程序不能在 32 位平台上运行。

下面列示了补偿 64 位程序的性能缺点的一些方法。

- 避免执行混合 32 位和 64 位的操作。例如，32 有符号数据类型与 64 位数据类型相加要求对 32 位类型进行符号扩展以根据符号设置寄存器的高 32 位。设置寄存器的高位会减慢计算速度。如果 32 位类型是无符号的，则将对高位清零。
- 尽可能避免 64 位长整型除法。乘法运算通常比除法运算快。如果需要使用同一除数执行多个除法运算，则将该除数的倒数赋予一个临时变量，然后将所有除法运算更改为与该临时变量相乘。例如，函数

```
double preTax(double total) { return total * (1.0 / 1.0825); }
```

将比下面这种直接一些的方式执行速度要快:

```
double preTax(double total) { return total / 1.0825; }
```

原因是在编译时由于常量合并而只对除法 (1.0 / 1.0825) 求值一次。如果 **-qnostrict** 选项生效 (当在优化级别 **-O2** 及更高级别进行编译时就会生效), 则优化通常由编译器进行。

- 将 **long** 变量用作数组下标而不是有符号、无符号和无格式的 **int** 类型。这样做使得编译器在引用数组时无需执行符号扩展。

如果移至 64 位编程模型, 则写得好的代码只需要进行极少量的修改和调试就可正确编译和运行。术语写得好意味着遵循语言标准且具有良好的编程习惯。在移至 64 位编程模型的上下文中, 该术语还有这样的含义: 代码既不依赖于特定字节顺序也不依赖于外部数据格式, 它使用函数原型、适当的系统头文件和系统派生的数据类型。

AIX 上的 32 位开发环境和 64 位开发环境

C 和 C++ 编译器与 AIX 操作系统提供了两种不同的编程模型: ILP32 和 LP64。ILP32 是 32 位整数 / 长整数 / 指针 (integer/long/pointer 32) 的缩写, 它是 AIX 上的本机 32 位编程环境。ILP32 数据模型提供了 32 位的地址空间, 理论上内存限制为 4 GB。LP64 是 64 位长整数 / 指针 (long/pointer 64) 的缩写, 它是 AIX 上 64 位编程环境, 也是来自所有主要系统供应商的 64 位基于 UNIX 的系统的事实上的标准数据模型。总的来说, 除了数据类型大小和对齐之外, LP64 支持与 ILP32 模型相同的编程功能部件并且与最广泛使用的 **int** 数据类型向后兼容。

编译器支持

缺省情况下, 编译器调用会以 32 位方式调用编译器和链接程序。提供了下列功能部件以支持 64 位开发:

- **__64BIT__** 预处理器宏, 它是在以 64 位方式下进行编译时预定义的。该宏使程序员可以在同一源文件中对 32 位和 64 位执行选择不同的代码行。
- **OBJECT_MODE** 环境变量。此环境变量更改缺省编译方式以接受 32 位和 / 或 64 位方式的目标。在 C 和 C++ 应用程序开发中经常用到的许多 AIX 实用程序会使用此环境变量。但是, 即使 AIX 提供的大部分 C 和 C++ 库是混合方式归档 (同时包含 32 位和 64 位目标), 编译器和链接程序也都不支持 **OBJECT_MODE=32_64**。
- **-q64** 编译器选项。此选项设置 64 位方式支持, 它与 **-qarch** 选项一起使用, 后者指定目标体系结构的指令集。**-q32** 和 **-q64** 优先于 **-qarch** 选项的设置。当 **-q32** 与 **-q64** 发生冲突时, 使用后面指定的选项。
- 对 64 位编译的 **-qarch** 支持。此选项指示编译器生成可以在给定 64 位体系结构获取最优执行的代码。因此, 只接受 **-qarch** 与 **-q64** 的某些组合。

请参阅 *XL C/C++ Programming Guide* 以获取更多信息。

AIX 实用程序命令支持

AIX 操作系统提供了大量实用程序命令, 它们处理目标文件并且是在 AIX 上处理目标和库所必需的。当使用 **-X** 选项 (它指定实用程序处理的目标文件的位方式) 进行调用时, 这些实用程序支持 64 位 XCOFF 目标格式。“扩展公共目标文件格式” (XCOFF) 是 AIX 的目标格式。**-X** 的可能值有;

32 只处理 32 位目标文件。

64 只处理 64 位目标文件。
32_64 处理 32 位和 64 位目标文件。

用于处理共享目标和库的 AIX 实用程序命令

命令	描述
ar	维护链接编辑器使用的带索引的库。当在 32 位方式下时，会忽略 64 位目标而不进行提示。
dump	显示目标文件的已选择的部分。在链接和装入的上下文中，dump 命令用来检查可执行文件和共享目标的头信息。dump -H 命令使您能够在运行时通过显示头信息来确定可执行文件或共享目标对于符号解析的相关性。dump -Tv 命令显示共享目标或可执行文件的符号定义：链接时目标导出的符号、在装入时目标或可执行文件将尝试导入的符号以及包含这些符号的共享目标的名称（如果知道的话）。
file	显示文件的位方式以及该文件是否已被剥离。
genkld	列示装入在系统内存（具体来说是在系统共享库段）中的共享目标。专用共享目标不会显示在 genkld 命令输出中，即使它们已被装入到内存中。
ldd	列示将装入以启动可执行文件的共享目标和归档成员。
lorder	查找目标库中的成员文件的最佳顺序。
nm	显示有关目标文件、可执行文件和目标文件库中的符号的信息。nm -g 命令处理指定库归档中包含的所有归档成员。与 dump 不同，nm 不显示将提供符号的共享目标名或归档成员名。
ranlib	将归档库转换为随机库。
rtl_enable	将为缺省链接、静态链接或延迟装入编译的共享库转换为支持运行时链接的库。
size	显示 XCOFF 目标文件的段大小。
slibclean	只允许 root 用户从系统共享库段卸装使用计数为零的所有共享目标。该实用程序供处于软件维护阶段（特别是在除去不再需要的应用程序之前或在更新已安装的应用程序之前）的生产系统上使用。
strip	通过除去绑定程序使用的信息和符号调试信息来减小 XCOFF 目标文件的大小。

相关参考

- AIX 5L Version 5.2 Reference Documentation: Commands Reference

AIX 上的目标和库

与其它 UNIX 操作系统相似，AIX 操作系统提供了用于创建、开发、测试和调试共享库以及使用它们的应用程序的设施。AIX 共享库功能部件大体上与其它 UNIX 操作系统兼容。然而，AIX 还提供了用于创建和使用动态绑定的共享库的设施。借助动态绑定，装入程序会在运行时解析用户代码中引用的、在共享库中定义的外部符号。动态链接使用较少的内存来运行程序并生成较小的可执行文件。另外，AIX 支持动态装入（由一组子例程而不是链接程序选项或特殊目标文件类型提供的编程方案）。

要实现和支持这些不同的链接方法，AIX 要求文件格式为“扩展公共目标文件格式”（XCOFF）。在讨论目标和共享库时，注意在 AIX 上进行应用程序开发的上下文中使用的术语的精确含义很重要。

目标文件是包含可执行代码、数据、重定位信息、符号表和其它信息的文件的一般术语。可以将多个目标文件归档到单个库归档文件（也可以简称为库）中。包含在库中的目标文件称为归档成员。与多个目标文件相比，库的优点是创建可执行文件所需处理的文件较少。

要在 AIX 上构建可执行文件，所有参与的目标文件和归档成员都必须处于同一位方式。但是，库归档既可包含 32 位目标模块又可包含 64 位目标模块。AIX 提供的大部分系统库都处于混合方式。AIX 操作系统提供了用于查询、维护以及处理目标和库的命令实用程序，例如，**ar** 用于创建和维护库归档文件，**file** 用于显示目标文件的位方式，**dump** 用于查询归档中的符号，而 **nm** 用于查询库中的可用符号（给定特定定位方式）。

AIX 上共享目标与库之间的差别

术语共享库和共享目标在其它 UNIX 操作系统通常可以互换。其它 UNIX 操作系统称为共享库（也称为动态链接库或 DLL）的东西在 AIX 上称为共享目标。

在 AIX 上，共享库与共享目标之间存在一个明显的差别。当使用几个系统库链接程序时，共享目标是可执行文件的从属模块。在 AIX 上，除了在同一个库中之外，共享目标的名称不必是唯一的。同名但在不同库中的共享目标被看作不同的模块。AIX 提供的大部分系统库都包含一个或多个共享目标。

共享目标是在它的 XCOFF 头中具有 SHROBJ 标志的单个目标文件。在 AIX 上，共享目标的命名约定是 *name.o*，它使用编译器生成的缺省文件扩展名。

在 AIX 上，共享库指的是 **ar** 命令创建的归档库文件，其中的一个或多个归档成员是共享目标。库还可以包含非共享目标文件（也称为静态目标文件）。在 AIX 上，共享库的命名约定是使用格式 *libname.a*，这是链接程序期望的文件扩展名。

XCOFF（扩展公共目标文件格式）是 AIX 操作系统的目标文件格式。它是机器映像目标和可执行文件的正式定义。这些目标文件主要由绑定程序和系统装入程序使用。XCOFF 扩展了标准的“公共目标文件格式”（COFF）以规定动态链接和目标文件中的单元的替换。XCOFF 还规定了既有 32 位也有 64 位实现的目标和可执行文件。

可以编写一些程序来了解 32 位 XCOFF 文件和 / 或 64 位 XCOFF 文件。可以在任一位方式下编译这些程序以创建 32 位或 64 位程序。通过定义预处理器宏，应用程序可以从 XCOFF 头文件中选择正确的结构定义。

汇编程序和编译器生成 XCOFF 文件作为输出。绑定程序将各个目标文件合并成一个 XCOFF 可执行文件。系统装入程序读取 XCOFF 可执行文件以创建程序的可执行内存映像。符号调试器读取 XCOFF 可执行文件以提供对可执行内存映像的函数和变量的访问。

AIX 上共享目标与静态目标之间的差别

在许多 UNIX 操作系统上，文件扩展名 *.so* 指示共享目标的文件，扩展名 *.o* 指示静态目标的文件。在 AIX 上，静态目标文件和共享目标文件都使用文件扩展名 *.o*。对于共享目标文件，约定（而不是要求）使用扩展名 *.o*：链接程序读取文件头来确定文件的内容。共享目标文件不同于静态目标文件，区别在于共享目标文件是完全链接和解析的，可以装入和执行，并且 XCOFF 文件头的 FLAGS 字段设置了 SHROBJ 位。SHROBJ 位是在动态引用目标时设置的。否则，链接程序会将目标看作静态的。

AIX 还支持文件名为 `.so` 的共享目标。除了约定的 `.o` 文件扩展名之外，在 AIX 上可以使用任何扩展名命名共享目标文件。在 AIX 上，文件扩展名 `.so` 有时用来指示运行时链接。

AIX 操作系统提供了各种 AIX 命令来显示文件的 XCOFF 头以确定目标是共享的还是静态的。`dump -ov` 命令显示指定文件的 XCOFF 头。`dump -gov` 命令使您可以检查指定库的归档成员来确定归档成员是共享的还是静态的。

链接时和装入时

在 AIX 上，共享目标和库在创建和执行程序时的两个阶段中使用。

在链接时，链接编辑器会搜索指定的共享目标和库以解析创建可执行文件所需的所有未定义符号。如果共享目标包含被引用的符号，则产生的可执行文件的 XCOFF 头的装入程序部分就包含对该共享目标的引用。对于作为共享目标的库成员中的被引用符号，情况相同。就会从这些共享目标或库中导出符号并将它们导入到该可执行文件中。

在程序装入时，系统装入程序会读取可执行文件中的 XCOFF 头信息并尝试找到被引用的共享库。如果找到了所有被引用的共享目标和库，就会启动可执行文件。系统装入程序尝试将程序可执行组件的各部分装入到进程地址空间中的适当段中。包含在共享目标和库中的程序文本会被需要它的第一个程序装入全局系统内存，随后使用的它的所有程序可共享它。

链接时错误的诊断

常见的链接时错误有符号无法解析、符号重复和用于链接程序的内存不足。

符号无法解析

当应用程序与许多库（特别是第三方产品提供的库）链接在一起时，通常会发出链接程序错误警告。当不能解析外部符号时，链接会失败且不会生成可执行文件。产生符号无法解析错误最常见的原因是缺少输入文件。例如，如果调用了不在缺省 C 运行时库 `libc.a` 中的库函数，则需要指定找到符号的归档库文件。

系统链接编辑器接受诸如目标文件、共享目标文件、归档目标文件、库以及导入文件和导出文件之类的输入。如何找到定义了无法解析的符号的库文件呢？可以搜索产品文档、在链接命令中包括所有提供的库并让链接编辑器查找该符号的位置或者使用 `nm` 命令来亲自查找它。

某些链接程序选项会生成一些日志文件，可以分析这些日志文件来确定引用无法解析的符号的库或目标文件。日志文件可以跟踪错误使用的相互依赖的库或多余的库。

- **-bnoquiet** 选项将每个绑定程序子命令及其结果写至标准输出。任何无法解析的符号引用都会出现在该输出中。输出还列示从指定库模块导入的符号。
- **-bmap:filename** 选项生成地址映射。无法解析的符号列示在文件的顶部，随后是导入的符号。
- **-bloadmap:filename** 选项生成一个日志文件，该文件包括有关传递至链接程序的所有自变量、正在读取的共享目标和正在导入的符号数的信息。如果找到了无法解析的符号，则该日志文件会列示引用了该符号的目标文件或共享目标。要在第三方产品提供的库中查找无法解析的符号，可以使用该日志文件来搜索产品提供的其它库。

符号重复

符号重复错误通常说明存在编程错误。拥有同名的多个外部函数定义是不正确的。当源代码包含多个外部函数定义时，链接编辑器使用它遇到的第一个定义，结果有可能不是期望的。建议的解决方案是更改函数名或使用静态函数。

在 C++ 中，当模板是在多个源文件中隐式实例化时，使用该模板函数也会在链接时产生符号重复错误。建议的解决方案是使用 **-qtemplateregistry** 选项和 VisualAge C++ Professional V6 引入的模板处理机制。

导致符号重复的另一个原因是内联函数。每个内联的函数都会创建出现为单独实例化的符号表条目。使用选项 **-qweaksymbol** 在 AIX 5.2 上进行编译可减少这些警告的数目。

用于链接程序的内存不足

链接非常大的文件会耗尽允许链接程序进程使用的内存。解决方案是增加调用命令的用户的调页空间或资源限制。**ulimit** 命令控制用户调用链接程序的资源限制。另外，可以考虑在 32 位大型内存模型中运行链接程序进程。

运行时错误的诊断

程序可能会成功编译和链接，但在执行时产生意外的结果。编译器不能诊断不违反语言的语法的编程错误。本节描述一些常见错误、如何检测它们以及如何更正它们。

变量未初始化

具有自动存储器持续时间的对象是不会隐式初始化的，因此，它的初始值是不确定的。如果在设置它之前使用了 **auto** 变量，则每次运行程序时，它可能会也可能不会产生相同的结果。**-qinfo=gen** 编译器选项显示设置之前使用的 **auto** 变量的位置。**-qinitauto** 选项指示编译器将所有自动变量初始化为指定的值。此选项会降低应用程序的运行时性能，建议只用于调试。

运行时检查

-qcheck 选项将运行时检查代码插入到可执行文件中。这些子选项指定对空指针、数组下标超界和被零除的情况进行检查。与 **-qinitauto** 相似，**-qcheck** 会降低应用程序性能，建议只用于调试。

ANSI 别名判别

基于类型的别名判别（也称为 ANSI 别名判别）限制可安全地用来访问数据对象的左值。当在强制遵从语言标准的语言级别下进行编译时，在优化期间，C 和 C++ 编译器会强制使用基于类型的别名判别。ANSI 别名判别规则声明只能解引用指向相同类型或兼容类型的指针。例外情况是符号和类型限定符不遵从基于类型的别名判别，并且字符指针可指向任何类型。将指针强制转换为不兼容的数据类型然后对它解引用这种常见的编码作法违反了此规则。

通过设置 **-qalias=noansi** 关闭 ANSI 别名判别可更正程序行为，但是这样做会减少编译器优化应用程序的机会，从而降低了运行时性能。建议的解决方案是将程序更改为符合基于类型的别名判别规则。

#pragma option_override

有时，仅当使用优化时才会出现错误。最好（尤其是对于复杂的应用程序）是能对已知包含编程错误的函数关闭优化但允许对程序的其余部分进行优化。`#pragma option_override` 伪指令使您可以为特定函数指定其它优化选项。

`#pragma option_override` 伪指令还可用来确定产生该问题的函数。发现方法是有选择地对伪指令中的每个函数关闭优化直到问题消失为止。

共享内存并行化

提供了几项成熟技术以便可以并行执行程序并且使作业完成速度比在单处理器上运行更快。这些技术包括：

- 基于伪指令的共享内存并行化（SMP）
- 指示编译器自动生成共享内存并行化
- 基于共享或分布式内存并行化（MPI）的消息传递
- POSIX 线程（pthread）并行化
- 使用 `fork()` 和 `exec()` 的低级 UNIX 并行化

毫无疑问，应用程序的可移植性需求是选择要使用的最佳技术的因素。另外，该选择很大程度上取决于应用程序、程序员的技术和偏好以及目标机器的性能。在 AIX 上实现并行化的两种主要方法是通过使用手工编写的 POSIX 线程（pthread）和 OpenMP 伪指令。

AIX 操作系统的并行编程设施基于线程的概念。并行编程充分利用多处理器系统的优点，同时维护与现有单处理器系统的完全二进制兼容。这意味着在单处理器系统上运行的多线程程序可以利用多处理器系统而无需重新编译。

Pthread 在高效使用多处理器方面提供了很大的灵活性，这是因为它们提供了对并行化过程的最大程度的控制。这一平衡会明显增加代码的复杂性。在许多情况下，通过使用 OpenMP 伪指令、支持 SMP 的库或编译器的自动 SMP 功能的较简单方法更可取。显式使用线程不一定能得到更好的性能。调试多线程的应用程序尤其不可取。但是，在某些程序中，它是唯一适用的方法。

下面列示了不同技术的一些利弊。

编译器完成的自动并行化

- 易于实现（用 `-qsmp=auto` 编译）。
- 易于开展小组工作。
- 可伸缩性有限，这是因为会忽略数据作用域。
- 依赖于编译器（即使在发行特定编译器时也不例外）。
- 无需移植。

支持 SMP 的库

- 需要工作量最小。
- 无需移植（通常是专用的）。
- 灵活性有限。

OpenMP 伪指令

- 可移植。
- 可提高自动并行化的可伸缩性。
- 采用统一内存访问。

混合方法（OpenMP 和 pthread、或者 UNIX fork() 和 exec() 并行化、特定于平台的构造的子集或混合）

- 可能支持小组工作。
- 需要反复测试的概念来确保性能和可移植性。
- 无需移植。

pthread

- 可移植。
- 提供对并行化过程的最大程度的控制。
- 可提高自动并行化的可伸缩性。
- 需要有经验的程序员来处理代码复杂性。

OpenMP 伪指令

OpenMP 伪指令是一组命令，它们指导编译器应如何并行化特定循环。源代码中存在这些伪指令可使得编译器无需对并行代码执行任何并行分析。使用 OpenMP 伪指令要求存在 pthread 库以提供用于并行化的必要基础结构。

OpenMP 伪指令解决并行化应用程序的三个重要问题。第一，子句和伪指令可用于限定作用域的变量。通常，不应共享变量；即，每个处理器都应具有它自己的变量副本。第二，共享伪指令的工作指定应如何在 SMP 处理器中分发包含在代码的并行区域中的工作。最后，存在一些用于处理器之间的同步的伪指令。

编译器支持 OpenMP V2.0 规范。

相关参考

- 第 61 页的附录 B，『OpenMP 一致性和支持』

AIX 上的线程

在 AIX 上线程的实现归类为外部可移植性问题，这是因为它与外部接口及其假定语义（程序使用这些语义）有关。AIX 线程库符合 Single UNIX Specification V2，它包括 IEEE POSIX 1003.1c 标准，称为 POSIX 线程标准。该库还符合 Open Group 98 规范，此规范将扩展的线程函数添加至 POSIX 线程标准。因此，pthread 的标准化接口可确保线程程序的可移植性。

AIX 上的 POSIX 线程库

在 AIX 上，将 POSIX 线程（pthread）定义为一组 C 语言编程类型和子例程调用，用头文件（/usr/include/pthread.h）和 POSIX 线程库（/usr/lib/libpthreads.a）实现。

以下数据类型是在 pthread.h 头文件里为线程库定义的。这些数据类型的定义会因系统而异。

pthread_t	标识线程。
pthread_attr_t	标识线程属性对象。
pthread_cond_t	标识条件变量。
pthread_condattr_t	标识条件属性对象。
pthread_key_t	标识特定于线程的数据键。
pthread_mutex_t	标识互斥锁。
pthread_mutexattr_t	标识互斥锁属性对象。
pthread_once_t	标识一次初始化对象。

AIX 提供了对为 POSIX 线程标准版本 7 的草稿版编写的现有多线程应用程序的二进制兼容。提供兼容性 POSIX 线程库 `/usr/lib/libpthreads_compat.a` 只是为了与那些应用程序向下兼容。兼容性 POSIX 线程库仅支持 32 位应用程序。

以下操作系统文件提供了 pthread 的 AIX 实现:

/usr/include/pthread.h	具有大部分 pthread 定义的 C/C++ 头文件。
/usr/include/sched.h	具有部分调度定义的 C/C++ 头文件。
/usr/include/unistd.h	具有 pthread_atfork() 定义的 C/C++ 头文件。
/usr/include/sys/limits.h	具有部分 pthread 定义的 C/C++ 头文件。
/usr/include/sys/pthdebug.h	具有大部分 pthread 调试定义的 C/C++ 头文件。
/usr/include/sys/sched.h	具有部分调度定义的 C/C++ 头文件。
/usr/include/sys/signal.h	具有 pthread_kill() 和 pthread_sigmask() 定义的 C/C++ 头文件。
/usr/include/sys/types.h	具有部分 pthread 定义的 C/C++ 头文件。
/usr/lib/libpthreads.a	提供 UNIX98 和 POSIX 1003.1c pthread 的 32 位 / 64 位库。
/usr/lib/libpthreads_compat.a	提供 POSIX 1003.1c Draft 7 pthread 的仅 32 位库。
/usr/lib/profiled/libpthreads.a	提供 UNIX98 和 POSIX 1003.1c pthread 的概要 32 位 / 64 位库。
/usr/lib/profiled/libpthreads_compat.a	提供 POSIX 1003.1c Draft 7 pthread 的仅概要 32 位库。

线程不安全的库只能由程序中的一个线程使用。线程不安全的代码只能由主线程安全使用。这些限制是由于访问全局 `errno`（而不是特定于线程的 `errno`）产生的并且也适用于主线程执行的库调用。

AIX 上的 Pthread

传统做法是，使用多个单线程进程来获得并行性，但某些程序可得益于更精细的并行性程度。多线程进程在进程中提供了并行性并且共享涉及编写多个单线程进程的许多概念。

本节讨论有关 AIX 中并行编程设施的实现，并与在其它 UNIX 系统中的情况作比较。

术语: 在传统的单线程进程系统中，线程和进程特征归类到称为进程的单个条目中。在其它系统中，线程有时候与轻量级进程同义，或者词语线程的意义有时与进程只有细微的差别。

在 AIX 上，区分了内核线程与用户线程。在 AIX 上，术语轻量级进程通常指内核线程。用户线程是独立的控制流，它运行的地址空间与进程中其它独立控制流所在的地址空间相同。本讨论的余下部分中，术语线程指的是用户线程。

内核线程是系统调度程序可处理的可调度实体。在 AIX 上，内核线程在进程内运行，但可由系统中的任何其它线程引用。但是，内核线程不能由程序员直接控制。内核线程非常依赖于实现，因此库（如 POSIX 线程库）为用户提供线程以方便编写可移植的程序。

用户线程是程序员用来处理程序中的多个控制流的实体。用来处理用户线程的标准化 API 由线程库提供。用户线程只能存在于进程中并且不能引用另一进程中的用户线程。库使用专用接口来处理用于执行用户线程的内核线程。

属性: 在传统的单线程进程系统中，进程具有一组属性。在如 AIX 之类的多线程进程系统中，进程与线程之间的属性是划清界线的。

多线程系统中的进程是可更改的实体。必须将它看作一个执行框架。它具有传统进程的属性，如进程标识、进程组标识、用户标识、组标识、环境和工作目录。进程还提供公共的地址空间和公共的系统资源，如文件描述符、信号操作、共享库和进程间通信工具。

线程是可调度实体，只具有确保它是独立控制流所需的那些属性。这些属性包括堆栈、调度策略或优先级、暂挂和阻塞信号集以及特定于线程的数据。特定于线程的数据的一个示例是 **errno** 错误指示符。在 AIX 上，每个线程都具有它自己的 **errno** 错误指示符。在多线程系统中，**errno** 通常是返回特定于线程的 **errno** 值（而不是全局变量）的子例程。其它系统可能会提供 **errno** 的其它实现。

不能将进程中的线程看作一组进程。所有线程共用同一地址空间。这意味着两个线程中具有相同值的两个指针引用相同的数据。另外，如果任何线程更改了其中一个共享的系统资源，则进程中的所有线程都受影响。例如，如果线程关闭了一个文件，则会对所有线程关闭该文件。

线程不维护已创建线程的列表，也不知道创建它的线程。线程创建与进程创建的不同点在于线程之间不存在父子关系：除了初始线程之外的所有线程都在同一个层次级别上。初始线程由操作系统在创建进程时自动创建。

必须在创建线程之前定义线程属性对象（它概括线程的属性）。在创建时还必须指定入口点例程和自变量。每个线程都具有带一个自变量的入口点例程，但若干个线程可使用同一个入口点例程。

线程模型和虚拟处理器: 虚拟处理器（VP）将用户线程映射至线程库中的内核线程。进行映射的方式称为线程模型。在 AIX 上，缺省线程模型是 M:N 模型，在这种模型中，所有用户线程都映射至内核线程池；所有用户线程都在虚拟处理器池上运行。这是效率最高但最复杂的线程模型，在此模型中，用于用户线程编程的工具在线程库和内核线程之间共享。M:N 模型包含了单用户线程绑定至特定虚拟处理器（1:1 线程模型）和所有未绑定用户线程共享其余 VP 的情况。

1:1 线程模型将每个用户线程映射至一个内核线程；所有用户线程都在一个 VP 上运行。用于用户线程编程的大部分设施都由内核线程直接处理。1:1 模型可通过将 AIXTHREAD_SCOPE 变量的值设置为 S 来启用。

M:1 线程模型可在任何系统上使用，尤其可在传统的单线程系统上使用。库调度程序将所有用户线程都映射至一个内核线程；所有用户线程都在一个 VP 上运行。用于用户线程编程的所有设施都完全由库完成。

pthread 生命周期概要： 线程程序中的每个线程都有它自己的专用程序计数器、堆栈和寄存器。内存状态和文件描述符是共享的。

pthread.h 头文件

pthread.h 头文件必须是使用线程库的每个源文件的第一个要包括的文件，这样才能确保使用线程安全的子例程。该头文件包含所有子例程原型、宏和其它定义以使用线程库。它还将 **errno** 全局变量重新定义为返回特定于线程的 **errno** 值的函数。因此，**errno** 标识不再是多线程程序中的左值。

下列全局符号是在 pthread.h 文件中定义的：

POSIX_REENTRANT_FUNCTIONS	指定所有函数都应该是可重入的。
POSIX_THREADS	指定 POSIX 线程 API 的可用性。

编译器调用

线程应用程序应该使用编译器的其中一个带 **_r** 后缀的调用来编译和链接。存在一个可重入的调用命令，它们确保使用充分且适当的选项以及程序与可重入的线程安全的库链接在一起。例如，**xlC_r** 定义符号 **_THREAD_SAFE** 并与 pthreads 库链接在一起。

线程创建

执行 pthread 程序就如操作系统创建的单个线程那样开始。根据需要，创建和终止其它线程以并发地将工作调度到可用的处理器。

初始线程之外的线程是使用 pthread_create 函数创建的。此函数具有四个自变量：线程标识，它在成功完成时返回；指向线程属性对象的指针；线程将执行的函数；以及要传递至线程函数的自变量。线程函数采用单个指针自变量（类型为 **void ***）并返回一个指针（类型为 **void ***）。实际上，线程函数的自变量通常是指向结构的指针，而结构可能包含线程函数可访问的许多数据项。

程序可创建固定数目的线程。但是，在许多情况下，在通过设置环境变量来提供覆盖缺省行为的能力时，在运行时让程序确定要创建多少线程很有用。例如，对于 OpenMP 程序，缺省情况是有多少处理器可用就创建多少线程。在 AIX 中，可通过从 libc 调用 sysconf 例程来获取在线处理器数。

线程终止

线程在线程函数执行完成时隐式终止。线程可通过调用 pthread_exit 显式终止它自己。这种情况也是可能的：一个线程通过调用 pthread_cancel 函数终止其它线程。

初始线程具有特殊的属性。如果初始线程到达了其执行流的末尾并返回，则会调用出口例程，并且在那时候将终止属于该进程的所有线程。但是，初始线程可以创建拆离

线程，然后安全地调用 `pthread_exit`。在这种情况下，余下的线程将继续执行它们的线程函数，而进程将保持活动，直到最后一个线程退出为止。在许多应用程序中，为初始线程创建一组线程然后在继续或退出之前等待它们终止很有用。可使用可连接的线程达到此目的。在 AIX 上，缺省设置是 `detached`。函数 `pthread_join` 暂挂调用的线程，直到被引用的线程终止为止。当假定 N 个线程并发运行于 N 个处理器上时，`AIXTHREAD_SCOPE` 环境变量的系统作用域属性是正确的。

同步： 在多线程程序中，若干个控制流可同时访问相同的函数和相同的资源。为了保护资源的完整性，为多线程程序编写的代码必须是可重入的并且是线程安全的。可重入和线程安全性是与函数有关的独立概念。函数可以是可重入的和 / 或线程安全的，或者既不可重入也不是线程安全的。

可重入的函数不保留连续调用的静态数据，也不返回指向静态数据的指针。所有数据都由函数的调用程序提供。可重入函数也不能调用非可重入函数。在大多数情况下，非可重入函数将需要用具有修改过的接口的函数替换以使函数成为可重入函数。非可重入函数通常也是线程不安全的。

线程安全的函数使用锁来防止并发访问共享资源。线程安全性只涉及函数的实现，不影响它的外部接口。在多线程程序中，多个线程调用的所有函数都必须是线程安全的。

不经过显式同步或序列化就使用全局数据是线程不安全的。应对每个线程维护或封装全局数据以便可序列化对它的访问。线程可能会读取到对应于另一线程导致的错误的错误代码。在 AIX 中，每个线程都具有它自己的错误值。

一些标准 C 子例程都是非可重入的，如 `ctime` 和 `strtok` 子例程，即使它们属于认为是线程安全的库。子例程的可重入版本名称是原始子例程的名称加 `_r` 后缀。

线程私有变量与线程共享变量之间的区别对于程序的正确性和性能至关重要。对于使用 OpenMP 伪指令获取共享内存并行化的程序也是如此。必须同步对共享变量的访问以避免发生冲突以及确保能获得正确结果。但是，使用同步需要与性能和可伸缩性的降低相权衡。

共享内存的并行编程的主要困难是找出局部变量与全局变量的最佳平衡点，因为作用域限定定义了哪些变量是专有的，哪些变量是共享的。全局变量的争用（如在归约求和中）是产生性能问题的主要原因。引入临时的局部变量通常有助于解决此类问题。

在多线程应用程序中，更新共享内存位置通常可保护互斥锁。操作系统确保序列化对共享数据的访问。在给定时间，只有一个线程可进入锁定与解锁之间的区域来修改数据。

在 AIX 进程之间共享内存： AIX 和大多数 UNIX 系统都允许几个进程共享公共数据空间（称为共享内存）。条件变量和互斥锁的进程共享属性都允许在共享内存中分配这些对象以支持在属于不同进程的线程之间同步。但是，由于共享内存管理没有业界标准接口，所以未在 AIX 线程库中实现进程共享 POSIX 选项。

调试具有多线程的程序

AIX 操作系统提供了 `dbx` 命令，该命令对 C/C++ 程序和 Fortran 程序调用符号调试程序。`dbx` 命令具有用于显示与线程有关的对象（包括属性、条件、互斥锁和线程）的子命令。


对于内核编程，操作系统提供了内核调试程序。有关更多信息，请参阅 *AIX 5L Version 5.2 Kernel Extensions and Device Support Programming Concepts*，它是 AIX 文档的一部分。


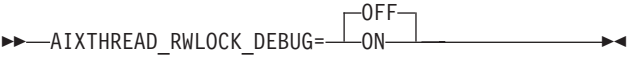
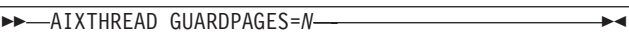
调整多线程程序

调整多线程程序的一个方面涉及调整线程生命周期中不同状态所花的时间。各种线程技术函数控制一系列生命周期事件消耗的空间和资源。在线程的并行工作完成之后，线程等待（运转）一定时间（运转等待时间），但它在等待时会消耗处理器时间。经过了运转等待时间之后，如果指定了产生等待时间，则线程可为另一可运行的线程空出它在内核线程中的位置。如果超过了产生等待时间，则线程会进入休眠状态。与处于产生状态的线程相比，从休眠状态重新激活线程需要更多的资源。

AIX 操作系统提供了修改各种线程技术函数行为的环境变量。下表显示了在调整多线程程序时会很有用的所选环境变量。

影响线程技术的所选环境变量

描述	语法
AIXTHREAD_SCOPE 线程争用作用域控制应用程序级别 pthread 至系统调度队列中的实体的映射。AIXTHREAD_SCOPE 通过不必修改应用程序就可更改线程作用域的能力允许探索应用程序行为。单个进程可包含两个作域的 pthread。这可在 pthread_create() 的属性自变量不是 NULL 时获得。然后，属性结构的内容确定线程的作用域。	语法 AIXTHREAD_SCOPE=  其中 P 表示进程争用作用域（缺省值），隐含 M:N 线程模型。最好在线程数比处理器多时使用。 S 表示系统争用作用域，隐含 1:1 线程模型。每个用户线程直接映射至一个内核线程。
AIXTHREAD_MNRATIO 控制在创建和终止 pthread 时在 pthread 库中使用的比例因子。仅适用于作用域为进程的线程。缺省 M:N 比是 8:1，即，每个内核线程对八个 pthread。通过设置并导出右边的环境变量修改 M:N 比。	语法 AIXTHREAD_MNRATIO= <i>M:N</i> 其中 <i>M</i> 表示用户线程数。缺省值为 8。 <i>N</i> 表示内核线程数。缺省值为 1。
SPINLOOPTIME 控制在互斥锁可用性上造成阻塞之前 pthread 将尝试获取互斥锁的次数。通过设置并导出右边的环境变量来修改运转循环。	语法 SPINLOOPTIME= <i>N</i> 其中 <i>N</i> 表示在产生另一 pthread 之前重试繁忙锁定的次数。缺省值为 40。
YIELDLOOPTIME 控制在尝试获取繁忙运转锁定时在转入休眠之前系统产生处理器的时间。如果 pthread 运转在锁定上并且不能获得它，则 thread 将被转入休眠状态。增大产生循环时间的值可防止 pthread 延迟 pthread 进入休眠状态。仅当同时设置了 SPINLOOPTIME 时此变量才可用。产生循环时间的值可通过设置并导出右边的环境变量来修改。	语法 YIELDLOOPTIME= <i>N</i> 其中 <i>N</i> 表示产生以获得繁忙互斥锁或运转锁定的次数。缺省值为 0。
AIXTHREAD_MINKTHREADS 设置应该用于进程的最小内核线程数。平均线程进程将至少具有此数目的内核线程可用于安排的 pthread。	语法 AIXTHREAD_MINKTHREADS= <i>N</i> 其中 <i>N</i> 表示内核线程数。缺省值为 8。

描述	语法
AIXTHREAD_MUTEX_DEBUG 控制收集有关正在运行的线程进程中的互斥锁的调试信息的开关。缺省值为 OFF。维护调试信息对性能可能会有负面影响。	 ►►—AIXTHREAD_MUTEX_DEBUG=—  —►►
AIXTHREAD_COND_DEBUG 控制收集有关条件变量的调试信息的开关。缺省值为 OFF。维护调试信息对性能可能会有负面影响。	 ►►—AIXTHREAD_COND_DEBUG=—  —►►
AIXTHREAD_RWLOCK_DEBUG 导致 pthreads 库维护可由调试程序查看的所有读写锁定的列表。缺省值为 OFF。	 ►►—AIXTHREAD_RWLOCK_DEBUG=—  —►►
AIXTHREAD_GUARDPAGES 指定要创建的 4 KB 保护页数。保护页用于检测线程堆栈何时超出它的最大大小并防止错误的内存写操作。在进程堆上创建的线程堆栈可通过在堆栈顶部设置只读保护页来保护。写至这些页的任何尝试都会导致立即分段违例。研究堆栈溢出时的条件有助于调试程序以及确定适当的更正操作。设置并导出此变量。	 ►►—AIXTHREAD_GUARDPAGES= <i>N</i> —►► 其中 <i>N</i> 表示 4 KB 大小页的数目。缺省值为 0。 如果应用程序指定它自己的堆栈或将大量页用于它的进程堆，则不会创建保护页。
AIXTHREAD_SLPRATIO 设置系统的休眠率。该值告知系统应为休眠 pthread 预留多少内核线程。此调整参数允许更好地管理内核资源。缺省休眠比是 1:12。	 ►►—AIXTHREAD_SLPRATIO= <i>k</i> : <i>p</i> —►► 其中 <i>k</i> 表示内核线程数。缺省值为 1。 <i>p</i> 表示正在休眠的 pthread 数。缺省值为 12。 可对 <i>k</i> 和 <i>p</i> 指定任何正整数值。如果 <i>k</i> > <i>p</i> ，则将比率看作 1:1（即，指定的内核线程数不能多于 pthread 数）。
AIXTHREAD_STK 修改线程堆栈大小。使用此环境变量来指定最大 256MB 的堆栈大小，而不是调整 pthread 属性结构中的参数然后重新编译和重建应用程序。注意，增加线程堆栈的大小会减少可用于动态分配内存的空间，这是因为每个线程堆栈都是在线程堆上创建的。导出此环境变量以修改所创建 pthread 的堆栈大小而不必通过编程指定堆栈大小。	 ►►—AIXTHREAD_STK= <i>N</i> —►► 其中 <i>N</i> 表示字节数。缺省线程堆栈大小对于 32 位应用程序是 96 KB，对于 64 位应用程序是 192 KB。

与 GNU C 和 C++ 可移植性相关的功能

为了便于移植使用 GNU C 开发的应用程序或代码，XL C/C++ 支持 C99 和标准 C++ 的 GNU C 和 C++ 语言扩展的一部分。本节中的表列示了受支持的功能、不受支持的功能以及接受其语法但忽略其语义的功能。

要在 C 代码中使用受支持的扩展，请使用 **xlc** 或 **cc** 调用命令，或指定 **-qlanglvl=extc89**、**-qlanglvl=extc99** 或 **-qlanglvl=extended** 中的一个。要在 C++ 代码中使用这些功能，请指定 **-qlanglvl=extended** 选项。在 C++ 中，缺省情况下，接受所有受支持的 GNU C 和 C++ 功能。

在下列表中，标记为接受／忽略的扩展是编译器认为可以接受的编程关键字，但其 GNU C/C++ 语义不受支持。这意味着当编译器遇到接受／忽略的关键字或扩展时，编译不会停止，但在应用程序中不会实现其 GNU 语义。在严格的语言级别（stdc89 或 stdc99）下编译使用这些扩展的源代码将导致错误消息。

相关参考

在 GNU 手册（网址为 <http://gcc.gnu.org/onlinedocs>）中对 GNU C 和 C++ 语言扩展进行了完整的介绍。

函数属性

当声明或定义函数时使用关键字 `__attribute__` 来指定特殊属性。此关键字后跟用两个括号括住的属性说明。XL C/C++ 支持 GNU C 和 C++ 函数属性的一部分。描述为接受／忽略的行为意味着接受其语法，但是忽略其语义，编译继续进行。

GNU C/C++ 函数属性与 XL C/C++ 的兼容性

函数属性	行为
alias	支持
always_inline	支持
cdecl	接受／忽略
const	支持
constructor	接受／忽略
destructor	接受／忽略
dllexport	接受／忽略
dllimport	接受／忽略
eightbit_data	接受／忽略
exception	接受／忽略
format	支持
format_arg	支持
function_vector	接受／忽略
interrupt	接受／忽略
interrupt_handler	接受／忽略
longcall	接受／忽略
model	接受／忽略
no_check_memory_usage	接受／忽略
no_instrument_function	接受／忽略
noinline	支持
noreturn	支持
pure	支持
regparm	接受／忽略
section	接受／忽略
stdcall	接受／忽略
tiny_data	接受／忽略
weak	支持


相关参考

- *XL C/C++ Language Reference* 中的 “Function attributes”

变量属性

使用关键字 `__attribute__` 来指定变量或结构字段的特殊属性。此关键字后跟用两个括号括住的属性说明。XL C/C++ 支持 GNU C 和 C++ 变量属性的一部分。描述为接受 / 忽略的行为意味着接受其语法，但是忽略其语义，编译继续进行。

GNU C/C++ 变量属性与 XL C/C++ 的兼容性

变量属性	行为
<code>aligned</code>	支持
 <code>init_priority</code>	接受 / 忽略
<code>mode</code>	支持
<code>model</code>	接受 / 忽略
<code>nocommon</code>	接受 / 忽略
<code>packed</code>	支持
<code>section</code>	接受 / 忽略
<code>transparent_union</code>	接受 / 忽略
<code>unused</code>	接受 / 忽略
<code>weak</code>	接受 / 忽略


相关参考

- *XL C/C++ Language Reference* 中的 “Variable attributes”

类型属性

当定义 `struct` 和 `union` 类型时，使用关键字 `__attribute__` 来指定这些类型的特殊属性。此关键字后跟用两个括号括住的属性说明。XL C/C++ 支持 GNU C 和 C++ 类型属性的一部分。描述为接受 / 忽略的行为意味着接受其语法，但是忽略其语义，编译继续进行。

GNU C/C++ 类型属性与 XL C/C++ 的兼容性

类型属性	行为
<code>aligned</code>	支持
<code>packed</code>	支持
<code>transparent_union</code>	 支持
<code>unused</code>	接受 / 忽略

相关参考

- *XL C/C++ Language Reference* 中的 “Type attributes”



GNU C 和 C++ 断言

使用断言来测试已编译程序将在哪一种计算机或系统上运行。预定义了断言 `#cpu`、`#machine` 和 `#system`。也可以使用预处理伪指令 `#assert` 和 `#unassert` 来定义断言。

GNU C 断言	行为
<code>#assert</code>	支持
<code>#unassert</code>	支持
<code>#cpu</code>	支持
<code>#machine</code>	支持
<code>#system</code>	支持可能的值为 <code>aix</code> 和 <code>unix</code>

与 GNU C 和 C++ 相关的其它扩展

与 GNU C 和 C++ 相关的下列功能在扩展的语言级别（`extc89`、`extc99` 和 `extended`）下受支持。

- 使用伪指令 `#warning` 来使预处理器发出警告并继续处理。
- 使用伪指令 `#include_next` 来指定在当前头文件之后包含目录中的下一个头文件。
- 可以在每个词法块的开始处声明局部标号。
- 使用括号内用花括号括起的复合语句作为表达式。
- 使用 `__typeof__` 关键字表示表达式的类型。
- 使用复合表达式、条件表达式和强制类型转型作为左值。
- 使用已计算的 `goto` 语句来跳转至某一标号，该标号已获取其地址且该地址用作一个值。
- 使用关键字 `__alignof__` 来查询变量对齐或某种类型通常所需要的对齐。
- 使用以下关键字的备用拼写：`__asm__`、`__const__`、`__volatile__`、`__inline__`、`__signed__` 和 `__typeof__`。
- 当在严格的语言级别方式下使用扩展语言功能时，使用 `__extension__` 关键字来避免错误。
- 可以出现零长度的数组而不生成错误。
-  允许在另一个函数定义中出现的函数定义（嵌套函数）。
-  可以将联合成员强制类型转型为它所属的联合类型。

在扩展语言级别（`extc89`、`extc99` 和 `extended`）下，XL C/C++ 识别以下功能的语法，但不支持它们的语义。

-  寄存器变量的声明（全局或局部）可以建议首选寄存器。

附录 A. 语言支持

本附录讨论 C 和 C++ 编程语言的实现和 XL C/C++ 所提供的语言扩展。

与 ISO/IEC 国际标准的兼容性

XL C/C++ 可以鼓励强调可移植性的编程风格。语法和语义构成编程语言的完整规范，但特定语言规范的一致实现可能会由于语言扩展不同而不同。

严格地遵循它的语言规范的程序在不同环境中将具有最大可移植性。从理论上讲，在硬件差异所允许范围内，用一种符合标准的编译器可正确编译并且不使用任何扩展或实现定义的行为的程序将在所有其它符合标准的编译器中正确地进行编译和执行。正确利用语言实现提供的语言扩展的程序可以提高其目标代码的效率。

ISO/IEC 14882:2003(E) 国际标准兼容性


XL C/C++ 符合 “ISO/IEC 国际标准” 14882:2003(E)，它指定 C++ 编程语言编写的程序的格式并对其作出解释。该国际标准用于提高 C++ 程序在各种实现中的可移植性。ISO/IEC 14882:1998 是第一个 C++ 语言。

ISO/IEC 9899:1990 国际标准兼容性

“ISO/IEC 9899:1990 国际标准”（也称为 C89）指定用 C 编程语言编写的程序的格式并对其作出解释。设计此规范的目的是为了提升 C 程序在各种实现间的可移植性。此“标准”经历了 ISO/IEC 9899/COR1:1994、ISO/IEC 9899/AMD1:1995 和 ISO/IEC 9899/COR2:1996 这些版本，得到了改进和修正。要确保源代码严格遵循改进和修正后的 C89 标准，应指定 `-qlanglvl=stdc89` 编译器选项。

ISO/IEC 9899:1999 国际标准支持

“ISO/IEC 9899:1999 国际标准”（也称为 C99）是对用 C 编程语言编写的程序进行更新以后的标准。它用于增强 C 语言的能力，提供对 C89 的澄清并加入技术更正。XL C/C++ 支持此语言规范的许多功能。

 C 编译器支持在 “C99 标准” 中指定的所有语言功能。要确保源代码遵循此语言功能集，应使用 `c99` 调用命令。注意，该 “标准” 还指定运行时库中的功能。这些功能在当前运行时库和操作环境中可能不受支持。系统头文件是否存在说明了这种支持是否存在。

C99 中的主要功能

XL C/C++ 实现所有 C99 语言功能。以下是选择的主要功能的表。参考表示 *XL C/C++ Language Reference* 中的章节。

对 IBM C 的 ISO/IEC 9899:1999 国际标准扩展

C99 功能	相关参考
指针的 restrict 类型限定符	The restrict Type Qualifier
通用字符名称	The Unicode Standard

对 IBM C 的 ISO/IEC 9899:1999 国际标准扩展

C99 功能	相关参考
预定义标识 <code>__func__</code>	Predefined Identifiers
带有可变数目的自变量和空自变量的拟函数宏	Function-Like Macros
<code>_Pragma</code> 一元运算符	The <code>_Pragma</code> Operator
可变长度数组	Variable Length Arrays
数组下标声明中的 <code>static</code> 关键字	Arrays
<code>complex</code> 数据类型	Complex Types
<code>long long int</code> 和 <code>unsigned long long int</code> 类型	Integer Variables
十六进制浮点常数	Hexadecimal Floating Constants
聚集类型的复合文字	Compound Literals
指定的初始化函数	Initializers
C++ 样式注释	Comments
不允许隐式函数声明	Function Declarations
混合声明和代码	The <code>for</code> Statement
<code>_Bool</code> 类型	Simple Type Specifiers
<code>inline</code> 函数声明	Inline Functions
聚集的初始化函数	Initializing Arrays Using Designated Initializers

C99 中支持的 C89 的更改和说明

C99 标准中的一些规范基于 C89 标准的更改和说明，而不是基于语言的新功能。XL C/C++ 支持所有 C99 语言功能，包括以下功能：

- 允许灵活的数组成员。具有两个或更多成员的结构最后一个成员可以不声明大小。
- 不支持声明隐式 `int`。所有声明都必须具有类型说明符。
- 在枚举说明符中允许尾随逗号。
- 除非另外显式指定，否则接受（忽略）重复的类型限定符。
- 如果 `return` 语句中缺少所需要的表达式，则发出诊断消息。
- 在预处理期间求值的常量表达式现在使用 `long long` 和 `unsigned long long` 数据类型。
- 在拟函数宏中允许空的宏自变量。
- `#line` 的最大值已增加到 2 147 483 647。

XL C/C++ 中的 C99 功能

还在 C++ 中实现了“ISO/IEC 9899:1999 国际标准”（C99）的某些功能。在指定了 `-qlanglvl=extended` 编译器选项的情况下，这些扩展可用。

IBM C++ 的 ISO/IEC 9899:1999 国际标准扩展

C99 功能	参考
指针的 <code>restrict</code> 类型限定符	The <code>restrict</code> Type Qualifier
通用字符名称	The Unicode Standard
预定义标识 <code>__func__</code>	Predefined Identifiers

C99 功能	参考
可变长度数组	Variable Length Arrays
complex 数据类型	Complex Types
十六进制浮点常数	Hexadecimal Floating Constants
聚集类型的复合文字	Compound Literals
带有可变数目的自变量和空自变量的拟函数宏	Function-Like Macros
_Pragma 一元运算符	The _Pragma Operator

增强的语言级别支持

-qlanglvl 编译器选项用于指定受支持的语言级别，因此会影响编译代码的方式。也可以通过使用不同编译器调用命令隐式地指定语言级别。一般情况下，在标准语言级别下正确编译和运行的有效程序在启用垂直扩展后应能继续正确编译和运行并产生相同的结果。

例如，要编译 C 程序以便它们严格符合“ISO/IEC 9899:1990 国际标准”（C89），需要指定 -qlanglvl=stdc89。stdc89 子选项指示编译器严格遵循该标准，不允许任何语言扩展。（c89 编译器调用命令隐式地指定此语言级别。）

也可以使用标准语言级别的扩展。不影响标准功能的扩展称为垂直扩展。例如，当编译 C 程序时，可以通过指定 -qlanglvl=extc89 启用 C89 的垂直扩展。






“ISO/IEC 9899:1999 国际标准”（C99）中描述的大多数语言功能都被认为是 C89 的垂直扩展。

当编译 C++ 程序时，可以通过指定 -qlanglvl=extended 启用垂直扩展。

另一方面，非垂直扩展会影响在某个国际标准中所描述的语言的某些方面或与它们发生冲突。必须通过特定编译器选项显式地启用接受这些扩展。对非垂直扩展的依赖使得将应用程序移植到不同环境更难。

以下列示了 -qlanglvl 选项的主要子选项。

选择的 -qlanglvl 子选项

-qlanglvl 子选项	子选项描述
-qlanglvl=stdc99	 指定严格遵循 C99 标准。
-qlanglvl=stdc89	 指定严格遵循 C89 标准。
-qlanglvl=ansi	 指定严格遵循 C89 标准并启用 -qlonglong 编译器选项。
-qlanglvl=extc99	 启用 C99 的所有垂直扩展。
-qlanglvl=extc89	 启用 C89 的所有垂直扩展。

选择的 `-qlanglvl` 子选项

<code>-qlanglvl</code> 子选项	子选项描述
<code>-qlanglvl=extended</code>	<div data-bbox="776 262 873 294"> C</div> 启用 C89 的所有垂直扩展并指定 <code>-qpconv</code> 编译器选项。 <div data-bbox="776 367 873 399"> C++</div> 启用“标准 C++”的所有垂直扩展。

附录 B. OpenMP 一致性和支持

OpenMP 应用程序编程接口 (API) 是可移植且可伸缩的编程模型, 它为用 C、C++ 和 Fortran 开发多平台的共享内存并行应用程序提供标准接口。该规范由 OpenMP 组织定义, 该组织由主要的计算机硬件和软件供应商组成, 其中包括 IBM。

XL C/C++ 符合 “OpenMP 规范 2.0”。该编译器能识别并保持下列 OpenMP V2.0 元素的语义:

- `#pragma omp` 伪指令中多个子句的逗号定界符。
- `num_threads` 子句。
- `copyprivate` 子句。
- `threadprivate` 静态块作用域变量。
- 支持 C99 可变长度数组。
- 私有变量的冗余声明。
- 计时例程 `omp_get_wtick` 和 `omp_get_wtime`。

下面所描述的伪指令、库函数和环境变量可用于创建和管理并行程序, 同时还保持可移植性。

要启用 OpenMP 并行处理, 必须指定 **-qsmp** 编译器选项。

- 要选择自动并行性, 可指定 **-qsmp** 或 **-qsmp=auto**。除了能识别和实现程序中包含的任何 OpenMP 伪指令、库函数和环境变量外, 此子选项还使编译器能够执行隐式并行性。
- 要选择严格遵循 “OpenMP 规范 2.0”, 可指定 **-qsmp=omp**。此子选项确保编译器仅实现在代码中指定的 OpenMP 伪指令、库函数和环境变量。它不执行任何其它的自动并行处理。

相关参考

- <http://www.openmp.org>
- *XL C/C++ Compiler Reference* 中的 “Pragmas to control parallel processing”
- *XL C/C++ Compiler Reference* 中的 “Program parallelization”

OpenMP 伪指令


每个伪指令以 `#pragma omp` 开始, 以减少与其它编译指示伪指令冲突的可能。

XL C/C++ 中的 OpenMP 伪指令

伪指令名称	伪指令描述
<code>parallel</code>	<code>parallel</code> 伪指令定义并行区域, 该区域是程序的要由多个线程并行执行的一个区域。
<code>for</code>	<code>for</code> 伪指令标识一个迭代的共享工作构造, 该构造指定了一个区域, 在这个区域中关联循环的迭代将并行执行。 <code>for</code> 循环的迭代分布在已经存在的线程上。
<code>sections</code>	<code>sections</code> 伪指令标识非迭代的共享工作构造, 该构造指定了一组构造, 要将它们分配给一个队的多个线程。每一个 <code>section</code> 由队中的一个线程执行一次。

XL C/C++ 中的 OpenMP 伪指令

伪指令名称	伪指令描述
single	single 伪指令标识一种构造，该构造指定关联的结构化块仅由队中的一个线程（不必是主线程）执行。
parallel for	parallel for 伪指令是包含单个 for 伪指令的并行区域的快捷格式。其语义等同于显式指定直接后跟 for 伪指令的 parallel 伪指令。
parallel sections	parallel sections 伪指令提供用于指定包含单个 sections 伪指令的并行区域的快捷格式。其语义等同于显式指定直接后跟 sections 伪指令的 parallel 伪指令。
master	master 伪指令标识一种构造，该构造指定由队中主线程执行的结构化块。
critical	critical 伪指令标识一种构造，该构造限制每次只能有单个线程来执行相关的结构化块。可选的 name 可以用来标识关键区域。线程在关键区域的开始处等待，直到没有其它线程具有相同名称执行关键区域为止。所有未命名的 critical 伪指令都映射到同一未指定的名称。
barrier	barrier 伪指令使一个队中的所有线程同步。当遇到该伪指令时，每个线程将等待，直到所有其它线程都达到此点为止。在所有线程都遇到 barrier 后，每个线程开始以并行方式执行 barrier 伪指令之后的语句。
atomic	atomic 伪指令标识特定的内存位置，该位置必须自动更新且未暴露于多个同时执行写操作的线程。
flush	flush 伪指令标志了一个点，在该点编译器确保并行区域中的所有线程都具有内存中指定目标的相同视图。
ordered	ordered 伪指令标识必须按顺序执行的结构化代码块。
threadprivate	threadprivate 伪指令声明文件作用域、名称空间作用域或静态块作用域变量是线程私有的。

 C 编译器识别下列用于程序并行性的其它伪指令。它们不是“OpenMP 规范 1.0”或“OpenMP 规范 2.0”的一部分，而且 C++ 编译器不能识别它们。

- #pragma ibm critical
- #pragma ibm independent_calls
- #pragma ibm independent_loop
- #pragma ibm iterations
- #pragma ibm parallel_loop
- #pragma ibm permutation
- #pragma ibm schedule
- #pragma ibm sequential_loop

OpenMP 数据作用域属性子句

可以在伪指令中指定子句来控制在并行或工作共享构造的持续时间内变量的作用域属性。

XL C/C++ 中的 OpenMP 数据作用域属性子句

数据作用域属性子句名称	数据作用域属性子句描述
private	private 子句声明列表中的变量是队中每个线程私有的。
firstprivate	firstprivate 子句提供 private 子句提供的功能的超集。
lastprivate	lastprivate 子句提供 private 子句提供的功能的超集。
copyprivate	copyprivate 子句提供另外一种方法来替代使用共享变量将值广播到一个组这种方法。该机制使用私有变量来将值从小组的一个成员广播到另一个成员。
num_threads	num_threads 子句提供了一种为并行构造请求特定数目线程的能力。
shared	shared 子句在队中所有线程之间共享在列表中出现的变量。队中的所有线程访问 shared 变量的同一存储区域。
reduction	reduction 子句使用指定的运算符对出现在列表中的标量变量执行归约。
default	default 子句允许用户影响变量的数据作用域属性。

OpenMP 库函数

OpenMP 运行时库函数包括在头文件 <omp.h> 中。它们包括执行环境函数和锁定函数，前者可以用来控制和查询并行执行环境，后者可以用来使数据访问同步。

XL C/C++ 中的 OpenMP 运行时库函数

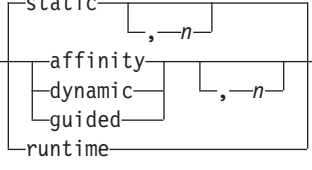

运行时库函数名称	运行时库函数描述
omp_set_num_threads	设置要用于后续并行区域的线程数。
omp_get_num_threads	返回当前在队中执行并行区域的线程数，这些线程是从该并行区域调用的。
omp_get_max_threads	返回 omp_get_num_threads 调用可以返回的最大值。
omp_get_thread_num	返回队中正在执行函数的线程的线程号。队中的主线程是线程 0。
omp_get_num_procs	返回可以指定给程序的最大处理器数。
omp_in_parallel	如果在并行执行的并行区域的动态范围内调用它，返回非零；否则它返回 0。
omp_set_dynamic	启用或禁用可用于执行并行区域的线程数的动态调整。
omp_get_dynamic	如果启用动态线程调整，则返回非零，否则返回 0。
omp_set_nested	启用或禁用嵌套并行性。
omp_get_nested	如果启用嵌套并行性，则返回非零，而如果禁用它，则返回 0。
omp_init_lock	初始化简单锁定。

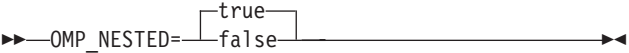
XL C/C++ 中的 OpenMP 运行时库函数

运行时库函数名称	运行时库函数描述
omp_destroy_lock	除去简单锁定。
omp_set_lock	等到简单锁定可用为止。
omp_unset_lock	释放简单锁定。
omp_test_lock	测试简单锁定。
omp_init_nest_lock	初始化可嵌套的锁定。
omp_destroy_nest_lock	除去可嵌套的锁定。
omp_set_nest_lock	等到可嵌套的锁定可用为止。
omp_unset_nest_lock	释放可嵌套的锁定。
omp_test_nest_lock	测试可嵌套的锁定。
omp_get_wtick	返回连续时钟信号之间的秒数。
omp_get_wtime	返回经过的挂钟时间（以秒计）。

OpenMP 环境变量

OpenMP 环境变量控制并行代码的执行。环境变量的名称必须始终为大写，尽管它们的值不区分大小写。

描述	语法
<p>OMP_SCHEDULE</p> <p>设置运行时调度和块大小。仅适用于将调度类型设置为 runtime 的 OpenMP 伪指令。</p>	<p>▶▶—OMP_SCHEDULE=</p>  <p>其中</p> <p>affinity</p> <p>一个仅对 C 有效的 IBM 扩展。指定循环的迭代最初分为本地分区，大小等于迭代数顶值除以线程数：$\text{CEILING}(\text{number_of_iterations} \div \text{number_of_threads})$。每个本地分区进一步细分为块，大小等于本地分区中余下的迭代数一半的顶值的块：$\text{CEILING}(\text{iterations_left_in_local_partition} \div 2)$。当线程变空闲时，它从它的本地分区中取出下一块。如果本地分区中没有块，则线程从另一线程的分区中取可用的块。如果指定了 n，则每个本地分区细分为大小为 n 的块。如果未指定 n，则缺省值是 1。</p> <p>dynamic</p> <p>指定 for 循环的迭代应划分为大小为 n 的一系列块以及根据后面的进程处理块。对等待赋值的线程指定一组迭代，线程执行这些迭代然后等待下一次赋值。一直重复此过程，直到指定了所有块为止。如果未指定 n，则缺省块大小是 1。</p> <p>guided</p> <p>指定应将 for 循环的迭代指定给块中大小不断减小的块中的线程以及根据后面的进程处理块。将完成了指定迭代块的线程动态指定给另一块，直到指定了所有的块为止。如果未指定 n，则初始块大小的缺省值是 1。</p> <p>static</p> <p>指定 for 循环的迭代应划分为大小为 n 的一系列块以及根据后面的进程处理块。按由线程号确定的顺序将可用的线程指定给块。当未指定 n 时，迭代空间划分为大小大致相等的块，对每个线程指定一个块。</p> <p>n 是一个正数，表示块大小。</p>
<p>OMP_DYNAMIC</p> <p>启用或禁用可用于执行并行区域的线程数的动态调整。</p>	<p>▶▶—OMP_DYNAMIC=</p>  <p>其中</p> <p>true</p> <p>启用可用线程数的动态调整。</p> <p>false</p> <p>禁用可用线程数的动态调整。</p>
<p>OMP_NUM_THREADS</p> <p>可用于执行的线程数的设置。</p>	<p>▶▶—OMP_NUM_THREADS=n—</p> <p>其中</p> <p>n 表示线程数。</p>

描述	语法
OMP_NESTED 启用或禁用嵌套并行性。	 其中 true 启用嵌套并行性。 false 禁用嵌套并行性。

OpenMP 实现定义的行为

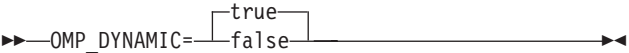

在标准中未指定下列信息。标准的每个实现可以有其自己的实现定义的值。

条件编译	<code>_OPENMP</code> 宏定义为 199810。
调度	<code>schedule</code> 子句指定在队的线程之间如何划分 for 循环的迭代。可能的 OpenMP 标准值是 <code>static</code> 、 <code>dynamic</code> 、 <code>guided</code> 和 <code>runtime</code> 。此外，IBM C 添加了值 <code>affinity</code> 作为扩展。当没有显式定义的 <code>schedule</code> 子句时，XL C/C++ 的缺省调度是 <code>static</code> 。

调整 OpenMP 程序

除了调整 `pthread` 生命周期中的各种定时的环境变量之外，下列环境变量有助于调整 OpenMP 应用程序。

选择的用于调整 OpenMP 应用程序的环境变量

描述	语法
OMP_DYNAMIC 启用或禁用可用于执行并行区域的线程数的动态调整。当启用了该变量（缺省设置）时，运行时环境可调整它用于执行并行区域的线程数以便它使用系统资源的效率最高。在以下三种情况下，应禁用该变量：基准测试，缩放测试，以及应用程序依赖于特定线程数目的情况（因为动态检查会增加少量的开销）。	 其中 true 启用可用线程数的动态调整。 false 禁用可用线程数的动态调整。
MALLOCMULTIHEAP 多个堆很有用，有了多个堆，线程应用程序就可以让多个线程发出内存分配子例程调用。对于单个堆，将对尝试进行 <code>malloc()</code> 、 <code>free()</code> 或 <code>realloc()</code> 调用的所有例程序列化（即，一次只能有一个线程可以调用这些函数中的任何一个）。这种情况对于多处理器上的机器有严重的影响。对于多个堆，每个线程都可获得它自己的堆，最多可达 32 个独立的堆。通过导出此环境变量来启用对多个堆的使用（不是缺省设置的）。	

选择的用于调整 OpenMP 应用程序的环境变量

描述	语法
XLSMPOPTS 对于 32 位 OpenMP 应用程序，对每个线程的堆栈大小的缺省限制太小了，如果超出此限制，就会产生运行时错误。如果出错，可通过用堆栈子选项设置 XLSMPOPTS 环境变量来增大堆栈大小。	►—export—XLSMPOPTS=stack=<i>n</i>—◄◄ 其中 <i>n</i> 表示堆栈大小（以字节计）。所有线程的总堆栈大小不能超过 256 MB（一个内存段）。一个内存段限制不适用于 64 位的应用程序。缺省值是每个线程 4 MB。

相关参考

- 第 51 页的『调整多线程程序』

声明

本信息是为在美国提供的产品和服务编写的。

IBM 可能在其他国家或地区不提供本文档中讨论的产品、服务或功能特性。有关您当前所在区域的产品和服务的信息，请向您当地的 IBM 代表咨询。任何对 IBM 产品、程序或服务的引用并非意在明示或暗示只能使用 IBM 的产品、程序或服务。只要不侵犯 IBM 的知识产权，任何同等功能的产品、程序或服务，都可以代替 IBM 产品、程序或服务。但是，评估和验证任何非 IBM 产品、程序或服务，则由用户自行负责。

IBM 公司可能已拥有或正在申请与本文档内容有关的各项专利。提供本文档并未授予用户使用这些专利的任何许可。您可以用书面方式将许可查询寄往：

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

有关双字节（DBCS）信息的许可查询，请与您所在国家或地区的 IBM 知识产权部门联系，或用书面方式将查询寄往：

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

本条款不适用英国或任何这样的条款与当地法律不一致的国家或地区：
INTERNATIONAL BUSINESS MACHINES CORPORATION “按现状”提供本出版物，不附有任何种类的（无论是明示的还是暗含的）保证，包括但不限于暗含的有关非侵权、适销和适用于某种特定用途的保证。某些国家或地区在某些交易中不允许免除明示或暗含的保证，因此本条款可能不适用于您。

本信息中可能包含有技术方面不够准确的地方或印刷错误。此处的信息将定期更改；这些更改将编入本资料的新版本中。IBM 可以随时对本资料中描述的产品和 / 或程序进行改进和 / 或更改，而不另行通知。

本信息中对非 IBM Web 站点的任何引用都只是为了方便起见才提供的，不以任何方式充当对那些 Web 站点的保证。那些 Web 站点中的资料不是 IBM 产品资料的一部分，使用那些 Web 站点带来的风险将由您自行承担。

IBM 可以按它认为适当的任何方式使用或分发您所提供的任何信息而无须对您承担任何责任。

本程序的被许可方如果要了解有关程序的信息以达到如下目的：（i）允许在独立创建的程序和其他程序（包括本程序）之间进行信息交换，以及（ii）允许对已经交换的信息进行相互使用，请与下列地址联系：

Lab Director
IBM Canada Ltd. Laboratory

B3/KB7/8200/MKM
8200 Warden Avenue
Markham, Ontario, Canada L6G 1C7

只要遵守适当的条件和条款，包括某些情形下的一定数量的付费，都可获得这方面的信息。

本资料中描述的许可程序及其所有可用的许可资料均由 IBM 依据 IBM 客户协议、IBM 国际软件许可协议或任何同等协议中的条款提供。

涉及非 IBM 产品的信息可从这些产品的供应商、其出版说明或其他可公开获得的资料中获取。IBM 没有对这些产品进行测试，也无法确认其性能的精确性、兼容性或任何其他关于非 IBM 产品的声明。有关非 IBM 产品性能的问题应当向这些产品的供应商提出。

本信息包含在日常业务操作中使用的数据和报告的示例。为了尽可能完整地说明这些示例，示例中可能会包括个人、公司、品牌和产品的名称。所有这些名称都是虚构的，与实际商业企业所用的名称和地址的任何雷同纯属巧合。

版权许可：

本信息包括源语言形式的样本应用程序，这些样本说明不同操作平台上的编程方法。如果是为按照在编写样本程序的操作平台上的应用程序编程接口（API）进行应用程序的开发、使用、经销或分发为目的，您可以任何形式对这些样本程序进行复制、修改、分发，而无须向 IBM 付费。这些示例并未在所有条件下作全面测试。因此，IBM 不能担保或暗示这些程序的可靠性、可维护性或功能。用户如果是为了按照 IBM 应用程序编程接口开发、使用、经销或分发应用程序，则可以任何形式复制、修改和分发这些样本程序，而无须向 IBM 付费。

凡这些样本程序的每份拷贝或其任何部分或任何衍生产品，都必须包括如下版权声明：

©（贵公司的名称）（年）。此部分代码是根据 IBM 公司的样本程序衍生出来的。© Copyright IBM Corp. 1998, 2004. All rights reserved.

如果您正在以软拷贝格式查看本信息，图片和彩色图例可能无法显示。

编程接口信息

编程接口信息用来帮助您使用此程序来创建应用软件。

通用编程接口允许客户编写获取此程序工具的服务的应用软件。

但是，此信息也可能包含诊断、修改和调整信息。这些诊断、修改和调整信息用于帮助您调试应用软件。

警告： 不要将此诊断、修改和调整信息用作编程接口，因为它是会更改的。

商标和服务标记

下列各项是国际商业机器公司在美国和 / 或其他国家或地区的商标:

AIX	POWER3	pSeries
@server	POWER4	Redbooks
IBM	POWER5	RS/6000
	PowerPC	VisualAge
	PowerPC Architecture	

UNIX 是 The Open Group 在美国和其他国家或地区的注册商标。

Linux 是 Linus Torvalds 在美国和 / 或其他国家或地区的商标。

其他公司、产品和服务名称可能是其他公司的商标或服务标记。

业界标准

支持下列标准:

- C 语言符合 “国际标准 C (ANSI/ISO-IEC 9899–1990 [1992]) ”。此标准已正式替换 “美国信息系统编程语言 C 国家标准 (X3.159–1989) ”，它在技术上等价于 ANSI C 标准。编译器支持 ISO/IEC 9899:1990/Amendment 1:1994 对 C 标准所进行的更改。
- C 语言符合 “信息系统编程语言 C 国际标准 (ISO/IEC 9899–1999 (E)) ”。
- C++ 语言符合 “信息系统编程语言 C++ 国际标准 (ISO/IEC 14882:1998) ” (该语言的第一个正式定义)。
- C++ 语言还符合 “信息系统编程语言 C++ 国际标准 (ISO/IEC 14882:2002 (E)) ”，当前也称为 *Standard C++*。
- C 和 C++ 编译器支持 OpenMP C 和 C++ 应用程序编程接口版本 2.0。