XL C/C++ Advanced Edition

IBM XL C/C++ Advanced Edition V7.0 for Linux スタートアップ・ガイド

バージョン 7.0

SC88-9977-01
(英文原典：SC09-7944-01)

XL C/C++ Advanced Edition

IBM XL C/C++ Advanced Edition V7.0 for Linux スタートアップ・ガイド

バージョン 7.0

SC88-9977-01
(英文原典：SC09-7944-01)

ご注意

本書および本書で紹介する製品をご使用になる前に、 69 ページの『特記事項』に記載されている情報をお読みください。

本書は、IBM XL C/C++ Advanced Edition のバージョン 7.0.1 (プロダクト番号 5724-K77)、および新しい版で特に明記されていない限り、以降のすべてのリリースおよびモディフィケーションに適用されます。

本マニュアルに関するご意見やご感想は、次の URL からお送りください。今後の参考にさせていただきます。

http://www.ibm.com/jp/manuals/main/mail.html

なお、日本 IBM 発行のマニュアルはインターネット経由でもご購入いただけます。詳しくは

http://www.ibm.com/jp/manuals/ の「ご注文について」をご覧ください。

(URL は、変更になる場合があります)

お客様の環境によっては、資料中の円記号がバックスラッシュと表示されたり、バックスラッシュが円記号と表示されたりする場合があります。

原　典： SC09-7944-01
XL C/C++ Advanced Edition
Getting Started with IBM XL C/C++
Advanced Edition V7.0 for Linux
Version 7.0

発　行： 日本アイ・ビー・エム株式会社

担　当： ナショナル・ランゲージ・サポート

第1刷 2005.1

この文書では、平成明朝体™W3、平成明朝体™W7、平成明朝体™W9、平成角ゴシック体™W3、平成角ゴシック体™W5、および平成角ゴシック体™ W7を使用しています。この (書体*)は、(財) 日本規格協会と使用契約を締結し使用しているものです。フォントとして無断複製することは禁止されています。

注* 平成明朝体™W3、平成明朝体™W7、平成明朝体™W9、平成角ゴシック体™W3、
平成角ゴシック体™W5、平成角ゴシック体™W7

目次

本書について

XL C/C++ Advanced Edition は、PowerPC® アーキテクチャーで実行される Linux オペレーティング・システム用の最適化標準コマンド行コンパイラーです。このコンパイラーは、拡張 C および C++ プログラム言語で 32 ビットおよび 64 ビット・アプリケーションを作成および保守するためのプロフェッショナル・プログラミング・ツールです。

本書 では XL C/C++ コンパイラーについて紹介します。説明する内容は、さまざまなコンパイラー呼び出しと、コンパイル環境をカスタマイズしてコンパイル・プロセスを制御する方法についてです。本書では、コンパイラーが実行できる変換のタイプ、入出力可能なファイル・タイプ、コンパイラー・オプションのカテゴリー別の要約、および既存アプリケーションの移植時の考慮事項も紹介します。また、本書では、アプリケーションのパフォーマンスを最適化する方法も簡単に紹介します。コンパイラーの機能を最適化すると、PowerPC プロセッサーの多層アーキテクチャーを活用できます。

Linux および AIX® は、補完的なオペレーティング・システムです。IBM C for AIX または VisualAge® C++ Professional for AIX の知識を持っている方は、その makefile を Linux プラットフォームで機能するように簡単に適応させることができます。IBM C および C++ コンパイラーの初心者の方は、本書でコンパイル、リンク、および実行時にパフォーマンスを向上させる方法を学習することができます。

本書は、読者が C および C++ プログラム言語、Linux オペレーティング・システム、および bash シェルについての知識をもっていることを前提としています。

強調表示の規則

太字	コマンド、キーワード、およびシステムによって名前が事前定義されているその他の項目を識別します。
イタリック	その実際の名前または値がプログラマーによって提供されるパラメーターを識別します。*イタリック* は、新規用語を最初に言及する際にも使用されます。
例	特定のデータ値の例、ユーザーに表示されるテキストに類似したテキストの例、プログラム・コードの部分、システムからのメッセージ、またはユーザーが実際に入力するべき情報の例などを識別します。

これらの例は、言語の使用方法を説明するもので、実行時間の最小化、ストレージの節約、エラーのチェックを行うためのものではありません。これらの例では、言語構成の使用についてのすべては説明しません。例の中には、コードの一部分だけを示し、コードを追加しないとコンパイルできないものもあります。

構文図の読み方

- 構文図は、左から右、上から下に、線のパスに従って読んでください。

►►— は、コマンド、ディレクティブ、またはステートメントの先頭を示します。

—► は、コマンド、ディレクティブ、またはステートメント構文が、次の行に続いていることを示します。

►— は、コマンド、ディレクティブ、またはステートメントが、前の行から続いていることを示します。

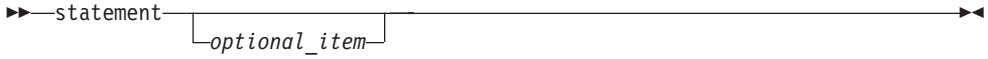
—►◄ は、コマンド、ディレクティブ、またはステートメントの終わりを示します。

完全なコマンド、ディレクティブ、またはステートメント以外の構文単位の図は、 ►— 記号で始まり、—► 記号で終わります。

注: 次の図で、statement は、C または C++ コマンド、ディレクティブ、またはステートメントを表しています。

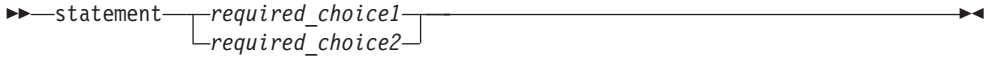
- 必須項目は、水平線 (メインパス) 上に記述されます。

- オプション項目は、メインパスの下に記述されます。



- 2 つ以上の項目から選択可能な場合は、スタック内に垂直に記述されます。

  いずれか 1 つの項目の選択が必須 の場合は、スタック内の項目のいずれか 1 つがメインパス上に記述されます。



  いずれか 1 つの項目の選択がオプションの場合は、スタック全体がメインパスの下に記述されます。

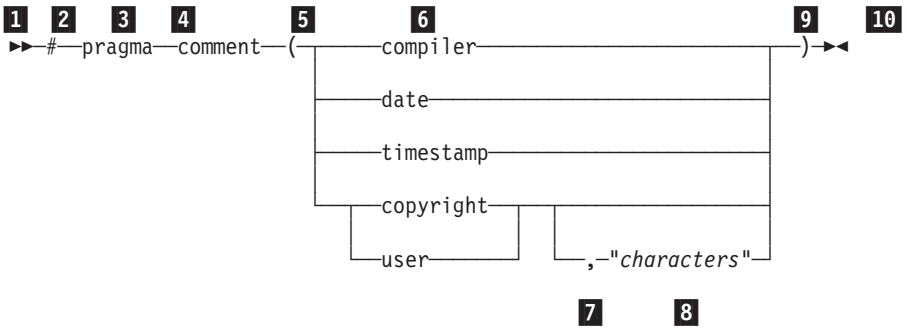
  デフォルト項目は、メインパスの上に記述されます。

- メインパスの線の上の左に戻る矢印は、繰り返し可能な項目を示します。

  スタックの上の繰り返し矢印は、スタック内の項目から複数の項目を選択するか、1 つの項目を繰り返し選択できることを示しています。
- キーワードは、非イタリック体で記述されています。示されているとおりに正確に入力する必要があります (例えば extern)。

変数は、イタリック体の小文字で記述されます (例えば、*identifier*)。変数は、ユーザー提供の名前または値を表します。
- 構文図に、句読記号、小括弧、算術演算子、または、ほかの同様な記号が示されている場合は、構文の一部としてこれらの文字を入力する必要があります。

次の構文図の例では、**#pragma comment** ディレクティブの構文を示しています。**#pragma** ディレクティブの詳細については、「*XL C/C++ ランゲージ・リファレンス*」を参照してください。



1 構文図の始まりを示します。

2 記号 # を最初に記述します。

3 キーワード pragma は、記号 # の次に記述されます。

4 プラグマの名前 comment は、キーワード pragma の次に記述します。

5 左括弧が必要です。

6 コメントの型を、表示されている compiler、date、timestamp、copyright、または user のうちいずれか 1 つだけ入力します。

7 コンマが、コメントの型 copyright または user とオプションの文字ストリングの間に必要です。

8 文字ストリングをコンマの次に記述します。文字ストリングは、二重引用符で囲みます。

9 右小括弧は必須です。

10 これが、構文図の終わりを示します。

次の **#pragma comment** ディレクティブの例は、上記のダイアグラムに従っており、構文上正しい例です。

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

XL C/C++ 概要

XL C/C++ Advanced Edition は、 PowerPC アーキテクチャーを持つ PowerPC ハードウェアで実行される Linux オペレーティング・システム用の、最適化を行う、標準ベースのコマンド行コンパイラーです。アプリケーション開発者は、このコンパイラーを使用することにより、最適化された 32 ビット・アプリケーションおよび 64 ビット・アプリケーションを作成および保守することができます。これらのアプリケーションでは、自動または明示の共用メモリー・プログラム並列化を使用して、パフォーマンスを改善することができます。 PowerPC アーキテクチャーによって可能になる最適化およびパフォーマンスの改善の多くは、コンパイラー・オプション、プラグマ、およびコンパイラー起動モードによって制御されます。そのため、特殊な手作業によるコーディングの量を最小限にして、このハードウェア・アーキテクチャーの利点を実現することができます。

この製品は、IBM VisualAge C++ for Linux バージョン 6.0 の後継リリースです。IBM は *VisualAge C++* を *XL C/C++* に商標変更しました。

共用メモリー・プログラム並列化

プログラム・パフォーマンスは、PowerPC アーキテクチャーで可能な共用メモリー・プログラム並列化 (SMP) を活用することにより、かなり改善されます。コンパイラーでは、SMP のインプリメントのために以下の方式が用意されています。

- プログラム・ループの自動並列化 (暗黙の並列化)
- OpenMP 準拠のプラグマ・ディレクティブの使用による C および C++ プログラム・コードの明示的並列化

OpenMP ディレクティブは、反復的および非反復的コードの平行領域の定義に関係します。

ソース互換性

このコンパイラーは、C のために次の 2 つの ISO プログラム言語仕様をサポートしています: ISO/IEC 9899:1990 (C89 として参照) および ISO/IEC 9899:1999 (C99)。またこのコンパイラーは次の両方の C++ 標準もサポートしています: ISO/IEC 14882:2003 (*Standard C++* として参照)、およびこの言語の最初の公式仕様である、ISO/IEC 14882:1998 (C++98 として参照)。標準化された言語レベルのほかに、このコンパイラーは、GNU C および C++ 言語拡張機能のサブセットを含む、数多くの言語拡張機能もサポートしています。

バイナリー互換性

XL C/C++ は、C++ Abstract Binary Interface (ABI) をサポートします。コンパイラーは、GNU C/C++ バージョン 3.2 または 3.3 コンパイラーで作成されたものと互換性のある、バイナリー・ファイルまたはオブジェクト・ファイルを作成します。また、このコンパイラーは OpenMP バージョン 2.0 標準もサポートしています。OpenMP 準拠の C および C++ オブジェクト・ファイルは、IBM XL Fortran for Linux で作成されたものとバイナリー互換であるため、C または C++ と Fortran の間での言語間呼び出しが可能です。

オープン・ソース・リソースとの共存

GNU C または C++ コンパイラーでコンパイルされたコードとのバイナリー互換性を獲得するため、 XL C/C++ でコンパイルされたプログラムには、同じシステムに常駐する GNU コンパイラーで使用されるものと同じヘッダーが組み込まれます。コンパイラーは、GNU C で作成されたオブジェクトとのバイナリー互換性を維持しながら、プログラムを最適化します。この関係について、注意すべき点は以下のとおりです。

- PowerPC プロセッサー用の IBM 組み込み関数は、GNU C 組み込み関数と共存する。
- コンパイルでは Linux 配布版に含まれる GNU C および C++ ヘッダー・ファイルが使用される。
- コンパイルでは GNU アセンブラーが使用される。
- リンクでは GNU リンカーが使用される。
- コンパイル済みプログラムでは GNU C および C++ ランタイム・ライブラリーが使用される。
- デバッグでは GNU デバッガー gdb が使用される。

コマンド行 C および C++ コンパイラー

XL C/C++ では基本コンパイラー呼び出しコマンドを選択でき、さまざまなバージョン・レベルの C および C++ 言語をサポートしています。各呼び出しコマンドは、言語レベルのコンパイラー・サブオプション、その他の関連言語フィーチャーのオプション、および関連する定義済みマクロを自動的に設定します。多くの場合、C ソース・ファイルをコンパイルするときは **xlc** コマンドを使用し、C++ ソース・ファイルをコンパイルするとき、または C と C++ の両方のソース・ファイルが存在するときは **xlC** コマンドを使用します。 **xlc++** 呼び出しは **xlC** 呼び出しと同等です。

さまざまな基本コマンドがあり、特殊環境およびファイル・システムの要件をサポートしています。コマンドの各バリエーションは、基本コマンドにサフィックスを付け足すことで形成されます。 Linux プラットフォームでは、スレッド・セーフ・アプリケーションをコンパイルするために、*_r* サフィックスが提供されています。これらのバリエーションは「*再入可能コンパイラー呼び出し*」とも呼ばれます。

また、gxlc および gxlc++ ユーティリティーは特殊なコンパイラー呼び出しです。

ライブラリー

XL C/C++ は GNU C および C++ ヘッダーを使用し、その結果得られるアプリケーションは、オペレーティング・システムとともにロードされた gcc レベルで提供される C および C++ ランタイム・ライブラリーとリンクされます。 XL C/C++ には、XL C/C++ コンパイラーの自動並列化および OpenMP フィーチャーをサポートするため、SMP ランタイム・ライブラリーが同梱されています。

高性能の数学ライブラリー

XL C/C++ では、バージョン 7 以降、調整された数学組み込み関数から成る、IBM Mathematics Acceleration Subsystem (MASS) ライブラリーを出荷しています。MASS ライブラリーはスレッド・セーフで、対応する `libm` ルーチンに対して改良

されたパフォーマンスを提供します。さらに、MASS ライブラリーは、コード変更を必要とすることなく、使用できます。コンパイラーは 32 ビットと 64 ビットをサポートします。 MASS ベクトル・ライブラリーのバージョン、それぞれ `libmassvp4.a` および `libmassvp4_64.a`。これらのライブラリーには、単精度および倍精度の逆関数および平方根関数用のベクトル・ルーチンが含まれます。

ユーティリティーとコマンド

XL C/C++ Advanced Edition には、プログラム開発を支援するため、以下の特殊なコマンドが用意されています。詳しくは、*XL C/C++ コンパイラー・リファレンス* を参照してください。

vac_configure ユーティリティー
GNU コンパイラーおよびその他の構成情報のロケーションを指定する、構成ファイル vac.cfg を作成するプログラム。 C および C++ コンパイラーでは、構成のためにこの vac.cfg を使用します。

gxlc および gxlc++ ユーティリティー
GNU C または GNU C++ 呼び出しコマンドを対応する **xlc** または **xlc++** コマンドに変換し、XL C/C++ コンパイラーを呼び出す、呼び出しメソッド。このユーティリティーは、GNU コンパイラーでビルドされた既存アプリケーションで使用した makefile に対する変更数を最小限にすることと、XL C/C++ への移行を容易にすることを目的としています。

cleanpdf コマンド
PDFDIR ディレクトリーを管理するために使用されるプロファイル指示フィードバックに関連するコマンド。指定されたディレクトリー、PDFDIR ディレクトリー、または現行ディレクトリーからすべてのプロファイル情報を除去します。

mergepdf コマンド
プロファイル指示フィードバック (PDF) に関連するコマンドで、2 つ以上の PDF レコードを単一のレコードに結合するときに、それぞれのレコードの重要性に応じて重みを付けることができます。これらの PDF レコードは、同一の実行可能ファイルから派生しなければなりません。

resetpdf コマンド
resetpdf コマンドの現行の振る舞いは cleanpdf コマンドと同じで、他のプラットフォームでの以前のリリースとの互換性を保持しています。

showpdf コマンド
プロファイル指示フィードバック・トレーニング実行 (オプション `-qpdf1` および `-qshowpdf` を指定したコンパイル) で実行されたすべてのプロシージャーの呼び出しおよびブロック数を表示するコマンド。

各国語サポート

XL C/C++ は、Unicode 規格、マルチバイト文字、UTF-16 および UTF-32 ストリング・リテラル、複数のロードされたロケール、および双方向性をサポートします。これらのフィーチャーにより、国際化対応アプリケーションを容易に作成できるようになります。

関連参照

- 「*XL C/C++ ランゲージ・リファレンス*」の『ユニコード規格』
- 「*XL C/C++ コンパイラー・リファレンス*」の『各国語サポート』

資料とオンライン・ヘルプ

XL C/C++ Advanced Edition では、以下の形式で製品資料を提供しています。

- README ファイル。
- インストール可能なマニュアル・ページ。
- HTML ベースのヘルプ・システム。
- PDF 文書。

これらの項目は以下の場所にあるか、以下の場所でアクセスできます。

README ファイル	README ファイルは、/opt/ibmcmp/vacpp/7.0 およびインストール CD のルート・ディレクトリーにあります。
マニュアル・ページ	マニュアル・ページは、コンパイラー呼び出しおよび製品に提供されているすべてのコマンド行ユーティリティーのために提供されています。
HTML ベースのヘルプ・システム	HTML ファイルで構成されるヘルプ・システムが提供されています。ヘルプ・システムは、製品のインフォメーション・センターの一部として、オンラインでも使用可能です。
PDF 文書	PDF ファイルは、/opt/ibmcmp/pdf ディレクトリーにあります。Adobe Acrobat Reader で表示および印刷ができます。 Adobe Acrobat Reader をインストールしていない場合には、http://www.adobe.com からダウンロードすることができます。

XL C/C++ PDF 文書の全ライブラリーは、以下のファイルで構成されています。

install.pdf

「*XL C/C++ インストール・ガイド*」には、コンパイラーのインストール、およびマニュアル・ページの使用可能化に関する説明が含まれています。

getstart.pdf

「*XL C/C++ スタートアップ・ガイド*」には、XL C/C++ コンポーネントの概要、新フィーチャーの説明、コンパイル環境およびプロセスのカスタマイズに関する手引き情報、カテゴリー別のコンパイラー・オプションの概要表、パフォーマンス最適化およびチューニングに対する概要、およびアプリケーションの Linux プラットフォームへの移植に関する一般情報が含まれています。

language.pdf

「*XL C/C++ ランゲージ・リファレンス*」には、GNU C および C++ で元々開発されたアプリケーションの移植に関するインプリメンテーション定義の拡張機能などの、C および C++ プログラム言語の IBM インプリメンテーションに関する情報が記載されています。

compiler.pdf

「*XL C/C++ コンパイラー・リファレンス*」には、並列処理での使用も含め、コンパイラー・オプション、プラグマ、マクロ、および組み込み関数についての情報が記載されています。

proguide.pdf

「*XL C/C++ プログラミング・ガイド*」には、他の資料で説明されていない、XL C/C++ を使用したプログラミングについての情報が記載されています。

license.pdf

IBM XL C/C++ Advanced Edition V7.0 for Linux License Information には、製品ライセンスについての情報が含まれています。

追加情報へのアクセス

XL C/C++ の最新情報については、製品資料および以下の URL のサポート・ページをご覧ください。さらに、IBM Technical Support Organization が作成した IBM Redbooks には、実際の経験からの現実的なシナリオに基づいた技術情報が含まれています。

- インフォメーション・センター (http://www.ibm.com/software/awdtools/vacpp/library)
- 製品サポート・サイト (http://www.ibm.com/software/awdtools/ccompilers)
- IBM Redbooks (http://www.redbooks.ibm.com)

XL C/C++ でのアプリケーション開発に役立つと思われる Redbooks としては、以下が挙げられます。

__ *POWER4 Processor Introduction and Tuning Guide*、 SG24-7041-00。

__ *Understanding IBM eServer pSeries Performance and Sizing*、SG24-4810-01。

バージョン 7 の新機能

XL C/C++ Advanced Edition では以下のオペレーティング・システムがサポートされるようになりました。

- SUSE LINUX Enterprise Server 9.0 (SLES 9)
- Red Hat Enterprise Linux 3.0 Update 3 (RHEL 3 U3)
- Red Hat Enterprise Linux 4.0 (RHEL 4)
- Y-HPC (Terra Soft の 64-bit High Performance Computing OS for PowerPC)

XL C/C++ Advanced Edition の新規フィーチャーおよび機能拡張は、以下の 3 つのカテゴリーに分類することができます。パフォーマンスおよび最適化、業界標準への準拠、および使用上の利便性です。

パフォーマンスおよび最適化

新フィーチャーと機能拡張の多くは、最適化とパフォーマンス・チューニングに分類されるものです。

マシン・アーキテクチャーとハードウェア

オプション -qarch および -qtune の改良

コンパイラー・オプション `-qarch` を使用すると、指定されたマシン・アーキテクチャーに生成される特定の命令を制御することができます。また、オプション `-qtune` を使用すると、命令、スケジューリング、および他の最適化を調整し、指定されたハードウェアのパフォーマンスを向上することができます。これらのオプションを共に使用すると、指定されたアーキテクチャーに最良のパフォーマンスを与えるアプリケーション・コードを生成できます。この 2 つのオプションを組み合わせてうまく使用することが、IBM プロセッサーおよびハードウェアを最大限に活用する鍵となります。本リリースでは、これらのオプションの組み合わせを機能拡張することにより、POWER5 および PowerPC 970 ハードウェア・プラットフォームがサポートされ、また使いやすさも向上しました。

`-qarch` で特定のアーキテクチャーを指定した場合、デフォルトの `-qtune` サブオプションを使用してコンパイルすると、そのアーキテクチャーに最適なパフォーマンスを与えるコードを生成します。現在、オプション `-qarch` には、アーキテクチャーのグループを指定することができます。 `-qtune=auto` を使用してコンパイルすると、指定したグループ内の全アーキテクチャーで稼働するコードを生成しますが、命令シーケンスはコンパイルするマシンのアーキテクチャーで最適なパフォーマンスを行うものになります。

`-qarch` のデフォルト設定は、ご使用のプラットフォームによって異なります。

- SLES 9、RHEL 3 U3、または RHEL 4 をご使用の場合、デフォルト設定は `-qarch=ppc64grsq` になりました。
- Y-HPC をご使用の場合、デフォルト設定は `-qarch=ppc970` になりました。

AltiVec (VMX) のサポート

XL C/C++ for Linux は、AltiVec プログラミング・モデルをサポートし、GNU C および C++ コンパイラーとの最大限の互換性を確保するための追加のフィーチャーを提供します。 AltiVec のデータ型および関連する演算は、アーキテクチャーが PowerPC SIMD 拡張 (別名 VMX エンジン) をサポートしていれば、32 ビットおよび 64 ビット・モードで使用可能です。 SIMD (Single Instruction, Multiple Data) 命令セットにより、マイクロプロセッサー・ハードウェアの使用率をより高くできます。コンパイラーは、高レベルの最適化で自動的に SIMD ベクトル化を使用可能にする機能を提供します。

POWER5 プロセッサーのための組み込み関数

以下の組み込み関数が POWER5 プロセッサーでもサポートされるようになりました。サポートされているすべての組み込み関数は、「*XL C/C++ コンパイラー・リファレンス*」に詳しく説明されています。

POWER5 プロセッサーのための組み込み関数

関数	説明
`int __popcnt4(unsigned int);`	32 ビット整数のビット・セット (=1) の数を戻します。
`int __popcnt8 (unsigned long long);`	64 ビット整数のビット・セット (=1) の数を戻します。
`unsigned long __popcntb (unsigned long);`	ソース・オペランドの各バイトで 1 ビットをカウントし、そのカウントを対応する結果のバイトに入れます。
`int __poppar4(unsigned int);`	ビットの奇数が 32 ビットの整数に設定されている場合、1 を戻します。その他の場合は、0 を戻します。
`int __poppar8 (unsigned long long);`	ビットの奇数が 64 ビットの整数に設定されている場合は、1 を戻します。その他の場合は、0 を戻します。
`double __fre(double);`	浮動小数点逆数演算の結果を戻します。結果は、`1/x` の倍精度の推定値です。
`float __frsqrtes(float);`	逆数平方根演算の結果を戻します。結果は、`x` の平方根の逆数の単精度推定値です。
`unsigned long __mfspr(const int);`	指定の特殊目的のレジスターに値を戻します。
`void __mtspr(const int, unsigned long);`	`const int` で指定した特殊目的のレジスターを設定します。
`unsigned long __mfmsr();`	マシン状態レジスターを戻します。
`void __mtmsr(unsigned long);`	マシン状態レジスターを設定します。
`void __protected_unlimited_stream_set_go(unsigned int direction, const void* addr, unsigned int ID);`	`ID ID` を使用する無制限の長さの protected ストリームを確立します。ストリーム ID の範囲は、0 から 15 である必要があります。ストリームは `addr` でキャッシュ・ラインから開始します。ストリームは、`direction` で指定したように、インクリメンタル・メモリー・アドレスまたはデクリメンタル・メモリー・アドレスのいずれかから取り出します。インクリメンタル・メモリー・アドレス (すなわち順方向) の場合 `direction` の値は 1 で、デクリメンタル・メモリー・アドレスの場合 `direction` の値は 3 です。ストリームは、ハードウェア検出ストリームによって置き換えられることはありません。

POWER5 プロセッサーのための組み込み関数

関数	説明
`void __protected_stream_set(unsigned int direction, const void* addr, unsigned int ID);`	ID ID を使用する限られた長さの protected ストリームを確立します。ストリームは addr でキャッシュ・ラインから開始し、その後、direction で指定したように、インクリメンタル・メモリー・アドレスまたはデクリメンタル・メモリー・アドレスのいずれかから取り出します。ストリームは、ハードウェアで検出されるストリームに置き換えられないように保護されます。
`void __protected_stream_count(unsigned int unit_cnt, unsigned int ID);`	ID によって識別する限られた長さの protected ストリームにキャッシュ・ライン数を設定します。キャッシュ・ライン数は、パラメーター unit_cnt によって指定し、0 から 1023 の範囲にする必要があります。
`void __protected_stream_go();`	すべての限られた長さの protected ストリームの事前取り出しを開始します。
`void __protected_stream_stop(unsigned int ID);`	ID によって識別する protected ストリームの事前取り出しを停止します。
`void __protected_stream_stop_all();`	すべての protected ストリームの事前取り出しを停止します。

浮動小数点除法のための新規組み込み関数

このリリースには、浮動小数点除法用の 4 つの新規組み込み関数が組み込まれました。浮動小数点除法アルゴリズムのソフトウェア・インプリメンテーションは、PowerPC アーキテクチャーを利用しており、また、ベクトル・コンテキストでの使用時に、対応するハードウェア命令よりもかなり高速になる可能性があります。これらの新規の組み込み関数は、POWER5 を含む、すべての PowerPC プロセッサーでサポートされます。

ハードウェア除法命令は、ソース・プログラムに浮動小数点除法がコード化されているが、コンパイラーがより高速と考える程度に応じて、ハードウェアまたはソフトウェアの除法コードの間での選択を行う場合、デフォルトで取得されます。ユーザーは新規の組み込み関数を使用することで、明示的にソフトウェア・アルゴリズムを呼び出すことができます。これらのルーチンを呼び出す際、デフォルトの丸めモード (最近似値に丸め) が有効でなければなりません。

浮動小数点除法のための組み込み関数

関数	説明
`double __swdiv_nochk(double, double);`	**double** 型の浮動小数点除法。範囲検査なし。引き数の制約事項: 無限大に等しい分子、または、無限大、ゼロ、または正規化解除に等しい分母は許可されません。分子と分母の商が正または負の無限大であってはなりません。
`double __swdiv(double, double);`	**double** 型の浮動小数点除法。引き数の制限はなし。
`float __swdivs_nochk(float, float);`	**float** 型の浮動小数点除法。範囲検査なし。引き数の制約事項: 無限大に等しい分子、または、無限大、ゼロ、または正規化解除に等しい分母は許可されません。分子と分母の商が正または負の無限大であってはなりません。
`float __swdivs(float, float);`	**double** 型の浮動小数点除法。引き数の制限はなし。

新規の XL C/C++ プラグマ

プラグマ・ディレクティブは「*XL C/C++ コンパイラー・リファレンス*」に詳しく説明されています。

プラグマ	説明
`#pragma novector`	コンパイラーが、直後に続くループを自動的にベクトル化することを禁止します。自動ベクトル化は、配列の連続するエレメントのループで実行される特定の操作を、同時に幾つかの結果を計算するルーチンに対する呼び出しに変換することを意味します。
`#pragma nosimd`	コンパイラーが、直後に続くループで自動的に VMX 命令を生成することを禁止します。
`#pragma unrollandfuse`	ネストされた **for** ループを最適化するプラグマ。コンパイラーに命令して、外部ループの本体部分、つまりループ・ネスト自体を複製し、複製したものを単一のループ・ネストに展開します。
`#pragma stream_unroll`	**for** ループに含まれるストリームを複数のストリームに分割します。大規模な反復数および小規模なストリーム数を持つループを対象としています。
`#pragma block_loop`	コンパイラーに命令して、ループ・ネストの特定の **for** ループのためにブロック化ループを作成します。ブロック化ループとは、ループの反復部分をパーツまたはブロックに分割したものです。*ブロック化ループ* という外部ループを追加で作成し、このループの各ブロックで元のループを実行します。
`#pragma loopid`	**for** ループにスコープ固有の ID でマークを付けます。ID は、そのループでの変換を制御するため、およびオプション `-qreport` の使用によってループ変換に関する情報を提供するために、`#pragma block_loop` および他のプラグマによって使用することができます。
`#pragma disjoint`	C++ インプリメンテーションの追加。
`#pragma unroll` の拡張	ループの展開とは、ループを完了するのに必要な反復数を減らすために、ループの本体部分を複製することです。#pragma アンロール・ディレクティブを指定すると、このディレクティブの直後に記述されている **for** ループが展開可能であることをコンパイラーに示します。このプラグマの機能が拡張され、最内部および最外部の両方の **for** ループに適用することができるようになりました。ただし、拡張された #pragma 機能では、代替エントリー・ポイントを持つ **for** ループへの適用はまだ含まれていません。

新しい最適化ユーティリティー

このリリースには、プロファイル指示フィードバック (PDF) コンパイル・プロセスに関連した 2 つの新しいユーティリティーが含まれています。プロファイル指示フィードバックを使用することにより、コンパイラーは、その実行可能ファイルを幾つかの異なるシナリオで実行した結果に基づいて、実行可能ファイルを最適化できます。計測する実行可能ファイルをいずれかのシナリオで実行すると、副次的に

PDF レコードも作成されます。このレコードは、そのプログラムの典型的な動作を定義するために照合するデータとなります。

showpdf コマンドでは、プロファイル指示フィードバック・トレーニング実行として実行したすべてのプロシージャーの呼び出しおよびブロック数を表示できます。このユーティリティーでは、オプション `-qpdf1` と `-qshowpdf` を指定してコンパイルする必要があります。

mergepdf コマンドでは、ユーザーが 2 つ以上の PDF レコードの関連する重要な部分を指定し、1 つのレコードに組み合わせることができます。これにより、ユーザーはより高い実行カウント (より長い実行時間) のトレーニング実行を補正し、そのトレーニング実行だけがプロファイル・データを決定付けてしまわないようにすることができます。

MASS ベクター・ライブラリーのサポート

コンパイラーは、IBM Mathematics Acceleration Subsystem (MASS) ベクター・ライブラリーの 32 ビットおよび 64 ビットのモード・バージョンに対するサポートを追加します。それぞれ、`libmassvp4.a` および `libmassvp4_64.a` です。これらのライブラリーには、単精度および倍精度の逆数および平方根関数用のベクター・ルーチンが含まれます。ベクター・ライブラリーはスレッド・セーフで、対応する `libm` ルーチンに対して改良されたパフォーマンスを提供します。

バージョン 7.0 から、コンパイラーとともに MASS ライブラリーが配布されるようになりました。

業界標準の準拠

C、C++、および Fortran に対する OpenMP API V2.0 サポート

OpenMP アプリケーション・プログラム・インターフェース (API) は、移植可能で拡張が容易なプログラミング・モデルであり、マルチプラットフォームでメモリーを共用する並列アプリケーションを C、C++、および Fortran で開発するための標準インターフェースを提供します。 OpenMP API の仕様は、OpenMP 組織という、IBM を含む主なコンピューター・ハードウェアおよびソフトウェア・ベンダーの集まりによって定義されます。 XL C/C++ Advanced Edition は OpenMP 仕様 2.0 に準拠しています。コンパイラーは、以下の OpenMP V2.0 エレメントのセマンティクスを認識し、保持します。

- `#pragma omp` ディレクティブ内の複数文節のコンマ区切り文字。
- `num_threads` 文節。
- `copyprivate` 文節。
- `threadprivate` 静的ブロック・スコープ変数。
- C99 可変長配列のサポート。
- private 変数の冗長な宣言。
- タイミング・ルーチン `omp_get_wtime` および `omp_get_wtick`。

拡張ユニコードおよび NLS サポート

C 標準委員会の最近のレポートでは、C コンパイラーで C99 を拡張し、新規データ型を追加して UTF-16 および UTF-32 リテラルをサポートすることを推奨してい

ます。また、C++ コンパイラーでも、C との互換性のために、これらの新規データ型をサポートすることを推奨しています。

Boost ライブラリーに対するサポート

XL C++ コンパイラーは、1.30.2 Boost ライブラリーでの高レベルの互換性を実現します。これらのライブラリーは再使用可能セット、標準化に適切な Open Source C++ ライブラリーを提供するために作成されました。詳しくは、Boost Web サイト http://www.boost.org を参照してください。

GNU C および C++ に関連する言語拡張機能

C99 に対する GNU C 拡張および標準 C++ に対する GNU C++ 拡張は、業界標準ではありません。 しかし、独自のものではなく、ある程度一般的となっているオープン・ソース・コミュニティーの言語フィーチャーです。 XL C/C++ は、GNU C および C++ 拡張のサブセットをインプリメントしています。本リリースでは、以下の GNU C フィーチャーに対するサポートが追加されています。

フィーチャー	注釈
ラベルを値として使用	計算済み **goto** 文を含む。このフィーチャーは現在、GNU C インプリメンテーションと完全に互換性があります。
型属性	属性 aligned、packed、transparent_union。
関数属性	属性 format、format_arg、always_inline、noinline。
変数属性	属性 `section` に追加された C++ サポート。
代替キーワード	**__extension__** のインプリメンテーションを内部的に変更。
ネストされた関数	C C のみサポート。
共用体型へのキャスト	C C のみサポート。
引き数の変数番号を含むマクロ	__VA_ARGS__ 引き数が指定されていない場合、__VA_ARGS__ の代わりに ID を使用し、末尾のコンマを除去する。
C の式オペランドを使用する gcc インライン・アセンブラー命令	部分的なサポートのみ。
GNU C 複素数型	C++ サポートが追加されました。
GNU C 16 進浮動小数点定数	C++ サポートが追加されました。
C99 複合リテラル	C++ サポートが追加されました。
長さゼロの配列	C++ サポートが追加されました。
可変長配列	C++ サポートが追加されました。

使用上の利便性

新規 C++ コンパイラー呼び出し

コンパイラー呼び出し **xlc++** が、すべてのサポートされているプラットフォームで移植可能になりました。この呼び出しは、すべてのプラットフォームで使用可能な呼び出し **xlC** と同等であり、使用を推奨されています。ただし、**xlC** も完全にサポートされています。

資料

各コンパイラー呼び出しコマンドと各コマンド行ユーティリティーに関するマニュアル・ページが提供されています。コンパイラー呼び出しに対するマニュアル・ページは、前のリリースで提供されていたテキストのヘルプ・ファイルを置き換えるものです。

テンプレート・レジストリー機能拡張

C++ コンパイラーは、テンプレートのインスタンス化の登録による、テンプレートのインスタンス化のバッチ処理スキームを使用します。本リリースでは、このコンパイラーにより、作成されるテンプレート・レジストリー・ファイルにバージョン管理情報を追加しています。この情報は、コンパイラーによって内部的に使用され、どのバージョンのテンプレート・レジストリー・ファイル・フォーマットを使用する必要があるかを追跡します。

新規の XL C/C++ オプション

新規および変更されたコンパイラー・オプションが、「*XL C/C++ コンパイラー・リファレンス*」で詳しく説明されています。

オプション	説明および注釈
-qabi_version=*n*	バージョン *n* の C++ ABI を使用するように、コンパイラーに指示します。ただし、n には以下を指定できます。 • 1。 G++3.2 で導入され、G++3.3 でも採用されている C++ ABI 振る舞いと同じ振る舞いが行われるようにします。 • 2。 G++3.4 で導入された C++ ABI 振る舞いと同じ振る舞いが行われるようにします。 デフォルトはオペレーティング・システムによって異なります。 • SLES 9、RHEL 3 U3、および Y-HPC では **1** • RHEL 4 では **2**
-qaltivec	AltiVec データ型に対するコンパイラー・サポートが使用可能です。デフォルトは **-qnoaltivec** です。
-qasm=gcc	C の式オペランドによるアセンブラー命令に対する部分的サポートを使用可能にします。 **asm** キーワードおよびその代替スペルを認識し、gcc 構文およびキーワードのセマンティクスを使用するようコンパイラーに命令します。デフォルトは、**-qnoasm** です。
-qasm_as	**asm** ディレクティブでコードを処理するために、代替アセンブラー・プログラムを呼び出すのに使用するパスおよびフラグを指定します。このオプションは、コンパイラー構成ファイルで定義される **as** コマンドのデフォルト設定をオーバーライドします。

オプション	説明および注釈
-qdirectstorage	ライトスルー使用可能またはキャッシュ禁止ストレージが、一定のコンパイル単位で参照される可能性があることを表明します。このオプションの意図は、PowerPC アーキテクチャーで使用可能な異なるストレージ管理属性のために、予期しない振る舞いを行わないようにすることです。デフォルトは、**-qnodirectstorage** です。
-qenablevmx	コンパイラーに対し、任意のコンパイラー・フェーズで VMX (AltiVec) コードを生成するよう命令します。このオプションによって、開発環境におけるオペレーティング・システムに対する **-qaltivec** の正しいデフォルト設定を確実にします。RHEL 3 U3 および Y–HPC の場合、デフォルトは**-qnoenablevmx** です。RHEL 4 および SLES 9 の場合、デフォルトは **-qenablevmx** です。
-qkeepparm	パフォーマンスを改善するために、異なるメモリー・ロケーションに移動させる代わりに、レジスターに渡される関数のパラメーターがスタックに確実に保管されるようにします。デフォルトは、**-qnokeepparm** です。
-qipa=threads[=N] | nothreads	N スレッドを作成し、並行して最大 N 個のバックエンドを実行するように、最適化プログラムに指示します。ここで、N は、1-MAXINT の範囲の整数です。デフォルトは **nothreads** サブオプションです (これは、単一シリアル処理の実行に相当します)。このフィーチャーを使用すると、マルチプロセッサー・コンピューターで IPA リンクに要する時間を削減することができます。
-qnoprefetch	自動的にソフトウェア事前取り出し命令を挿入しないようにコンパイラーに命令します。このオプションを使用すると、ユーザーは、事前取り出しという最適化の機能をオフにすることができます。デフォルトは、**-qprefetch** です。
-qnotrigraph	どの言語レベルを指定しているかに関わらず、3 文字表記を解釈しないようにコンパイラーに命令します。 Linux では、デフォルトは **-qtrigraph** です。
-qsaveopt	ソース・ファイルを対応するオブジェクト・ファイルにコンパイルするコマンド行オプションを保管するようにコンパイラーに命令します。このオプションは、コンパイルで `.o` ファイルが生成されない場合は効果がありません。デフォルトは、**-qnosaveopt** です。
-qshowpdf	**-qpdf1** で指定し、また最適化レベルが **-O** 以上である場合、コンパイラーは追加のプロファイル情報をコンパイル済みアプリケーションに挿入して、アプリケーション内のすべてのプロシージャーに対する呼び出しおよびブロック数を収集します。コンパイルされたアプリケーションを実行すると、呼び出しおよびブロック数をファイル `._pdf` に記録します。 `._pdf` の内容は **showpdf** ユーティリティーで検索することができます。デフォルトは、**-qnoshowpdf** です。
-qsourcetype	入力ファイル名の解釈を制御します。デフォルトの振る舞いは、ソース・ファイルのプログラミング言語がそのファイル名のサフィックスによって示すものです。デフォルトは、**-qsourcetype** です。
-qutf	ユニコード・エンコード形式に 16 ビットおよび 32 ビットのストリング・リテラルを提供する UTF リテラル構文の認識を使用可能にします。

オプション	説明および注釈
-qfltrap=nanq	NaNQ (Not a Number Quiet) をキャッチするために、コードに追加の命令を生成するようにコンパイラーに命令します。このオプションは、有効な演算によって作成されたものを含む、浮動小数点命令によって処理または生成された、すべての NaNQ を検出することを目的としています。
-qhot=simd	コンパイラーに自動 SIMD ベクトル化を試行するように命令します。デフォルトは、**-qhot=nosimd** です。
-qipa=infrequentlabel	標準的なプログラム実行中にまれに呼び出されることがあるラベルのリストを指定します。これらのラベルの呼び出しで最適化を行う対象を限定することによって、コンパイラーでプログラムのその他の部分をより高速に処理できる場合があります。このオプションは、ユーザー定義のラベルにのみ適用されます。

コンパイル環境のカスタマイズ

このセクションでは、インクルード・ファイル、ライブラリー、および GNU C または C++ コンパイラーのロケーションを含むディレクトリーの検索パスを指定するために XL C/C++ で使用される機構について説明します。これらの機構は、環境変数、インクルード・ファイル、構成ファイルの属性、およびコマンド行オプションです。有効な構成ファイルを作成しやすくするため、vac_configure ユーティリティーが用意されています。

XL C/C++ の重要な検索パスは、以下のものの標準ディレクトリー・ロケーションです。

- 32 ビット GNU コンパイラー、64 ビット GNU コンパイラー、または両方
- GNU C インクルード・ファイル
- GNU C++ インクルード・ファイル
- IBM C ヘッダーおよび C++ ヘッダー
- GNU C ライブラリー・パス

関連参照

- vac_configure について詳しくは、「*XL C/C++ インストール・ガイド*」を参照してください。

環境変数

コンパイル環境の一部に、ライブラリーやインクルード・ファイルなどの特殊ファイルの検索パスがあります。コンパイラーでは次のシステム変数が使用されます。

LD_LIBRARY_PATH
動的にロードされたライブラリーのディレクトリー・パスを指定します。リンク時と実行時に GNU リンカーによって使用されます。

LD_RUN_PATH
動的にロードされたライブラリーを実行時に検索する追加のディレクトリー・パスを指定します。この設定は、GNU リンカーでリンク時に使用される検索パスには影響しません。

MANPATH 製品マニュアル・ページのディレクトリー・パスを指定します。

NLSPATH 各国語サポート・ライブラリーのディレクトリー・パスを指定します。

PATH コンパイラーの実行可能ファイルのディレクトリー・パスを指定します。

PDFDIR プロファイル・データ・ファイルを作成するディレクトリーを指定します。デフォルト値は設定されず、コンパイラーはプロファイル・データ・ファイルを現行作業ディレクトリーに入れます。プロファイル指示フィードバックの場合は、この変数を絶対パスに設定することをお勧めします。

TMPDIR 一時ファイルを作成するディレクトリーを指定します。デフォルト

のロケーションは、高レベルの最適化には不適切である場合があります。高レベルの最適化の場合は、一時ファイルがディスク・スペースを大量に消費する可能性があるためです。

呼び出しコマンドのための環境のセットアップ

XL C/C++ のコマンド行インターフェースは、/usr/bin に自動的にはインストールされません。絶対パスを指定せずにコンパイラーを呼び出すには、以下のステップのうちの 1 つを行なってください。

- /opt/ibmcmp/vacpp/7.0/bin および /opt/ibmcmp/vac/7.0/bin に含まれる特定のドライバーのための /usr/bin へのシンボリック・リンクを作成します。
- PATH 環境変数に /opt/ibmcmp/vacpp/7.0/bin を追加します。

メッセージ・カタログの正しい NLSPATH の確保

NLSPATH 環境変数は、適切なメッセージ・カタログを検出する方法をコンパイラーに通知します。

パスが正しいことを確認するには、次のコマンドを出します。

```
export NLSPATH=$NLSPATH:smprt-path/msg/%L/%N:
  compiler-path/vacpp/6.0/msg/%L/%N
```

ここで、*smprt-path* および *compiler-path* は、パッケージのインストール時に指定したインストール・ロケーションです。

注: デフォルトのインストール・ロケーションを使用する場合、*smprt-path* および *compiler-path* は両方とも `/opt/ibmcmp` になります。

インクルード・ファイル

GNU、IBM、およびシステムのヘッダー・ファイルのロケーションの指定を最も簡便に行うには、構成ファイルで指定します。

コンパイラー・オプション **-I** *directory_name* を使用すると、構成ファイル内の検索パスへディレクトリーを追加できます。構成ファイル自体は、制御するディレクトリー・パスを設定するために内部的に **-I** オプションを使用します。コンパイラーは、コマンド行の **-I** オプションによって指定されるディレクトリーを検索する前に、構成ファイル内の **-I** によって指定されるディレクトリーを検索します。

詳細については、「*XL C/C++ コンパイラー・リファレンス*」を参照してください。

構成ファイル

構成ファイルは、コンパイラーを実行するたびに読み取られるオプションを指定するプレーン・テキスト・ファイルです。構成ファイルの名前は、.cfg ファイル名拡張子で終わります。

デフォルトの構成ファイル (/etc/opt/ibmcmp/vac/7.0/vac.cfg) が用意されると、他の構成ファイルを作成できるようになります。 XL C/C++ にはテンプレートが用意されています (これは /etc/opt/ibmcmp/vac/7.0/vac.cfg を作成するために使用されたテンプ

レートです)。ただし、既存の構成ファイルを、vac_configure ユーティリティーによって別の構成ファイルを作成するためのテンプレートとして使用することができます。

-F オプションを指定してコンパイラーを起動し、完全修飾ファイル名を指定することで、特定の構成ファイルを使用するようコンパイラーに指示することができます。

コマンド行オプション

標準インクルード・パスを制御するコンパイラー・オプションは、以下の表に示してあります。 *paths* の値は、コマンド行でユーザーが指定します。構成ファイルを使用した場合、*paths* の値は、2 番目の列で名前を指定した属性の値です。デフォルトで使用される構成ファイルは、vac_configure ユーティリティーで作成されたものです。コンパイラーは構成ファイルの各属性を処理して、インクルード・ファイルのために適切な検索パスが使用されるようにするために、対応するオプションを作成します。

Linux 特有の構成オプションおよび関連した属性、32 ビット・モードおよび 64 ビット・モード

オプション名	属性	使用法	競合解決
-qgcc_c_stdinc=<*paths*>	gcc_c_stdinc gcc_c_stdinc_64	gcc ヘッダーの検索ロケーションを指定します。デフォルト値はありません。	同じコマンドで複数回指定した場合、最後のものが使用されます。 **-qnostdinc** オプションが有効な場合、このオプションは無視されます。
-qgcc_cpp_stdinc=<*paths*>	gcc_cpp_stdinc gcc_cpp_stdinc_64	g++ ヘッダーの検索ロケーションを指定します。デフォルト値はありません。	同じコマンドで複数回指定した場合、最後のものが使用されます。 **-qnostdinc** オプションが有効な場合、このオプションは無視されます。
-qc_stdinc=<*paths*>	xlc_c_stdinc xlc_c_stdinc_64	IBM C ヘッダーの標準インクルード・ファイルの検索ロケーションを指定します。デフォルト値はありません。	同じコマンドで複数回指定した場合、最後のものが使用されます。 **-qnostdinc** オプションが有効な場合、このオプションは無視されます。
-qcpp_stdinc=<*paths*>	xlc_cpp_stdinc xlc_cpp_stdinc_64	IBM C++ ヘッダーの検索ロケーションを指定します。デフォルト値はありません。	同じコマンドで複数回指定した場合、最後のものが使用されます。 **-qnostdinc** オプションが有効な場合、このオプションは無視されます。

関連参照

- 「*XL C/C++ コンパイラー・リファレンス*」の『コンパイラー・コマンド行』

コンパイル・プロセスの制御

コンパイル・プロセス全体は、以下の 3 つのフェーズから構成されています。プリプロセス、オブジェクト・コードへの変換、およびリンクです。デフォルトでは、コンパイラー呼び出しコマンドは、コンパイル・プロセスのすべてのフェーズを呼び出し、ソース・コードからプログラムを変換して、出力として実行可能ファイルを作成します。コンパイラーを呼び出す際、入出力ファイルのファイル名を指定すると、コンパイラーは、入出力ファイルのファイル名サフィックス (拡張子) から開始および終了フェーズを判断します。

また、適切なコンパイラー・オプションを使用すると、どのコンパイル・フェーズでも特定のタイプの出力ファイルを作成することができます。例えば、**xlc** または **xlC** コマンドを **-E** または **-P** オプションを指定して呼び出すと、入力ファイルに対してプリプロセス・フェーズだけが実行されます。コンパイラー呼び出しにより、IBM コンパイラー、アセンブラー、またはリンカーを呼び出すかどうかを入力ファイル名の拡張子から判断します。アセンブラーとリンカーは、Linux オペレーティング・システムで提供されるツールです。

関連参照
- 「*XL C/C++ コンパイラー・リファレンス*」の『-qphsinfo コンパイラー・オプション』

コンパイラーの呼び出し

XL C/C++ では基本コンパイラー呼び出しコマンドを選択でき、さまざまなバージョン・レベルの C および C++ 言語をサポートしています。各呼び出しコマンドは、言語レベルのコンパイラー・サブオプション、その他の関連言語フィーチャーのオプション、および関連する定義済みマクロを自動的に設定します。多くの場合、C ソース・ファイルをコンパイルするときは **xlc** コマンドを使用し、C++ ソース・ファイルをコンパイルするとき、または C と C++ の両方のソース・ファイルが存在するときは **xlC** コマンドを使用します。スレッド化されたアプリケーションの要件をサポートするために、基本コマンドにさまざまなバリエーションが提供されています。これらのコマンドは、基本コマンドにサフィックス *_r* を追加した形式になっています。

サポートされている呼び出しコマンド

xlc	xlc_r	*_r* 呼び出しは、スレッド・セーフなアプリケーションのコンパイル (「*再入可能コンパイラー呼び出し*」とも呼ばれます) に使用されます。
xlc++	xlc++_r	
xlC	xlC_r	
cc	cc_r	
c89	c89_r	
c99	c99_r	

xlc++ 呼び出しは **xlC** 呼び出しと同等です。

また、gxlc および gxlc++ ユーティリティーは特殊なコンパイラー呼び出しです。

オブジェクト・モデル

Linux プラットフォームには、オブジェクト・モデルは 1 つしかありません。 AIX (ibm または classic) でサポートされているオブジェクト・モデルに依存するアプリケーションを移植するとき、コードの変更が必要になる場合があります。特定のオブジェクト・モデルを指定するためのコンパイラー・オプションおよびプラグマは、Linux には使用できません。

入出力ファイルの種類

コンパイラーは、ファイル名拡張子を使用して、適切なコンパイル・フェーズを判別し、関連するツールを呼び出します。

コンパイラーは、入力として次のタイプのファイルを受け入れます。

受け入れられる入力ファイル・タイプ

ファイル・タイプの説明	ファイル名拡張子	例
C および C++ ソース・ファイル	C 言語ソース・ファイルの場合は .c (小文字の *c*) C++ ソース・ファイルの場合は .C (大文字の *c*)、.cc、.cp、.cpp、.cxx、.c++	*file_name*.c *file_name*.C、*file_name*.cc、*file_name*.cpp、*file_name*.cxx、*file_name*.c++
プリプロセスされたソース・ファイル	.i	*file_name*.i
オブジェクト・ファイル	.o	`hello.o`
アセンブラー・ファイル	.s	`check.s`
アーカイブ・ファイル	.a	`v1r5.a`
ロード可能なモジュールまたは共用ライブラリー・ファイル	.so	`my_shrlib.so`
IPA 制御ファイル (**-qipa=***file_name*)	*file_name* の命名規則は強制されない。	`ipa.ctl`

コンパイラーを呼び出すときに、次のタイプの出力ファイルを指定できます。

出力ファイルの種類

ファイル・タイプの説明	例
実行可能ファイル	デフォルトでは、`a.out`
オブジェクト・ファイル	*file_name*.o
ロード可能なモジュールまたは共用ファイル	*file_name*.so
アセンブラー・ファイル	*file_name*.s
プリプロセスされたファイル	*file_name*.i
リスト・ファイル	*file_name*.lst
ターゲット・ファイル	*file_name*.d

関連参照

- 「*XL C/C++ コンパイラー・リファレンス*」の『-qsourcetype』コンパイラー・オプション

デフォルトの動作

オプションを指定せずにコンパイラーを呼び出すと、コンパイラーの動作は、次のデフォルト設定によって制御されます。

- 構成ファイルに指定されたオプションを読み取り、呼び出そうとする。
- LD_LIBRARY_PATH 変数のディレクトリーでライブラリー・ファイルを検索する。
- デフォルトの位置合わせ -qalign=linuxppc を使用して構造体の位置合わせをする。
- 最適化されていない a.out という名前の実行可能ファイルを現行ディレクトリーに生成する。
- AltiVec プログラミング構成体を構文エラーとして診断する。

詳細については、「*XL C/C++ コンパイラー・リファレンス*」を参照してください。

コンパイラー・オプションについて

コンパイラー・オプションは、さまざまな関数を実行します。例えば、コンパイラー特性の設定、生成されるオブジェクト・コードの記述、出力される診断メッセージの制御、および一部のプリプロセッサー関数の実行などです。コンパイラー・オプションは、コマンド行、構成ファイル内、ソース・コード内、またはこれらの技法の組み合わせで指定できます。明示的に設定されないほとんどのオプションが、デフォルト設定を受け入れます。

複数のコンパイラー・オプションを指定した場合、オプションの矛盾や非互換が起きる可能性があります。このような矛盾を一貫性のある方法で解決するために、これ以外の順位が指定されていない限り、コンパイラーは次の優先順位を適用します。

1. ソース・ファイルのオーバーライド
2. コマンド行のオーバーライド
3. 構成ファイルのオーバーライド
4. デフォルト設定

一般に、複数のコマンド行オプションでは、最後に指定されたものが優先されます。

注: **-I** コンパイラー・オプションは特殊なケースです。コンパイラーは、コマンド行で **-I** を使用して指定されたディレクトリーを検索する*前*に、vac.cfg ファイル内の **-I** で指定されたディレクトリーを検索します。これらのオプションは、先に認識されたものが優先されるのではなく、後から認識されたものが優先されます。

後から認識されたものが優先されるその他のオプションには、**-R** と **-l** (小文字の L) があります。

関連参照

- 詳細については、「*XL C/C++ コンパイラー・リファレンス*」を参照してください。

コンパイラー・メッセージ

XL C/C++ は、診断メッセージを 5 つのレベルに分類します。各重大度レベルは、コンパイラーの応答に関連付けられています。すべてのエラーでコンパイルが停止されるわけではありません。次の表に、重大度レベルに割り当てられた省略形と、関連付けられているコンパイラーの応答を示します。

重大度レベルとコンパイラーの応答

文字	重大度	コンパイラーの応答
I	通知	コンパイルは継続します。このメッセージは、コンパイルと途中で検出された条件を報告します。

重大度レベルとコンパイラーの応答

文字	重大度	コンパイラーの応答
W	警告	コンパイルは継続します。このメッセージは、有効であり、意図したものではない条件を報告します。
C E	エラー	コンパイルは継続し、オブジェクト・コードが生成されます。コンパイラーが訂正することのできるエラー条件が存在しますが、プログラムは期待される結果を作成できない可能性があります。
S	重大エラー	コンパイルは継続しますが、オブジェクト・コードは生成されません。コンパイラーが訂正できないエラー条件が存在します。
U	回復不能エラー	コンパイラーは停止します。回復不能エラーが見つかりました。メッセージがリソースの限界 (例えばファイル・システムまたはページング・スペースがいっぱいになった) を示している場合は、リソースを追加して再コンパイルしてください。異なるコンパイラー・オプションが必要であると示された場合は、それらを使用して再コンパイルしてください。メッセージが内部コンパイラー・エラーを示す場合は、そのメッセージを IBM 技術員に報告してください。

コンパイラーのデフォルトの動作は、オプション **-qnoinfo** または **-qinfo=noall** でコンパイルすることです. **-qinfo** のサブオプションは、特定のカテゴリーの情報診断を指定する機能を提供します。例えば、**-qinfo=por** は、移植性の問題に関連するメッセージへの出力を制限します。

注: C では、サブオプションなしで指定されるオプション **-qinfo** は **-qinfo=all** に相当します。C++ では、サブオプションなしで指定される **-qinfo** は **-qinfo=all:noppt** に相当します。

戻りコード

コンパイルの終了時に、コンパイラーは、以下の任意の条件のもとで戻りコードをゼロに設定します。

- メッセージが発行されない。
- 診断されたすべてのエラーの中の最高重大度レベルが、**-qhalt** コンパイラー・オプションの設定値よりも小さく、かつエラーの数が **-qmaxerr** コンパイラー・オプションで設定した限界値に達していない。
- **-qhaltonmsg** コンパイラー・オプションで指定されたメッセージが発行されない。

それ以外の場合、コンパイラーは「*XL C/C++ コンパイラー・リファレンス*」に記載されている戻りコードの 1 つを設定します。

コンパイラー・メッセージ・フォーマット

デフォルトでは、診断メッセージは次の形式になります。

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

ここで、15 はコンパイラー製品 ID、*cc* は、メッセージを発行したコンパイラー・コンポーネントを示す 2 桁のコード、*nnn* はメッセージ番号、さらに、*severity* は重大度レベルの文字です。 *cc* に指定できる値は、以下のとおりです。

00 メッセージを生成または最適化するコード
01 コンパイラー・サービス・メッセージ
05 C コンパイラーに固有のメッセージ
06 C コンパイラーに固有のメッセージ
40 C++ コンパイラーに固有のメッセージ
86 プロシージャー間分析 (IPA) に固有のメッセージ

この形式は、**-qnosrcmsg** オプションを使用可能にしてコンパイルする場合と同じです。診断メッセージとともにソース行を表示する代替メッセージ・フォーマットにするには、**-qsrcmsg** オプションでのコンパイルを試行してください。このオプションを使用可能にすると、エラーがあるとコンパイラーが判断したソース行、その下 (2 行目) にそのソース行の特定の箇所を指し示す行 (可能な場合)、および診断メッセージを、標準エラーに出力するようにコンパイラーに指示します。

注: メッセージは、他のプログラムの入力として使用するためのものではありません。メッセージのフォーマットおよび内容は、プログラミング・インターフェースにするためのものではなく、リリース毎に変更する可能性があります。

プラットフォーム固有のオプション

この節では、Linux プラットフォームでのコンパイラー使用の基本となるオプションについて説明します。

詳細については、「*XL C/C++ コンパイラー・リファレンス*」を参照してください。

Linux プラットフォーム特有の選択可能コンパイラー・オプション

オプション名	説明
-qgcc_c_stdinc=<*paths*>	GNU C ヘッダーのディレクトリー検索パスを指定します。
-qgcc_cpp_stdinc=<*paths*>	GNU C++ ヘッダーのディレクトリー検索パスを指定します。
-qc_stdinc=<*paths*>	IBM C ヘッダーのディレクトリー検索パスを指定します。
-qcpp_stdinc=<*paths*>	IBM C++ ヘッダーのディレクトリー検索パスを指定します。

次のオプションは、ディレクトリー検索パス特有の制御を提供します。これらのオプションは、先に認識されたものが優先されるのではなく、後から認識されたものが優先されます。オプション **-L**、**-R**、および **-l** (小文字の *L*) でコマンド行に指定されたパスは、リンク時の優先順位が、構成ファイルにオプションとして指定されたパスよりも低くなりますが、構成ファイルの属性として指定されたパスよりは高くなります。

選択可能なパス制御オプション

オプション名	説明
-I	`#include` ファイルを検索する追加のディレクトリー・パスを指定します。
-l	静的にリンクする共用ライブラリーまたはアーカイブ・ファイルを指定します。

選択可能なパス制御オプション

オプション名	説明
-L	リンク時に検索するライブラリー検索パスを指定します。
-R	動的にロードされたライブラリーを実行時に検索するディレクトリー・パスを指定します。

関連参照

- *XL C/C++ プログラミング・ガイド* の「最適化およびパフォーマンスに関するオプションの要約」

gxlc および gxlc++ を使用した GNU C および C++ コンパイラー・オプションの再利用

各 gxlc および gxlc++ ユーティリティーは、GNU C または C++ コンパイラー・オプションを受け取り、それを同等の XL C/C++ オプションに変換します。両方のユーティリティーは、XL C/C++ オプションを使用して **xlc** または **xlC** 呼び出しコマンドを作成し、それを使用して XL C/C++ を呼び出します。これらのユーティリティーは、GNU C および C++ で以前に開発されたアプリケーション用に作成された Make ファイルを簡単に再利用するために提供されています。ただし、XL C/C++ の機能を十分に活用するためには、XL C/C++ 呼び出しコマンドおよびその関連オプションを使用することをお勧めします。

gxlc および gxlc++ のアクションは、構成ファイル `gxlc.cfg` によって制御されます。XL C または XL C++ に対応する GNU C および C++ オプションは、このファイルに示されています。すべての GNU オプションに対応する XL C/C++ オプションがあるわけではありません。 gxlc および gxlc++ は、変換されなかった入力オプションについて警告を戻します。

gxlc および gxlc++ オプション・マッピングは変更可能です。 gxlc および gxlc++ 構成ファイルへの追加または編集の詳細については、32 ページの『オプション・マッピングの構成』を参照してください。

例

Hello World プログラムの C バージョンをコンパイルするために gcc `-ansi` オプションを使用するには、以下を使用することができます。

```
gxlc -ansi hello.c
```

これは、以下のように変換されます。

```
xlc -F:c89 hello.c
```

変換後、このコマンドを使用して XL C コンパイラーを呼び出します。

gxlc および gxlc++ 戻りコード

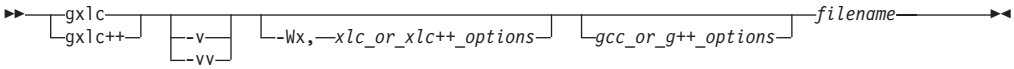
他の呼び出しコマンドのように、gxlc および gxlc++ はリスト、コンパイルに関する診断メッセージ、GNU オプションの変換の失敗に関する警告、および戻りコードなどの、出力を戻します。 gxlc または gxlc++ が正常にコンパイラーを呼び出すことができない場合、戻りコードを以下のいずれかの値に設定します。

40 gcc または g++ オプション・エラーまたは回復不能エラーが検出されました。

255 プロセスの実行中にエラーが検出されました。

gxlc および gxlc++ 構文

以下の図は gxlc および gxlc++ 構文を示しています。



以下、説明です:

filename
コンパイルするファイルの名前です。

-v XL C/C++ を呼び出すのに使用されるコマンドを確認することができます。 gxlc または gxlc++ は、作成した XL C/C++ 呼び出しコマンドを、コンパイラーの呼び出しに使用する前に表示します。

-vv シミュレーションを実行することができます。 gxlc または gxlc++ は、作成した XL C/C++ 呼び出しコマンドを表示しますが、コンパイラーを呼び出しません。

-Wx,*xlc_or_xlc++_options*
指定した XL C/C++ オプションを xlc または xlc++ 呼び出しコマンドに直接送信します。 gxlc または gxlc++ は、作成している XL C/C++ 呼び出しに、指定したオプションを変換せずに追加します。このオプションを既知の XL C/C++ オプションと共に使用し、ユーティリティーのパフォーマンスを改善します。複数の *xlc_or_xlc++_options* がコンマ区切り文字を使用します。

gcc_or_g++_options
xlc または xlc++ オプションに変換する gcc または g++ オプションです。ユーティリティーは、変換できないオプションに対しては警告を発行します。現在 gxlc および gxlc++ によって認識される gcc および g++ オプションは、構成ファイル gxlc.cfg にリストされています。複数の *gcc_or_g++_options* は、スペース文字で区切られています。

GNU C および C++ から XL C/C++ へのオプション・マッピング

以下のテーブルは、gxlc および gxlc++ が受け取り、変換する GNU C および C++ オプションをリストしています。リストにない GNU オプションをこれらのユーティリティーへの入力として指定した場合、そのオプションは無視されるか、またはエラーを生成します。 GNU オプションにマイナス記号がある場合、マイナス記号も gxlc および gxlc++ によって認識され、変換されます。

オプション・マップ: GNU C および C++ から XL C/C++

GNU C および C++ オプション	XL C/C++ オプション
-###	-#
-ansi	-F:c89
-B	-B

オプション・マップ: GNU C および C++ から XL C/C++

GNU C および C++ オプション	XL C/C++ オプション
-C	-C
-c	-c
-D*macro[=defn]*	-D*macro[=defn]*
-E	-E
-e	-e
-fdollars-in-identifiers	-qdollar
C++ -fdump-class-hierarchy	-qdump_class_hierarchy
C++ -fexceptions	-qeh
C++ -ffor-scope	-qlanglvl=ansifor
C++ -fno-for-scope	-qlanglvl=noansifor
-ffunction-sections	-qfuncsect
-finline	-qinline
-finline-functions	-qinline **注: -qinline=limit=n** は、XL C/C++ において有効なサブオプションではありません。**-finline-limit=n** GNU C および C++ サブオプションをこの XL C/C++ サブオプションにマップすることはできません。
C++ -fkeep-inline-functions	-qkeepinlines
C++ -fno-gnu-keywords	-qnokeyword=typeof
C++ -fno-operator-names	-qnokeyword=and -qnokeyword=bitand -qnokeyword=bitor -qnokeyword=compl -qnokeyword=not -qnokeyword=or -qnokeyword=xor
-fpascal-strings	-qmacpstr
-fPIC	-qpic=large
-fpic	-qpic=small
C++ -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

オプション・マップ: GNU C および C++ から XL C/C++

GNU C および C++ オプション	XL C/C++ オプション
-g3	-g
-gdwarf	-g
-ggdb	-g
-I*dir*	-I*dir*
-L*dir*	-L*dir*
-l*library*	-l*library*
-M	-M
-MD	-M
-mcpu=power	-qarch=pwr
-mcpu=powerpc	-qarch=ppc
-mcpu=powerpc64	-qarch=ppc64
-mno-fused-madd	-qfloat=nomaf
-mfused-madd	-qfloat=maf
-mlong-double-64	-qnolonglong
-mlong-double-128	-qlonglong
-mpower	-qarch=pwr
-mpowerpc	-qarch=ppc
-mpowerpc64	-qarch=ppc64
-mtune=power	-qtune=pwr
-mtune=powerpc	-qtune=ppc
-mtune=powerpc64	-qtune=ppc64
-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
C -std=c89	-F:c89
C -std=iso9899:1990	-F:c89

オプション・マップ: GNU C および C++ から XL C/C++

GNU C および C++ オプション	XL C/C++ オプション
C -std=iso9899:199409	-F:c89
C -std=c99	-F:c99
C -std=c9x	-F:c99
C -std=iso9899:1999	-F:c99
C -std=iso9899:199x	-F:c99
C -std=gnu89	-qlanglvl=extc89
C -std=gnu99	-qlanglvl=extc99
C -std=gnu9x	-qlanglvl=extc99
C++ -std=c++98	-qlanglvl=strict98
C++ -std=gnu++98	-qlanglvl=extended
-time	-qphsinfo
-trigraphs	-qtrigraph
-U*macro*	-U*macro*
-u	-u
-Wformat	-qformat
-Wuninitialized	-qinfo=ini
-Wunreachable-code	-qinfo=eff
-Wa,*option*	-Wa,*option*
-Wl,*option*	-Wl,*option*
-Wp,*option*	-Wp,*option*
-w	-w
-x assembler	-qsourcetype=assembler
-x c	-qsourcetype=c
-x c++	-qsourcetype=c++
-x none	-qsourcetype=default
-Z	-Z

その他の GNU オプションはすべて無視され、通知メッセージを発行します。

オプション・マッピングの構成

gxlc および gxlc++ ユーティリティーは、構成ファイル `gxlc.cfg` を使用して、GNU C および GNU C++ オプションを XL C/C++ オプションに変換します。`gxlc.cfg` の各項目は、ユーティリティーが GNU C または GNU C++ オプションを XL C/C++ オプションにマップする方法およびその処理方法を記述しています。

項目は、処理命令用のフラグのストリング、GNU C オプションのストリング、および XL C/C++ オプションのストリングから構成されています。 3 つのフィール

ドは、空白で分離する必要があります。項目に最初の 2 つのフィールドのみが含まれ、XL C/C++ オプションのストリングが省略されている場合、2 番目のフィールドの GNU C オプションは、gxlc で認識はされますが無視されます。

文字を使用して、構成ファイルにコメントを挿入することができます。コメントは、独立した行、または項目の最後に記述することができます。

gxlc.cfg の項目には、以下の構文を使用します。

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

以下、説明です:

a
: プレフィックスとして no- を追加し、オプションを無効にします。値は y (yes)、または n (no) のいずれかです。例えば、フラグが y に設定されると、 finline は fno-inline として使用不可となり、項目は以下のようになります。

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

  -fno-inline の場合、gxlc はそれを -qnoinline に変換します。

b
: XL C/C++ オプションに関連した値があることをユーザーに知らせます。値は y (yes)、または n (no) のいずれかです。例えば、オプション -fmyvalue=*n* は -qmyvalue=*n* にマップします。この場合、フラグは y に設定され、項目は以下のようになります。

  ```
  nyn*        "-fmyvalue"            "-qmyvalue"
  ```

  このように指定すると、gxlc および gxlc++ はこれらのオプションに値があることを予期します。

c
: オプションの処理を制御します。値は以下のようになります。

  - n。gcc-option フィールドにリストされているオプションを処理するようにユーザーに指示します。
  - i。gcc-option フィールドにリストされているオプションを無視するようにユーザーに指示します。 gxlc および gxlc++ は、オプションが無視されたというメッセージを生成し、指定されたオプションの処理を継続します。
  - e。gcc-option フィールドにリストされているオプションが検出されると、処理を停止するようにユーザーに指示します。 gxlc および gxlc++ はエラー・メッセージも生成します。

  例えば、gcc オプション -I- はサポートされていないため、gxlc および gxlc++ で無視する必要があります。この場合、フラグを i に設定します。項目は以下のようになります。

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

  gxlc および gxlc++ がこのオプションを入力として検出すると、それを処理せず、警告を生成します。

d
: コンパイラーのタイプに基づいて、gxlc および gxlc++ にオプションを含めるか、または無視させます。値は以下のようになります。

  - c。C のオプションのみを変換するように gxlc および gxlc++ に指示します。

- x。C++ のオプションのみを変換するように gxlc および gxlc++ に指示します。
- *。C および C++ のオプションを変換するように gxlc および gxlc++ に指示します。

例えば、-fwritable-strings は、両方のコンパイラーによってサポートされ、-qnoro にマップします。項目は以下のとおりです。

```
nnn*        "-fwritable-strings"        "-qnoro"
```

"gcc_or_g++_option"
GNU C バージョン 3.3 によってサポートされる gcc または g++ オプションを示すストリングです。このフィールドは必須で、二重引用符で囲む必要があります。

"xlc_or_xlc++_option"
XL C/C++ オプションを示すストリングです。このフィールドはオプションで、指定する場合は二重引用符で囲む必要があります。空のままにした場合、gxlc および gxlc++ はその項目の *gcc_or_g++_option* を無視します。

また、オプションの範囲をマップする項目を作成することが可能です。これは、アスタリスク (*) をワイルドカードとして使用することによって行うことができます。例えば、gcc -D オプションには、ユーザー定義の名前が必須で、オプションの値を取ることができます。これは、以下の一連のオプションを持つことができます。

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

このオプションのバージョン毎に項目を作成する代わりに、以下のように単一の項目を作成します。

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

ここで、アスタリスクは、-D オプションの後に続く任意のストリングに置き換えられます。

逆に、アスタリスクを使用して、指定したオプションの範囲を除外することができます。例えば、gxlc または gxlc++ ですべての -std オプションを無視する場合、項目は以下のようになります。

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

アスタリスクをオプション定義で使用する場合、オプション・フラグ *a* および *b* はこれらの項目に適用できません。

GNU C または GNU C++ オプションで文字 % を使用すると、そのオプションに関連するパラメーターがあることを示します。これは、gxlc または gxlc++ でオプションを無視し、関連するパラメーターも間違いなく無視するために使用されます。例えば、-include オプションはサポートされていません。このオプションにはパラメーターがあります。アプリケーションでは、オプションとパラメーターの両方を無視する必要があります。この場合、項目は以下のようになります。

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

関連参照

- GNU Compiler Collection のオンライン資料は http://gcc.gnu.org/onlinedocs/ にあります。

オプションの要約: C コンパイラー

この章の付録では、タイプ毎にグループ化して、C コンパイラー・オプションの要約を示します。上位のグループには、オプションのサブグループが含まれています。ソース・コードの基本変換用のサブグループに加え、あるサブグループは、特殊なデバッグ情報の追加などの特別なコード処理または制御用のオプションを構成します。別のサブグループは、リンカーおよびライブラリー検索パスの制御に関係しています。パフォーマンスおよび最適化に関連するオプションについては、41 ページの『最適化について』の最後に要約されています。オプションの説明、完全なオプション構文、および各オプションの使用方法については、「*XL C/C++ コンパイラー・リファレンス*」を参照してください。

基本変換

このグループのオプションは、ソース・コードの基本変換に広く適用することができます。コンパイラー・オプションのサブグループは、一般に以下に関係しています。

- 規格への準拠。
- コンパイル・モードまたはコンパイラー・ドライバーの制御。
- コード生成用ソース・コードの操作。
- 特殊な診断の生成。
- コンパイルされたコードの操作。

ソース・コードの基本変換に関連するオプション

規格への準拠	コンパイル・モードまたはコンパイラー・ドライバーの制御
-qgenproto, -qnogenproto -qlanglvl -qlibansi, -qnolibansi	-# -q32 -q64 -qabi_version=n, -qaltivec, -qnoaltivec -F -qpath -qproto、-qnoproto -qsourcetype
プリプロセッサーの制御	
-C -D -E -P -U	
ソース・コードの生成	
-qalloca -qasm, -qnoasm -qattr, -qnoattr -B -C -qcpluscmt, -qnocpluscmt -D -qdbcs, -qnodbcs -qdigraph, -qnodigraph -qdirectstorage, -qnodirectstorage -E -qenablevmx -qfuncsect, -qnofuncsect -qignprag -M	-qmakedep -qmbcs, -qnombcs -qminimaltoc, -qnominimaltoc -P -qsmallstack, -qnosmallstack -qsyntaxonly -t -qtabsize -qtrigraph, -qnotrigraph -U -qutf, -qnoutf -qvrsave, -qnovrsave -W
診断	**コンパイルされたコード**
-qflag -qinfo, -qnoinfo -qmaxerr, -qnomaxerr -qphsinfo, -qnophsinfo -qprint, -qnoprint -qshowinc, -qnoshowinc -qsource, -qnosource -qsuppress, -qnosuppress -V -v -w -qwarn64, -qnowarn64 -qxcall, -qnoxcall	-qbitfields -c -qchars -qdataimported -qdatalocal -qdollar, -qnodollar -o -qprocimported -qproclocal -qprocunknown -S -qstatsym, -qnostatsym -qtbtable -qupconv, -qnoupconv

特殊な診断

このグループに属するオプションは、コンパイル・プロセスに関連した情報を生成および表示するための機能を制御します。

特殊な診断を使用可能にするオプション

診断	
-qflag -qinfo, -qnoinfo -qmaxerr, -qnomaxerr -qphsinfo, -qnophsinfo -qprint, -qnoprint -qshowinc, -qnoshowinc -qsource, -qnosource	-qsuppress, -qnosuppress -V -v -w -qwarn64, -qnowarn64 -qxcall, -qnoxcall

特別な処理および制御

このグループのオプションは、変換処理のきめ細かい制御を提供し、基本変換オプションより一般に適用されません。コンパイラー・オプションのこのグループ内のトピックは、一般に以下に関係しています。

- データ位置合わせ。
- コンパイル・モードまたはコンパイラー・ドライバーの制御。
- コード生成用ソース・コードの操作。
- 特殊な診断の生成。
- コンパイルされたコードの操作。

特別な処理、微調整、およびデバッグのためのオプション

データ位置合わせ	並列化
-qalign -qenum	-qthreaded, -qnothreaded -qtls, -qnotls (RedHat Linux のみ)
浮動小数点および数値機能	
サイズ -qlonglit, -qnolonglit -qlonglong, -qnolonglong	***浮動小数点値の丸め*** -y
単精度値 pSeries プラットフォームの Linux には適用できません。	***その他の浮動小数点オプション*** -qfloat -qflttrap, -qnoflttrap
デバッグ	
-qcheck, -qnocheck -qdbxextra, -qnodbxextra -qfullpath, -qnofullpath -g -qhalt -qinitauto, -qnoinitauto	-qkeepparm, -qnokeepparm -qlinedebug, -qnolinedebug -qlist, -qnolist -qlistopt, -qnolistopt -qsaveopt, -qnosaveopt -qsymtab -qxref, -qnoxref

リンクおよびライブラリー関連オプション

このグループのオプションは、コンパイル処理のリンク・フェーズに関連しています。またこのグループには、ライブラリーおよびヘッダー・ファイルを見つけるための、検索パスを指定する特殊な方法を提供するオプションを含んでいます。これらのコンパイラー・オプションは、一般に以下に関係しています。

- ストリング・リテラルおよび定数の配置。
- 静的および動的リンクおよびライブラリー。
- 検索ディレクトリーの指定。

ld コマンドを制御するためのオプション

ストリング・リテラルおよび定数の配置	静的および動的リンクおよびライブラリー
-qkeyword, -qnokeyword -qro, -qnoro -qroconst, -qnoroconst	-qstdinc, -qnostdinc
検索ディレクトリー	**その他のリンカー・オプション**
-I -L -l (小文字の L) -qc_stdinc -qcomplexgccincl, -qnocomplexgccincl -qgcc_c_stdinc -qidirfirst, -qnoidirfirst -r	-qinlglue, -qnoinglue -qcrt, -qnocrt -qlib, -qnolib

オプションの要約: C++ コンパイラー

C コンパイラー・オプションの多くは、C++ プログラムのコンパイルでも使用可能です。以下のテーブルは、C++ プログラムのコンパイル特有の追加のコンパイラー・オプション、および Linux プラットフォームで C++ プログラムをコンパイルする際に使用できない C オプションを示しています。

C++ プログラム用コンパイラー・オプション

C++ 固有のオプション	C のみのオプション
-+	-qc_stdinc
-qcinc	-qcpluscmt, -qnocpluscmt
-qcpp_stdinc	-qdbxextra, -qnodbxextra
-qeh, -qnoeh	-qgcc_c_stdinc
-qgcc_cpp_stdinc	-qgenproto, -qnogenproto
-qhaltonmsg	-qproto, -qnoproto
-qpriority	-qsyntaxonly
-qrtti, -qnortti	-qupconv, -qnoupconv
-qstaticinline, -qnostaticinline	
-qtempinc, -qnotempinc	
-qtemplaterecompile,	
-qnotemplaterecompile	
-qtemplateregistry,	
-qnotemplateregistry	
-qtempmax	
-qtmplparse	
-qvftable, -qnovftable	

最適化について

単純なコンパイルとは、ソース・コードを実行可能ファイルまたは共有オブジェクトへ変換することです。最適化変換とは、アプリケーションの実行時のパフォーマンス全体を向上させる変換です。 XL C/C++ は、PowerPC アーキテクチャーに合わせて調整された変換を最適化するポートフォリオを提供します。このような変換では、以下を行うことができます。

- クリティカルな操作に対して実行する命令の数を減らす。
- 生成されたオブジェクト・コードを再構成して、PowerPC アーキテクチャーの使用を最適化する。
- メモリー・サブシステムの使用を改善する。
- アーキテクチャーの機能を活用して、大量の共用メモリー並列化を処理する。

これらの目的は、アプリケーションの実行速度を速くすることです。

コードの変換に影響する使用可能な制御を理解し、優れたコードを記述すれば、比較的少ない開発努力で、パフォーマンスを大きく改善することができます。 OpenMP などのプログラミング・モデルを使用することにより、パフォーマンスの高いコードを記述できます。本節では、ランタイム・パフォーマンス、手動でコーディングしたマクロ最適化、一般的な読み易さ、およびソース・コードの全体的な移植性という点に関して、相反するこれらのバランスを取るために、コンパイラーが実行できる最適化の一部について説明します。

この説明では、ユーザーがプロファイラーを使用して、最適化が適切であると思われるコードの領域を識別したと仮定しています。

最適化は、アプリケーション開発サイクルの後半のフェーズ (例えば製品リリース・ビルドなど) でよく試行されます。可能であれば、コードを最適化する前に、まず最適化しない状態でコードをテストおよびデバッグしてください。最適化とは、プログラムに最も効率的なアルゴリズムを選択し、それらを正しくインプリメントすることです。言語標準へ準拠しているかどうかは、コードを正常に最適化できる度合いに直接大きく関係します。最適化プログラムは、最終的な規格合致試験です。

最適化は、コンパイラー・オプション、ディレクティブ、およびプラグマによって制御されます。ただし、コンパイラー・フレンドリー・プログラミングの技法が、オプションやディレクティブと同様に、パフォーマンスに対して有益です。コードを手動で最適化する (例えば手動でループをアンロールする) ことは不要であり、過度に行うことはお勧めできません。異常な構成によって、コンパイラー (およびその他のプログラマー) に混乱を招き、アプリケーションを新しいマシン用に最適化する作業が困難になる場合があります。

最適化は必ずしもすべてのアプリケーションに有益であるとは限りません。デバッグ機能の削減に伴うコンパイル時間の増加と、コンパイラーによって行われる最適化の度合いは、相反する関係にあると言えます。

関連参照

- 「*XL C/C++ プログラミング・ガイド*」の『最適化レベルの使用』
- 「*XL C/C++ プログラミング・ガイド*」の『アプリケーションの最適化』

最適化のための選択可能コンパイラー・オプション

次のテーブルは、プログラム・パフォーマンスを最適化するための基本コンパイラー・オプションの選択を示しています。完全なリストについては、「*XL C/C++ プログラミング・ガイド*」を参照してください。使用可能なサブオプションの資料については、「*XL C/C++ コンパイラー・リファレンス*」を参照するか、またはオプションのマニュアル・ページを参照してください。

表 1. 最適化のための基本コンパイラー・オプション

オプション	説明
-qnoopt	コンパイラーは極めて限られた最適化を実行します。これがデフォルトです。アプリケーションを最適化し始める前に、**-qnoopt** と正しくコンパイルしているか確認してください。
-O2	コンパイラーは包括的な低レベルの最適化を実行しますが、これにはグラフ・カラーリング、共通副次式の除去、不要コードの除去、代数の単純化、定数伝搬、ターゲット・マシンの命令スケジューリング、ループのアンロール、およびソフトウェアのパイプラインを含みます。
-qarch **-qtune** **-qcache**	コンパイラーは、特定のハードウェアおよびアプリケーションが稼働する命令セットの特性を利用します。 **-qarch** を使用して、アプリケーション・コードを生成する対象のプロセッサー・アーキテクチャーのファミリーを指定します。 **-qtune** を使用して、最適化の対象を特定のマイクロプロセッサー上での実行に偏らせます。**-qcache** を使用して、特定のキャッシュまたはメモリー形状を定義します。
-qpdf1 **-qpdf2**	**-O** 以上の最適化レベルで指定したとき、コンパイラーは、プロファイル主導のフィードバックを使用して、異なるコード・セクションが一般に実行される頻度についての分析を基にしてアプリケーションを最適化します。PDF プロセスは、非構造の分岐を含むアプリケーションに最も役立ちます。
-O3	コンパイラーは、**-O2** で、より積極的な最適化を実行します。ループのアンロールが深いほど、ループのスケジューリングが良く、暗黙のメモリー使用に関する限度を除去します。
-qhot	コンパイラーが高位の変換を実行し、さらなるループ最適化を提供し、選択的に配列埋め込みを実行します。このオプションは、大容量の数値処理を実行する科学計算アプリケーションに最も役立ちます。
-qipa	コンパイラーはプロシージャー間分析を実行し、アプリケーション全体を単位として最適化します (プログラム全体の分析)。このオプションは、頻繁に使用される大量のルーチンを含むビジネス・アプリケーションに最も役立ちます。また、高レベルの抽出をする C++ プログラムにも役立ちます。多くの場合、このオプションはコンパイル時間を大幅に増加させます。
-O4	これは、**-O3 -qipa -qhot -qarch=auto -qtune=auto -qcache=auto** と同等です。コンパイルに時間がかかりすぎる場合、 **-O4 -qnoipa** でコンパイルを試行してください。

表 1. 最適化のための基本コンパイラー・オプション (続き)

オプション	説明
-O5	これは、**-O4 -qipa=level=2** と同等です。Linux プラットフォームでは、プロセッサーが PowerPC 970 で、AltiVec データ型がオペレーティング・システムによってサポートされている場合、このオプションは **-qhot=vector -qhot=simd** もオンにします。

移植の考慮事項

本節では、UNIX ベースのアプリケーションを Linux プラットフォームへ簡単に移植するために調査する一般的な項目を記載します。

アプリケーションを他のプラットフォームで実行するように移植する作業では、ソース・プラットフォームとターゲット・プラットフォームが必要です。コーディングを始める前に、最も基本的な段階として、次の質問を考えてください。ターゲット・プラットフォームへ*移植*することで、*何が変更されるのか?* 純粋な「移植」とは、ハードウェアとオペレーティング・システムのみを変更することを指します。

理論的には、プログラムが良い形式で記述されており、プラットフォームに固有な点に依存せず、業界標準 (POSIX など) に従い、標準の言語定義に準拠して標準外の言語拡張を使用しなければ、新たなオペレーティング・システムに容易に移植することができ、再コンパイルとデバッグ以外には最小限の追加作業しか必要としません。ソース・プラットフォームが比較的最近の UNIX ベースのオペレーティング・システムである場合は、変更する点は、業界標準により完全に準拠させることと、同じ業界標準の新しいバージョンに準拠させることのみになることもあります。アプリケーションがすでに Linux システムで実行している場合は、再コンパイルしてネイティブに実行するという選択肢があります。多くのアプリケーションは、変更せずとも再コンパイルして実行することができます。

しかも、異なる標準準拠コンパイラーでアプリケーションをコンパイルすることで、コンパイラー間で言語標準の実装が違うために、ソース・コードのわずかなぜい弱性を取り除くことができる場合があります。その結果、アプリケーションがより堅固になります。

移植を行う際に起こる問題は、内部移植性問題と外部移植性問題に分類することができます。*内部* 移植性問題では、プログラム言語に組み込まれる暗黙の前提事項を処理します。例えば、C プログラムは、整数内での特定のバイト・オーダー、整数の相対サイズのセット、および構造体内の特定のフィールド・レイアウトを前提とします。内部移植性は、ハードウェアに対するプログラム・コードの関係と関連します。この移植性問題のクラスはプログラマーの制御下にあります。

一方、*外部* 移植性問題は、プログラムが使用する外部インターフェース、プログラムが前提とするこれらのインターフェースのセマンティクス、およびプログラムに渡される引き数とプログラムから渡される戻り値の選択と関連しています。これらは、プログラムが依存するライブラリーおよびシステム呼び出し、プログラムが呼び出す (ただし、外部の) コードを取り扱っています。プログラムが標準化された外部インターフェースを使用する場合、外部移植性はプログラマーの制御下に置くことができます。

言語に組み込まれた移植性の問題

このセクションでは、Linux プラットフォームへの移植に関連する内部移植性の考慮事項の一部を示します。

- GNU C およびその他の言語拡張機能への依存の程度を調べる。 ISO 言語仕様に厳密に適合するアプリケーションは、最大限の移植が可能です。 IBM XL C/C++ は、C および C++ への GNU C および C++ 拡張機能のサブセットをサポートします。サポートされない拡張機能に依存するコードを再調査してください。
- ヌル・ポインターがどのように間接参照しているかを確認する。コード内の一部のエラーは、ハードウェア依存の特性によりプラットフォームで検出されません。これらの種類のエラーは、プログラムが他のプラットフォームに移植されると表示されることがあります。ソース・プラットフォームが AIX の場合は、オプション **-qcheck=nullptr** を使用すると、移植前にこのような状態を検出するのに役立ちます。

  AIX では、最低の 4K メモリー (つまり、アドレス 0 ～ 4K-1) が読み取り可能で、ゼロが含まれますが、Linux および Mac OS X プラットフォームでは読み取り不可なので、このようなプラットフォームでアクセスするとセグメンテーション違反になります。例えば、次のような場合です。

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

  この結果、Linux および Mac OS X ではセグメンテーション違反になりますが、AIX では違反になりません。
- C++ オブジェクト・モデルを変更する。Linux プラットフォームでは、GNU C++ オブジェクト・モデルのみがサポートされます。オブジェクト・モデルを指定するために **-qobjmodel** を使用する AIX makefile を移植する場合、そのオプションと、そのオブジェクト・モデルへのその他の参照をすべて削除する必要があります。
- 位置合わせを調べる。 AIX でサポートされる位置合わせのタイプは、同じ名前になる場合がありますが、Linux または Mac OS X プラットフォームでサポートされるものと同じではありません。**-qalign** または `#pragma align` の特定の値に依存するプログラムを移植する場合、プログラムを変更しなければならないことがあります。
- データ構造の移植性を確実にする。あるプラットフォームでアプリケーションを使用してデータを生成し、他のプラットフォームでアプリケーションを使用してデータを読み取ると、データは読み取るアプリケーションが予期するのとは異なる位置合わせを持つことがあります。この問題を回避するために、構造内のデータのレイアウトには必ずプラットフォームに依存しないメカニズムを使用してください。例えば、`#pragma pack(1)` および `#pragma pack(pop)` ペアで構造を囲む場合、位置合わせはすべてのプラットフォームで同じになります。
- Make ファイル内のコマンドを変換するために **gxlc** または **gxlc++** ユーティリティーを使用する。すべての gcc または g++ オプションに対応する XL C/C++ オプションが存在するわけではありません。
- 32 ビットまたは 64 ビット・アプリケーションに異なるコンパイラー呼び出しモードを使用する。
- Linux または Mac OS X では、デフォルトのグローバル演算子 `new` が呼び出され、割り振り要求を実行することができない場合、タイプ `std::bad_alloc` の例外がスローされます。AIX では、グローバル演算子 `new` のデフォルトの振る舞いで、割り振りが失敗した場合にヌル・ポインターが戻されます。
- テンプレートを使用するアプリケーションの移植性を確実にする。 C++ コンパイラーは、ソース・コードのテンプレートを手動で保守する代替手段として、テ

ンプレート・ファイルで機能する 2 つの異なるメソッドを提供します。各メソッドには関連するコンパイラー・オプションがあります。-qtemplateregistry コンパイラー・オプションはすべてのテンプレートのレコードを保守します。このメソッドをお勧めします。また、古いコンパイラー・オプション -qtempinc は、他のプラットフォームから移植するアプリケーションのために提供されます。ただし、Mac OS X プラットフォームでは、コンパイラー・オプション -qtempinc は使用すべきでないと考えられます。

関連参照

以下の IBM Redbooks には、移植に関連する情報が含まれています。他の Redbook は、www.redbooks.ibm.com にてオンラインで参照することができます。
- 「*pSeries と Linux アプリケーション*」(2003 年)

コンパイル時エラーの診断

移植プロジェクトでは、オプション **-qinfo=por** でアプリケーションをコンパイルすることを基本的にお勧めしています。**この -qinfo** サブオプションは、特に移植性に関係する診断メッセージを追加します。 **-qinfo=warn64** オプションは、64 ビット・モードへのアプリケーションの移植に固有の診断メッセージを発行するようコンパイラーに命令します。これらのメッセージは、調査の範囲を狭め、特定のコーディング構成を正確に示すのに役立ちます。

以下のテーブルは、コンパイル時エラーを検出し、修正する役に立つその他のオプションを示しています。

コンパイル時エラーに関するその他の診断オプション

オプション	説明
-qsrcmsg	コンパイラーがエラーを含むと判断したソース行、その下にソース行の特定の箇所を指し示す行、そして診断メッセージを標準エラーに出力する。
-qsource	コンパイラー・リストを戻すことを要求する。リストには、行番号付きのソース・コード、指定されたオプション、コンパイルに使用したすべてのファイルのリスト、診断メッセージの重大度レベル別の要約、読み込まれた行数、およびコンパイルが成功したかどうかなどのセクションが含まれます。**-qattr** および **-qxref** オプションを指定することでそれぞれリストの属性セクションと相互参照セクションを作成することができます。オブジェクト・セクションを作成するには、オプション **-qlist** を指定する必要があります。このセクションには、コンパイラーの生成する疑似アセンブリー・コードが表示され、診断実行時に問題が発生しおり、コード生成エラーによってプログラムが期待通りのパフォーマンスを示していないことが疑われる場合に使用します。
-qsuppress	コンパイラーが特定のメッセージを出力しないようにする。メッセージ番号をコロン区切りでリストすることで、複数のメッセージを抑止できます。

コンパイル時エラーに関するその他の診断オプション

オプション	説明
-qflag	指定した診断メッセージが端末およびリスト・ファイルに出力されないようにする。このオプションで、どのレベル以下のメッセージを無視するかを指定するには、デフォルトのコンパイラー・メッセージ形式で使用される単一文字による重大度コードを使用します。

32 ビットおよび 64 ビット・アプリケーションの開発

XL C/C++ を使用して 32 ビットと 64 ビット・アプリケーションの両方を開発できます。このセクションでは、参照情報と、C および C++ プログラムを 32 ビットから 64 ビット・モードに移行する上での、移植性に関するその他の考慮事項を説明します。

移植を行う際、変更点がハードウェアとオペレーティング・システムのみであるなら、それは純粋な「移植」と言えます。ただし、Linux に移行する過程で、32 ビット・アプリケーションを 64 ビット・プログラミング・モデルに変更する場合は、この行いはもはや純粋な「移植」ではなく、ある種の開発活動であることを意味します。以下の特徴のいずれかを有する 32 ビット・アプリケーションは、64 ビット環境に移植する際に変更を必要とする可能性が非常に高いです。

- カーネルのメモリーを直接読み取り、解釈している。
- /proc ファイル・システムを使用して 64 ビット・プロセスにアクセスしている。
- 64 ビット・バージョンにしかないライブラリーを使用している。
- デバイス・ドライバーである。
- Linux との間にインターオペラビリティー上の問題があるソース・プラットフォームからの移植である。

64 ビット・モードで開発すると、アプリケーションはより新しくて速い 64 ビット・ハードウェアとオペレーティング・システムを利用することが可能で、データベース・アプリケーションや科学計算アプリケーションなどの、大規模で複雑なメモリーを大量に使用するプログラムのパフォーマンスを向上することができます。データベース・アプリケーション、Web 検索エンジン、科学計算アプリケーションなどのアプリケーションで、32 ビットのアドレス・スペースによりパフォーマンスに制限が課せられているものは、多くの場合、64 ビット・モードに移行することで利便性が向上します。入出力によりパフォーマンスに制限が課せられているアプリケーションも、64 ビット・モードにすることで、データをディスクに書き込まずにメモリーに保持することができるため、より高いパフォーマンスを実現できます。これは、通常メモリー・アクセスの方が、ディスク I/O よりも速いためです。

64 ビット・マシンを使用することで、パフォーマンスに最も大きく貢献する点は、おそらく大規模な問題を物理メモリー内で直接処理できるようになる点でしょう。しかし、いくつかのアプリケーションは、64 ビット・モードで再コンパイルしたときよりも、32 ビット・モードでコンパイルした方が良いパフォーマンスを示します。この理由としては、以下のものが存在します。

- 64 ビット・プログラムの方が大きい。プログラム・サイズの増加により、物理メモリーへの負荷がより大きくなります。
- 64 ビットの long 型除法の方が、32 ビットの整数除法よりも時間がかかる。

- 64 ビット・プログラムで、配列指標に 32 ビットの符号付き整数を使用する場合は、配列を参照するたびに、符号拡張を行うための追加の命令が必要となる場合がある。

64 ビット・アプリケーションの他の欠点としては、より大きいレジスターを保持するためにより多くのスタック・スペースを必要とする点があります。ポインター・サイズが大きくなるため、アプリケーションのキャッシュ・フットプリントが大きくなります。 64 ビット・アプリケーションは、32 ビット・プラットフォーム上では実行できません。

64 ビット・プログラムのパフォーマンス上のマイナス影響を補正するには、次の方法があります。

- 32 ビットと 64 ビット混合の演算処理を避ける。例えば、32 ビットの符号付きデータ型と 64 ビットのデータ型を加算すると、32 ビット型の方を符号拡張し、符号によって、レジスターの上位 32 ビットを設定する必要があります。レジスターの上位ビットを設定すると、計算が遅くなります。 32 ビット型が符号なしの場合、レジスターの上位ビットはクリアされます。
- 64 ビットの long 型の除法を可能な限り避ける。乗算は、多くの場合、除法よりも高速です。同じ除数を使って多くの除法を実行する必要のある場合は、除数の逆数を一時変数に割り当て、すべての除法を一時変数に対する乗算に変更します。例えば、次の関数では、

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

  とした方が、以下の直接除法を行った場合よりも、実行が速くなります。

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

  その理由は、除法 (1.0 / 1.0825) は、定数フォールディングにより、コンパイル時に一度のみ評価されるからです。この最適化は通常、**-qnostrict** オプションに効力がある場合 (**-O2** 以上の最適化レベルでコンパイルすると起こります)、コンパイラーによって行われます。
- 配列指標には signed、unsigned、およびプレーンの **int** 型ではなく、**long** 変数を使用する。これにより、コンパイラーは配列を参照するときに符号拡張を行わずに済みます。

良い形式で書かれたコードは、64 ビット・プログラミング・モデルに移行すれば最小限の修正作業とデバッグで正常にコンパイルおよび実行できる可能性が高いです。「*良い形式*」という表現は、言語標準に準拠し、良いプログラミング慣習に従っていることを意味します。 64 ビット・プログラミング・モデルへ移行することに当てはめていうと、「良い形式」とは、コードが特定のバイト・オーダーや外部データ・フォーマットに依存していないこと、および関数プロトタイプ、適切なシステム・ヘッダー・ファイル、およびシステム派生のデータ型を使用していることも含まれます。

実行時エラーの診断

プログラムが正常にコンパイルおよびリンクしても、実行時に予期しない結果を起こす場合があります。コンパイラーは、言語の構文規則に違反しないプログラミング・エラーは診断しません。このセクションでは、一般的なエラーとその発見方法、および訂正方法を説明しています。

初期化されていない変数

自動ストレージ期間を持つオブジェクトは、暗黙的に初期化されないため、初期値は未定です。 **auto** 変数が値を設定される前に使用されると、プログラムの実行のたび、同じ結果を生成するかどうかが一定していません。コンパイラー・オプション **-qinfo=gen** を指定すると、値が設定される前に使用されている **auto** 変数の位置を表示します。 **-qinitauto** オプションは、すべての自動変数を指定した値で初期化するようにコンパイラーに指示します。このオプションを使用すると、アプリケーションの実行時のパフォーマンスが落ちるため、デバッグ時にのみ使用することをお勧めします。

実行時検査

-qcheck オプションを指定すると、実行可能ファイルに実行時検査のコードを挿入します。サブオプションで、NULL ポインター、配列の範囲を超える指標付け、およびゼロによる除法を検査するように指定します。 **-qinitauto** と同様に、**-qcheck** もアプリケーションのパフォーマンスが低下するため、デバッグ目的でのみ使用することが推奨されます。

ANSI 別名割り当て

型ベースの別名割り当て (「ANSI 別名割り当て」とも呼ばれます) では、データ・オブジェクトに安全にアクセスするために使用できる左辺値を制限します。言語標準への準拠を強制する言語レベルでコンパイルすると、C および C++ コンパイラーは、最適化を行うときに型ベースの別名割り当てを必ず実行します。 ANSI の別名割り当て規則では、ポインターは、同じ型のオブジェクトか、互換性のある型のオブジェクトに対してのみ間接参照することができると規定されています。この規則の例外は、符号および型の修飾子には型ベースの別名割り当ては適応されないこと、および文字型ポインターはどのタイプも指し示すことができるということです。よく使用されるコーディング方法として、ポインターを非互換の型にキャストしてから間接参照することがありますが、これはこの規則に違反しています。

-qalias=noansi を設定して、ANSI 別名割り当てをオフにすると、プログラムの動作は訂正できるかもしれませんが、コンパイラーがアプリケーションを最適化できる機会を減らし、実行時のパフォーマンスが劣化します。推奨される解決策は、プログラムを修正して型ベースの別名割り当て規則に合わせることです。

#pragma option_override

最適化を使用している場合にのみエラーが表示される場合があります。複雑なアプリケーションでは、プログラミング・エラーを含むことが分かっている関数に対して最適化をオフにし、プログラムの残りの箇所では最適化をオンにすると、特に有益な場合があります。 `#pragma option_override` ディレクティブを使用して、特定の関数に別の最適化オプションを指定することができます。

`#pragma option_override` ディレクティブは、エラーの原因となっている関数を判別するために使用することもできます。問題の関数を見つけるには、エラーが出なくなるまで、ディレクティブ内でそれぞれの関数の最適化を順番にオフにしていきます。

共用メモリーの並列処理

プログラムの並列実行を実現し、シングル・プロセッサーで実行するより高速でジョブを完了するための、証明された手法がいくつかあります。これらの手法には、以下のものがあります。

- ディレクティブ・ベースの共用メモリー並列処理 (SMP)
- コンパイラーへの共用メモリー並列処理の自動生成命令
- メッセージ引き渡しベースの共用または分散メモリー並列処理 (MPI)
- POSIX スレッド (pthread) 並列処理
- `fork()` および `exec()` を使用した下位 UNIX 並列処理

アプリケーションの移植性要件は、明らかに、使用に最適な技法を選択する際の要因です。この選択はまた、アプリケーション、プログラマーのスキルおよび設定、およびターゲット・マシンの特性に非常に依存します。 AIX での並列処理を達成する 2 つの主要な方法は、手動でコーディングした POSIX スレッド (pthread) の使用と OpenMP ディレクティブの使用です。

AIX オペレーティング・システムの並列プログラミング機能はスレッドの概念に基づいています。並列プログラミングはマルチプロセッサー・システムの利点を活用しながら、既存の単一プロセッサー・システムで完全なバイナリー互換性を維持しています。これは、単一プロセッサー・システムで作業するマルチスレッドのプログラムは、再コンパイルなしでマルチプロセッサー・システムを利用することができることを意味します。

pthread は、並列処理プロセスの最大限の制御を提供するため、複数のプロセッサーを効率的に使用する優れた柔軟性が実現します。そのトレードオフとして、コードが大幅に複雑化します。多くの場合、OpenMP ディレクティブ、 SMP 対応ライブラリー、またはコンパイラーの自動 SMP 機能の使用といった簡単な方法が推奨されます。スレッドを明示的に使用しても、必ずしもパフォーマンスが向上するわけではありません。マルチスレッドのアプリケーションをデバッグすると、どうしても問題が起こりがちです。ただし、一部のプログラムでは、これのみが実行可能な方法です。

以下に各技法の利点と欠点の一部をリストします。

コンパイラーによる自動並列処理

- 簡単なインプリメント (`-qsmp=auto` とコンパイル)。
- チームでの作業が容易。
- データ・スコープが無視されることによる限られたスケーラビリティー。
- コンパイラー依存 (特定のコンパイラーのリリースの場合でも)。
- 必ずしも移植可能ではない。

SMP 対応ライブラリー

- 作業がほとんど必要ない。
- 必ずしも移植可能ではない (通常専有)。
- 限られた柔軟性。

OpenMP ディレクティブ

- 移植可能。
- 自動並列処理において潜在的により優れたスケーラビリティー。
- ユニフォーム・メモリー・アクセスを前提とする。

ハイブリッド・アプローチ (OpenMP および pthread のサブセットまたは混合、または UNIX `fork()` および `exec()` 並列処理、プラットフォーム固有の構文)
- チームワークを可能にすることがある。
- パフォーマンスと移植性を確実にするため、十分に検証された概念が必要。
- 必ずしも移植可能ではない。

pthread
- 移植可能。
- 並列処理プロセスに対する最大限の制御を提供する。
- 自動並列処理の潜在的に最適なスケーラビリティー
- 複雑なコード化を処理するのに経験のあるプログラマーが必要。

OpenMP ディレクティブ

OpenMP ディレクティブは、特定のループをどのように並列処理するかをコンパイラーに命令するコマンド・セットです。ソースにディレクティブがあると、コンパイラーが並列コードで並列分析を実行する必要がなくなります。 OpenMP ディレクティブを使用するには、並列処理に必要なインフラストラクチャーを提供する pthread ライブラリーが必要です。

OpenMP ディレクティブは、アプリケーションの並列処理に重要な 3 つの問題に取り組みます。まず、文節およびディレクティブはスコープ変数に使用可能です。ほとんどの場合、変数を共有するべきではありません。つまり、プロセッサーはそれぞれ、それ自身の変数のコピーを持っている必要があります。第 2 に、作業共用ディレクティブは、コードの並列領域に含まれる作業を複数の SMP プロセッサー間で分散させる方法を指定します。最後に、プロセッサー間で同期用のディレクティブがあります。

コンパイラーは OpenMP バージョン 2.0 仕様をサポートします。

関連参照
- 61 ページの『付録 B. OpenMP 準拠とサポート』

GNU C および C++ の移植性に関連したフィーチャー

GNU C で開発されるアプリケーションまたはコードの移植を容易にするために、XL C/C++ は C99 と標準 C++ に対する GNU C および C++ 言語拡張機能のサブセットをサポートします。このセクションの表では、サポートされるフィーチャー、サポートされないフィーチャー、および構文は受け入れられるがセマンティクスは無視されるフィーチャーがリストされます。

C コードでサポートされる 拡張機能を使用するために、 **xlc** または **cc** 呼び出しコマンドを使用するか、または `-qlanglvl=extc89`、`-qlanglvl=extc99`、または

-qlanglvl=extended のうち 1 つを指定してください。デフォルトでは、C++ のサポートされているすべての GNU C および C++ フィーチャーが受け入れられます。

以下の表では *accept/ignore* とマークされた拡張機能は、受け入れ可能なプログラミング・キーワードとしてコンパイラーに認識はされますが、GNU C/C++ セマンティクスはサポートされていません。これは、コンパイラーが *accept/ignore* キーワードまたは拡張機能を検出する場合、コンパイルは停止せず、GNU セマンティクスはアプリケーションでインプリメントされないということです。厳密な言語レベル (stdc89、stdc99) の下でこれらの拡張機能を使用するソース・コードをコンパイルすると、エラー・メッセージが出されます。

関連参照

GNU C および C++ 言語拡張機能は、http://gcc.gnu.org/onlinedocs の GNU マニュアルで完全に文書化されています。

関数属性

関数の宣言または定義を行う際に、特殊な属性を指定するには、キーワード **__attribute__** を使用します。このキーワードには、二重括弧内に属性仕様が続きます。 XL C/C++ は、GNU C および C++ の関数属性のサブセットをサポートします。 *accept/ignore* と記述された振る舞いの意味は、構文は受け入れられるものの、セマンティクスは無視され、コンパイルが続行するということです。

GNU C/C++ の XL C/C++ との関数属性の互換性

関数属性	動作
alias	supported
always_inline	supported
cdecl	accept/ignore
const	supported
constructor	supported
destructor	supported
dllexport	accept/ignore
dllimport	accept/ignore
eightbit_data	accept/ignore
exception	accept/ignore
format	supported
format_arg	supported
function_vector	accept/ignore
interrupt	accept/ignore
interrupt_handler	accept/ignore
longcall	accept/ignore
model	accept/ignore
no_check_memory_usage	accept/ignore
no_instrument_function	accept/ignore
noinline	supported

GNU C/C++ の XL C/C++ との関数属性の互換性

関数属性	動作
noreturn	supported
pure	supported
regparm	accept/ignore
section	supported
stdcall	accept/ignore
tiny_data	accept/ignore
weak	supported

関連参照
- 「*XL C/C++ ランゲージ・リファレンス*」の『関数属性』

変数属性

キーワード **__attribute__** を使用して、変数または構造化フィールドの特殊な属性を指定します。このキーワードには、二重括弧内に属性仕様が続きます。 XL C/C++ は、GNU C および C++ の変数属性のサブセットをサポートします。 *accept/ignore* と記述された振る舞いの意味は、構文は受け入れられるものの、セマンティクスは無視され、コンパイルが続行するということです。

GNU C/C++ の XL C/C++ との変数属性の互換性

変数属性	動作
aligned	supported
C++ init_priority	supported
mode	supported
model	accept/ignore
nocommon	supported
packed	supported
section	supported
transparent_union	supported
unused	accept/ignore
weak	supported

関連参照
- 「*XL C/C++ ランゲージ・リファレンス*」の『変数属性』

型属性

キーワード **__attribute__** を使用して、struct および union 型を定義する際に、それらの特殊な属性を指定します。このキーワードには、二重括弧内に属性仕様が続きます。 XL C/C++ は、GNU C および C++ の型属性のサブセットをサポートします。 *accept/ignore* と記述された振る舞いの意味は、構文は受け入れられるものの、セマンティクスは無視され、コンパイルが続行するということです。

GNU C/C++ の XL C/C++ との型属性の互換性

型属性	動作
aligned	supported
packed	supported
transparent_union	サポート
unused	accept/ignore

関連参照

- 「*XL C/C++ ランゲージ・リファレンス*」の『型属性』

GNU C および C++ アサーション

アサーション を使用して、コンパイル済みプログラムが実行するコンピューターまたはシステムの種類をテストします。アサーション #cpu、#machine、および #system が事前定義されています。また、プリプロセス・ディレクティブ #assert と #unassert を使用してもアサーションを定義できます。

XL C/C++ での GNU C および C++ アサーション

GNU C アサーション	動作
#assert	supported
#unassert	supported
#cpu	supported 可能な値は powerpc
#machine	supported 可能な値は powerpc と bigendian
#system	supported 可能な値は unix と posix

その他の GNU C および C++ の移植性に関する問題

GNU C および C++ に関連する以下のフィーチャーは、拡張言語レベル (extc89、extc99、extended) の下でサポートされます。

- ディレクティブ #warning を使用して、プリプロセッサーに警告を出させ、処理を継続させます。
- ディレクティブ #include_next を使用して、ディレクトリーに現行のヘッダー・ファイルの後に次のヘッダー・ファイルを組み込むことを指定します。
- ローカル・ラベルは、それぞれのステートメント式の開始点で宣言できます。
- 小括弧内で中括弧で囲まれた複合ステートメントを式として使用します。
- **__typeof__** キーワードで式のタイプを参照します。
- 複合式、条件式、および lvalues としてのキャストを使用します。
- 計算された **goto** 文を使用して、ラベルにジャンプします。そのアドレスは取り出され、アドレスは値として使用されます。
- キーワード **__alignof__** を使用して、変数位置合わせまたは、型ごとに通常必要な位置合わせについて照会します。

- これらのキーワードの代替スペルを使用します:**__asm__**、**__const__**、**__volatile__**、**__inline__**、**__signed__**、および **__typeof__**。
- 厳密な言語レベル・モードで拡張言語フィーチャーを使用する場合、**__extension__** キーワードを使用して、エラーを回避します。
- ゼロの長さの配列が、エラーなしで生成されることがあります。
- **extern** キーワードを使用することにより、テンプレートのインスタンス化のフォワード宣言が許可されます。
- C 他の関数 (ネストされた関数) の定義内に現れる関数定義が許可されます。
- C 共用体のメンバーは、属する共用体型にキャストされることがあります。

拡張言語レベル (extc89、extc99、extended) で、 XL C/C++ は以下のフィーチャーの構文を認識しますが、それらのセマンティクスはサポートされていません。

- C++ レジスター変数の宣言は、グローバルであってもローカルであっても、優先されるレジスターを示すことができます。

Standard C++ Numerics Library ヘッダー・ファイル

XL C++ ヘッダー・ファイル <cmath> および <cstdlib> にある数学関数のすべてが、整数型の引き数を処理できるわけではないという点に、注意する必要があります。その理由は、整数型から浮動小数点型への引き数の型変換により、等しいランクの複数の多重定義解決が作成されるからです。

たとえば、C++ プログラムで **unsigned int** または **unsigned long** の絶対値をとると、XL C++ ではそのヘッダーに abs() ライブラリー関数のための必要なすべてのシグニチャーがインプリメントされるにもかかわらず、その結果は多重定義解決エラーになります。

```
int abs(int);
long abs(long);
float abs(float);
double abs(double);
long double abs(long double);
```

この問題は、XL ヘッダー <math.h> および <stdlib.h> を両方ともインクルードしても、残ります。この問題に対する予備手段は、関数で使用される前に、符号なし整数型の引き数を適切な型にキャストしておくことです。

付録 A. 言語サポート

この付録では、C および C++ プログラム言語のインプリメンテーションと、XL C/C++ によって提供される言語拡張機能について説明します。

ISO/IEC 国際規格との互換性

XL C/C++ を使用すると、移植性を重視したプログラミングを行うことができます。プログラム言語の完全な仕様は、構文とセマンティクスから構成されますが、インプリメンテーション時に言語拡張機能が使用されるため、特定言語仕様の準拠するインプリメンテーションはそれぞれ異なる可能性があります。

言語仕様に厳密に準拠するプログラムは、異なる環境間で最大の移植性を発揮します。理論上、標準に準拠した、あるコンパイラーで正しくコンパイルされ、拡張機能またはインプリメンテーション定義動作を使用しないプログラムは、ハードウェアの差異が許す範囲内で、他のすべての標準準拠のコンパイラーの下で適切にコンパイルされ、作動します。言語インプリメンテーションによって提供される言語への拡張機能を正しく活用したプログラムは、そのオブジェクト・コードの効率性を向上させることができます。

ISO/IEC 14882:2003(E) 国際規格の互換性

XL C/C++ は、ISO/IEC 国際規格 14882:2003(E) と整合しています。この規格は、書式を指定し、C++ プログラム言語で作成されたプログラムの解釈を確立するものです。この国際規格は、さまざまなインプリメンテーション間で、C++ プログラムの移植性を促進するために設計されています。 ISO/IEC 14882:1998 は最初の C++ 言語でした。

ISO/IEC 9899:1990 国際規格の互換性

ISO/IEC 9899:1990 国際規格 (C89 として知られる) では、書式を指定し、C プログラミング言語で作成されたプログラムの解釈を確立しています。この規格は、さまざまなインプリメンテーション間で、C プログラムの移植性をプロモートするために設計されています。この規格は、ISO/IEC 9899/COR1:1994、ISO/IEC 9899/AMD1:1995、および ISO/IEC 9899/COR2:1996 で修正されました。修正および訂正された C89 規格をソース・コードが厳密に順守しているかを確認するには、コンパイラー・オプション `-qlanglvl=stdc89` を指定してください。

ISO/IEC 9899:1999 国際規格サポート

ISO/IEC 9899:1999 国際規格 (C99 としても知られる) は C プログラミング言語で作成されたプログラム用の、更新された規格です。これは、C 言語の機能を拡張し、C89 へ説明を提供し、技術修正を具体化するために設計されています。 XL C/C++ は、この言語規格の多くのフィーチャーをサポートします。

C C コンパイラーは、C99 規格で指定されるすべての言語フィーチャーをサポートしています。この言語フィーチャーのセットをソース・コードが順守してい

るかを確認するには、**c99** 呼び出しコマンドを使用してください。また、この規格では、ランタイム・ライブラリーのフィーチャーも指定していることに注意してください。これらのフィーチャーは、現行のランタイム・ライブラリーおよびオペレーティング環境ではサポートされないことがあります。システム・ヘッダー・ファイルが使用可能かどうかにより、これらのフィーチャーがサポートされているかどうかが分かります。

C99 での主なフィーチャー

XL C/C++ はすべての C99 言語フィーチャーをインプリメントしています。以下は、選択した主なフィーチャーの表です。関連資料は、「*XL C/C++ ランゲージ・リファレンス*」の項目を参照しています。

IBM C に対する ISO/IEC 9899:1999 国際規格の拡張

C99 フィーチャー	関連参照
ポインター用の restrict 型修飾子	restrict 型修飾子
汎用文字名	ユニコード規格
定義済み ID __func__	事前定義済み ID
変数と空引き数を持つ関数類似マクロ	関数類似マクロ
_Pragma 単項演算子	_Pragma 演算子
可変長配列	可変長配列
配列指標宣言内での static キーワード	配列
complex データ型	複素数型
long long int と unsigned long long int 型	整変数
16 進浮動小数点定数	16 進浮動小数点定数
集合体型の複合リテラル	複合リテラル
指定された初期化指定子	初期化指定子
C++ 形式のコメント	コメント
許可されない暗黙の関数宣言	関数宣言
混合宣言とコード	for 文
_Bool 型	単純型指定子
inline 関数宣言	インライン関数
集合体の初期化指定子	指定された初期化指定子を使用した配列の初期化

C99 でサポートされる C89 の変更点と説明

C99 規格の一部の仕様は、言語の新しいフィーチャーに基づくものではなく、C89 規格を変更し、より明確にしたものです。XL C/C++ は、以下を含むすべての C99 言語フィーチャーをサポートしています。

- 配列メンバーを柔軟に使用できます。複数のメンバーを持つ構成の最後のメンバーは、サイズを指定せずに宣言できます。
- 暗黙の int 宣言はサポートされません。すべての宣言に型指定子が必要です。
- 列挙型指定子で、末尾のコンマが使用できます。

- 重複する型修飾子は、他に明示的な指定がなければ、受け取られて無視されます。
- return ステートメントから、必須の式が欠落している場合は、診断メッセージが出されます。
- プリプロセス中に評価されていた定数式は、long long と unsigned long long データ型を使用するようになりました。
- 空のマクロ引き数が関数類似マクロ内で使用できます。
- #line の最大値が 2 147 483 647 に増加しました。

XL C/C++ での C99 フィーチャー

ISO/IEC 9899:1999 国際規格 (C99) のいくつかのフィーチャーは、C++ でもインプリメントされます。これらの拡張は -qlanglvl=extended コンパイラー・オプションを指定すると有効になります。

IBM C++ に対する ISO/IEC 9899:1999 国際規格の拡張

C99 フィーチャー	参照
ポインター用の restrict 型修飾子	restrict 型修飾子
汎用文字名	ユニコード規格
定義済み ID __func__	事前定義済み ID
可変長配列	可変長配列
complex データ型	複素数型
16 進浮動小数点定数	16 進浮動小数点定数
集合体型の複合リテラル	複合リテラル
変数と空引き数を持つ関数類似マクロ	関数類似マクロ
_Pragma 単項演算子	_Pragma 演算子

拡張言語レベル・サポート

-qlanglvl コンパイラー・オプションは、サポートしている言語レベルを指定するために使用され、そのためユーザーのコードがコンパイルされる方法に影響を与えます。別のコンパイラー呼び出しコマンドを使用して、暗黙的に言語レベルを指定することもできます。一般に、標準言語レベルの下で正しくコンパイルされ、実行される有効なプログラムは使用可能な直交拡張機能を使用して同じ結果を生成するために、正しくコンパイルし、実行されるはずです。

例えば、ISO/IEC 9899:1990 国際規格 (C89) に厳密に準拠する方法で C プログラムをコンパイルするには、-qlanglvl=stdc89 を指定する必要があります。stdc89 サブオプションは、コンパイラーが規格に厳密に準拠し、どの言語拡張機能の使用も許可しないことを指示します。(**c89** コンパイラー呼び出しコマンドは、この言語レベルを暗黙的に指定します。)

標準言語レベルに拡張機能を使用することもできます。標準フィーチャーに干渉しない拡張機能は*直交* 拡張機能と呼ばれます。例えば、C プログラムをコンパイルする場合、-qlanglvl=extc89 を指定して、C89 に直交する拡張機能を使用可能にできます。

ISO/IEC 9899:1999 国際規格 (C99) で説明されている言語フィーチャーのほとんどは、C89 への直交拡張機能と見なされます。

一方では、*直交以外の* 拡張機能は、国際規格の 1 つに記載されているように、言語の観点から干渉され、競合することがあります。これらの拡張機能は、明示的に特定のコンパイラー・オプションで使用可能にしなければなりません。非直交拡張機能に依存すると、異なる環境へのアプリケーションの移植が容易にできなくなります。

`-qlanglvl` オプションの主なサブオプションは、以下にリストされています。

選択した -qlanglvl サブオプション

-qlanglvl サブオプション	**サブオプションの説明**
`-qlanglvl=stdc99`	C99 標準への厳密な合致を指定します。
`-qlanglvl=stdc89`	C89 標準への厳密な合致を指定します。
`-qlanglvl=extc99`	C99 に直交するすべての拡張機能を使用可能にします。
`-qlanglvl=extc89`	C89 に直交するすべての拡張機能を使用可能にします。
`-qlanglvl=extended`	C89 に直交するすべての拡張機能を使用可能にし、`-qupconv` コンパイラー・オプションを指定します。

付録 B. OpenMP 準拠とサポート

OpenMP アプリケーション・プログラム・インターフェース (API) は、移植可能で拡張が容易なプログラミング・モデルであり、マルチプラットフォームでメモリーを共用する並列アプリケーションを C、C++、および Fortran で開発するための標準インターフェースを提供します。 OpenMP API の仕様は、OpenMP 組織という、IBM を含む主なコンピューター・ハードウェアおよびソフトウェア・ベンダーの集まりによって定義されます。

XL C/C++ は、OpenMP 仕様 2.0 に準拠しています。コンパイラーは、以下の OpenMP V2.0 エレメントのセマンティクスを認識し、保持します。

- `#pragma omp` ディレクティブ内の複数文節のコンマ区切り文字。
- `num_threads` 文節。
- `copyprivate` 文節。
- `threadprivate` 静的ブロック・スコープ変数。
- C99 可変長配列のサポート
- private 変数の冗長な宣言。
- タイミング・ルーチン `omp_get_wtick` および `omp_get_wtime`。

以下に説明するディレクティブ、ライブラリー関数、および環境変数を使用すると、移植性を維持しながら並列プログラムを作成し管理できます。

OpenMP 並列処理を使用できるようにするには、**-qsmp** コンパイラー・オプションを指定してください。

- 自動並列処理を選択するには、**-qsmp** または **-qsmp=auto** を指定します。このサブオプションを使用すると、コンパイラーはプログラム内のすべての OpenMP ディレクティブ、ライブラリー関数、および環境変数を認識しインプリメントすることに加え、*implicit parallelism* を実行できます。
- OpenMP 仕様 2.0 への厳格な準拠を選択する場合は、**-qsmp=omp** を指定します。このサブオプションを使用すると、コンパイラーはコードに指定された OpenMP ディレクティブ、ライブラリー関数、および環境変数のみをインプリメントします。他の自動化された並列処理は行いません。

関連参照

- http://www.openmp.org
- 「*XL C/C++ コンパイラー・リファレンス*」の『並列処理を制御するプラグマ』
- 「*XL C/C++ コンパイラー・リファレンス*」の『プログラムの並列化』

OpenMP ディレクティブ

それぞれのディレクティブは #pragma omp で始まり、他のプラグマ・ディレクティブとの潜在的な競合を削減します。

XL C/C++ での OpenMP ディレクティブ

ディレクティブ名	ディレクティブ説明
parallel	parallel ディレクティブは、*並列領域* を定義します。これはマルチスレッドによって並列して実行されるプログラムの領域です。
for	for ディレクティブは、反復作業共用構成を識別します。これは、関連するループの反復が並列して実行される必要のある領域を指定します。 for ループの反復は既存のスレッド間で分散されます。
sections	sections ディレクティブは反復しない作業共用構成を識別します。これはチーム内のスレッド間で分割される構成のセットを指定します。それぞれのセクションはチーム内のスレッドで 1 度実行されます。
single	single ディレクティブは構成を識別します。これは関連する構造化ブロックがチーム内のただ 1 つのスレッドでのみ実行されることを指定します (マスター・スレッドである必要はありません)。
parallel for	parallel for ディレクティブは、単一 for ディレクティブを含む並列領域用のショートカット形式です。セマンティクスは、for ディレクティブのすぐ前に parallel ディレクティブを明示的に指定することと同じです。
parallel sections	parallel sections ディレクティブは、単一 sections ディレクティブを含んでいる並列セクションを指定するためのショートカット形式を提供します。セマンティクスは、sections ディレクティブのすぐ前に parallel ディレクティブを明示的に指定することと同じです。
master	master ディレクティブは、チームのマスター・スレッドによって実行される構造化ブロックを指定する構成を識別します。
critical	critical ディレクティブは、関連した構造化ブロックの実行を行えるのは一時点では 1 つのスレッドのみに限定する構成を識別します。クリティカル領域を識別するために、オプションの*名前* を使用することもできます。スレッドは、同じ名前で他のスレッドがクリティカル領域を実行しなくなるまで、クリティカル領域の開始で待ちます。すべての名前なし critical ディレクティブは、同じ未指定の名前にマップします。
barrier	barrier ディレクティブは、チーム内のすべてのスレッドを同期化します。スレッドがこのディレクティブに遭遇すると、それぞれのスレッドは他のスレッドがこのポイントに達するまで待ちます。すべてのスレッドがバリアに遭遇したら、各スレッドは barrier ディレクティブ後でステートメントの実行を並列に開始します。

XL C/C++ での OpenMP ディレクティブ

ディレクティブ名	ディレクティブ説明
atomic	atomic ディレクティブは、自動的に更新されなければならず、複数の同時書き込みスレッドに公開されない特定のメモリー・ロケーションを識別します。
flush	flush ディレクティブは、コンパイラーによって並列領域のすべてのスレッドがメモリー内の指定されたオブジェクトの同じビューを持つようにするポイントを識別します。
ordered	ordered ディレクティブは、順序付けられた順で実行しなければならない構造化されたブロックのコードを識別します。
threadprivate	threadprivate ディレクティブは、スレッドに private になるファイル・スコープ、名前スペース・スコープ、または静的ブロック・スコープ変数を宣言します。

OpenMP データ・スコープ属性文節

文節は、並列または作業共用構成の間に変数のスコープ属性を制御するために、ディレクティブで指定できます。

XL C/C++ での OpenMP データ・スコープ属性文節

データ・スコープ属性文節名	データ・スコープ属性文節記述
private	private 文節は、リスト内の変数がチーム内のそれぞれのスレッドに対して private であることを宣言します。
firstprivate	firstprivate 文節は、 private 文節によって提供される機能のスーパーセットを提供します。
lastprivate	lastprivate 文節は、 private 文節によって提供される機能のスーパーセットを提供します。
copyprivate	copyprivate 文節は、チームに値をブロードキャストする共用変数を使用する代替手段を提供します。このメカニズムでは、private 変数を使用して、あるチーム・メンバーから他のメンバーに値をブロードキャストします。
num_threads	num_threads 文節は、並列構成でスレッドの特定の数を要求する機能を提供します。
shared	shared 文節は、チーム内のすべてのスレッド間でリストに表示される変数を共用します。チーム内のすべてのスレッドは、shared 変数用に同じストレージ域を使用できます。
reduction	reduction 文節は、指定された演算子で、リスト内に表示されたスカラー変数の縮小を実行します。
default	default 文節は、変数のデータ・スコープ属性をユーザーが操作できるようにします。

OpenMP ライブラリー関数

OpenMP ランタイム・ライブラリー関数は、ヘッダー<omp.h> に組み込まれます。これらは、並列実行環境を制御および照会するために使用できる*実行環境関数*、およびデータへのアクセスを同期化するために使用できる*ロック機能* を含んでいます。

XL C/C++ での OpenMP ランタイム・ライブラリー関数

ランタイム・ライブラリー関数名	ランタイム・ライブラリー関数の説明
omp_set_num_threads	これに続く並列領域で使用するために、スレッド数を設定します。
omp_get_num_threads	呼び出される並列領域を実行しているチーム内で現在実行しているスレッド数を戻します。
omp_get_max_threads	omp_get_num_threads への呼び出しによって戻すことができる最大値を戻します。
omp_get_thread_num	チーム内の関数を実行しているスレッドのスレッド番号を戻します。チームのマスター・スレッドはスレッド 0 です。
omp_get_num_procs	プログラムに割り当てることのできるプロセッサーの最大数を戻します。
omp_in_parallel	並列で実行している並列領域の動的エクステント内で呼び出された場合に、ゼロ以外を戻します。それ以外の場合は 0 を戻します。
omp_set_dynamic	並列領域の実行で使用可能なスレッド数の動的調整を使用可能または使用不可にします。
omp_get_dynamic	動的スレッド調整が使用可能な場合にゼロ以外を戻し、それ以外の場合は 0 を戻します。
omp_set_nested	ネストされた並列性を使用可能または使用不可にします。
omp_get_nested	ネストされた並列性が使用可能な場合にゼロ以外を戻し、使用不可の場合に 0 を戻します。
omp_init_lock	単純ロックを初期設定します。
omp_destroy_lock	単純ロックを除去します。
omp_set_lock	単純ロックが使用可能になるのを待ちます。
omp_unset_lock	単純ロックをリリースします。
omp_test_lock	単純ロックをテストします。
omp_init_nest_lock	ネストできるロックを初期設定します。
omp_destroy_nest_lock	ネストできるロックを除去します。
omp_set_nest_lock	ネストできるロックが使用可能になるのを待ちます。
omp_unset_nest_lock	ネストできるロックをリリースします。
omp_test_nest_lock	ネストできるロックをテストします。

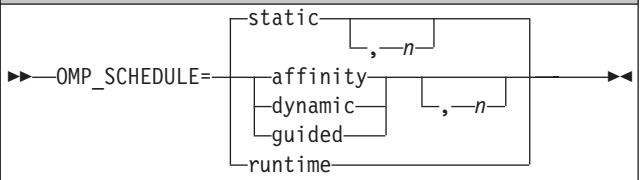
XL C/C++ での OpenMP ランタイム・ライブラリー関数

ランタイム・ライブラリー関数名	ランタイム・ライブラリー関数の説明
omp_get_wtick	連続する時計の 1 刻み間の秒数を戻します。
omp_get_wtime	経過した時刻を秒単位で戻します。

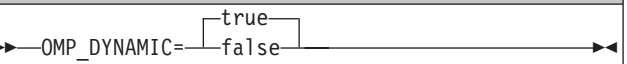
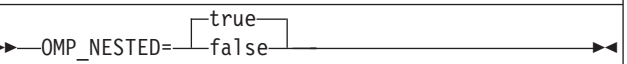
OpenMP 環境変数

OpenMP 環境変数は並列コードの実行を制御します。環境変数の名前は、常に大文字でなければなりませんが、それらの値は大/小文字の区別はありません。

XL C/C++ での OpenMP 環境変数

説明	構文
OMP_SCHEDULE ランタイム・スケジュール・タイプとチャンク・サイズを設定します。 `runtime` にスケジューリング・タイプ・セットを持つ OpenMP ディレクティブにのみ適用します。	 ここで、 **親和性** C にのみ有効な IBM 拡張。最初にループの反復がスレッド数によって分割された反復数の上限に等しいサイズのローカル区画に分割されることを指定します `CEILING(number_of_iterations ÷ number_of_threads)`。各ローカル区画はさらにローカル区画に残る反復数の半分の上限に等しいサイズのチャンクに細分されます `CEILING(iterations_left_in_local_partition ÷ 2)`。スレッドがフリーになると、そのローカル区画から次のチャンクを取ります。ローカル区画にチャンクがなくなると、スレッドは他のスレッドの区画から使用可能なチャンクを取ります。 *n* が指定されると、各ローカル区画はサイズ *n* のチャンクに細分されます。 *n* が指定されない場合は、デフォルト値は 1 です。 **動的** **for** ループの反復を一連の *n* サイズのチャンクに分ける必要があり、チャンクは次のプロセスに沿って処理されることを指定します。割り当てを待つスレッドに反復のチャンクが割り当てられると、スレッドはそのチャンクを実行し、次の割り当てを待ちます。このプロセスはすべてのチャンクが割り当てられるまで繰り返されます。*n* が指定されない場合は、デフォルトのチャンク・サイズは 1 です。 **ガイド** **for** ループの反復がサイズが減少しているチャンクのスレッドに割り当てられる必要があり、チャンクは次のプロセスに沿って処理されることを指定します。割り当てられた反復のチャンクを終了するスレッドには、動的に他のチャンクが割り当てられます。これは、すべてのチャンクが割り当てられるまで続きます。 *n* が指定されない場合は、最初のチャンク・サイズのデフォルト値は 1 です。 **静的** **for** ループの反復を一連の *n* サイズのチャンクに分割する必要があり、チャンクは次のプロセスに沿って処理されることを指定します。使用可能なスレッドは、スレッド数によって決定された順序でチャンクを割り当てます。 *n* が指定されない場合、反復スペースはサイズがほぼ等しいチャンクに分割され、1 つのチャンクが各スレッドに割り当てられます。 *n* 正数で、チャンク・サイズを示します。

XL C/C++ での OpenMP 環境変数

説明	構文
OMP_DYNAMIC 並列領域の実行で使用可能なスレッド数の動的調整を使用可能または使用不可にします。	 ここで、 **true** 使用可能なスレッド数の動的調整を使用可能にします。 **false** 使用可能なスレッド数の動的調整を使用不可にします。
OMP_NUM_THREADS 実行可能なスレッド数のセット。	 ここで、 *n* スレッド数を示します。
OMP_NESTED ネストされた並列性を使用可能または使用不可にします。	 ここで、 **true** ネストされた並列性を使用可能にします。 **false** ネストされた並列性を使用不可にします。

OpenMP インプリメンテーション定義動作

以下の情報は標準では指定されていません。標準のそれぞれのインプリメンテーションは、独自のインプリメンテーション定義値があります。

条件付きコンパイル

_OPENMP マクロは 199810 に定義されました。

スケジューリング

schedule 文節は、**for** ループの反復がどのようにチーム内のスレッド間で分割されるかを指定します。可能な OpenMP 標準値は、static、dynamic、guided、および runtime です。さらに、IBM C は値 affinity を拡張子として追加します。明示的に定義された schedule 文節がない場合は、XL C/C++ のデフォルトのスケジュールは static です。

特記事項

本書は米国 IBM が提供する製品およびサービスについて作成したものであり、本書に記載の製品、サービス、または機能が日本においては提供されていない場合があります。日本で利用可能な製品、サービス、および機能については、日本 IBM の営業担当員にお尋ねください。本書で IBM 製品、プログラム、またはサービスに言及していても、その IBM 製品、プログラム、またはサービスのみが使用可能であることを意味するものではありません。これらに代えて、IBM の知的所有権を侵害することのない、機能的に同等の製品、プログラム、またはサービスを使用することができます。ただし、IBM 以外の製品とプログラムの操作またはサービスの評価および検証は、お客様の責任で行っていただきます。

IBM は、本書に記載されている内容に関して特許権 (特許出願中のものを含む) を保有している場合があります。本書の提供は、お客様にこれらの特許権について実施権を許諾することを意味するものではありません。実施権についてのお問い合わせは、書面にて下記宛先にお送りください。

〒106-0032
東京都港区六本木 3-2-31
IBM World Trade Asia Corporation
Licensing

以下の保証は、国または地域の法律に沿わない場合は、適用されません。 IBM およびその直接または間接の子会社は、本書を特定物として現存するままの状態で提供し、商品性の保証、特定目的適合性の保証および法律上の瑕疵担保責任を含むすべての明示もしくは黙示の保証責任を負わないものとします。国または地域によっては、法律の強行規定により、保証責任の制限が禁じられる場合、強行規定の制限を受けるものとします。

この情報には、技術的に不適切な記述や誤植を含む場合があります。本書は定期的に見直され、必要な変更は本書の次版に組み込まれます。 IBM は予告なしに、随時、この文書に記載されている製品またはプログラムに対して、改良または変更を行うことがあります。

本書において IBM 以外の Web サイトに言及している場合がありますが、便宜のため記載しただけであり、決してそれらの Web サイトを推奨するものではありません。それらの Web サイトにある資料は、この IBM 製品の資料の一部ではありません。それらの Web サイトは、お客様の責任でご使用ください。

IBM は、お客様が提供するいかなる情報も、お客様に対してなんら義務も負うことのない、自ら適切と信ずる方法で、使用もしくは配布することができるものとします。

本プログラムのライセンス保持者で、(i) 独自に作成したプログラムとその他のプログラム (本プログラムを含む) との間での情報交換、および (ii) 交換された情報の相互利用を可能にすることを目的として、本プログラムに関する情報を必要とする方は、下記に連絡してください。

Lab Director
IBM Canada Ltd. Laboratory
B3/KB7/8200/MKM
8200 Warden Avenue
Markham, Ontario, Canada L6G 1C7

本プログラムに関する上記の情報は、適切な使用条件の下で使用することができますが、有償の場合もあります。

本書で説明されているライセンス・プログラムまたはその他のライセンス資料は、IBM 所定のプログラム契約の契約条項、IBM プログラムのご使用条件、またはそれと同等の条項に基づいて、 IBM より提供されます。

IBM 以外の製品に関する情報は、その製品の供給者、出版物、もしくはその他の公に利用可能なソースから入手したものです。IBM は、それらの製品のテストは行っておりません。したがって、他社製品に関する実行性、互換性、またはその他の要求については確証できません。 IBM 以外の製品の性能に関する質問は、それらの製品の供給者にお願いします。

本書には、日常の業務処理で用いられるデータや報告書の例が含まれています。より具体性を与えるために、それらの例には、個人、企業、ブランド、あるいは製品などの名前が含まれている場合があります。これらの名称はすべて架空のものであり、名称や住所が類似する企業が実在しているとしても、それは偶然にすぎません。

著作権使用許諾:

本書には、様々なオペレーティング・プラットフォームでのプログラミング手法を例示するサンプル・アプリケーション・プログラムがソース言語で掲載されています。お客様は、サンプル・プログラムが書かれているオペレーティング・プラットフォームのアプリケーション・プログラミング・インターフェースに準拠したアプリケーション・プログラムの開発、使用、販売、配布を目的として、いかなる形式においても、IBM に対価を支払うことなくこれを複製し、改変し、配布することができます。このサンプル・プログラムは、あらゆる条件下における完全なテストを経ていません。従って IBM は、これらのサンプル・プログラムについて信頼性、利便性もしくは機能性があることをほのめかしたり、保証することはできません。お客様は、IBM のアプリケーション・プログラミング・インターフェースに準拠したアプリケーション・プログラムの開発、使用、販売、配布を目的として、いかなる形式においても、IBM に対価を支払うことなくこれを複製し、改変し、配布することができます。

それぞれの複製物、サンプル・プログラムのいかなる部分、またはすべての派生的創作物にも、次のように、著作権表示を入れていただく必要があります。

© (お客様の会社名) (西暦年). このコードの一部は、IBM Corp. のサンプル・プログラムから取られています。 © Copyright IBM Corp. 1998, 2005 年. All rights reserved.

この情報をソフトコピーでご覧になっている場合は、写真やカラーの図表は表示されない場合があります。

プログラミング・インターフェース情報

プログラミング・インターフェース情報は、プログラムを使用してアプリケーション・ソフトウェアを作成する際に役立ちます。

一般使用プログラミング・インターフェースにより、お客様はこのプログラム・ ツール・サービスを含むアプリケーション・ソフトウェアを書くことができます。

ただし、この情報には、診断、修正、および調整情報が含まれている場合があります。診断、修正、調整情報は、お客様のアプリケーション・ソフトウェアのデバッグ支援のために提供されています。

警告: 診断、修正、調整情報は、変更される場合がありますので、プログラミング・インターフェースとしては使用しないでください。

商標

以下は、IBM Corporation の商標です。

AIX	PowerPC
@server	PowerPC ArchitecturepSeries
IBM	Redbooks
POWER3	VisualAge
POWER4	
POWER5	

UNIX は、The Open Group の米国およびその他の国における登録商標です。

Linux は、Linus Torvalds の米国およびその他の国における商標です。

他の会社名、製品名およびサービス名等はそれぞれ各社の商標です。

業界標準

以下の規格がサポートされます。

- C 言語は、International Standard C (ANSI/ISO-IEC 9899-1990 [1992]) に準拠しています。この規格は、American National Standard for Information Systems-Programming Language C (X3.159-1989) を正式に置き換えたもので、ANSI C 規格と技術的に同等です。コンパイラーは、ISO/IEC 9899:1990/Amendment 1:1994 によって C 標準に採用された変更をサポートしています。
- C 言語は、International Standard for Information Systems-Programming Language C (ISO/IEC 9899-1999 (E)) に準拠しています。
- C++ 言語は、言語の最初の公式定義である、International Standard for Information Systems-Programming Language C++ (ISO/IEC 14882:1998) に準拠しています。
- C++ 言語は、現在 *Standard C++* とも呼ばれている、 International Standard for Information Systems-Programming Language C++ (ISO/IEC 14882:2003 (E)) にも準拠しています。

- C および C++ コンパイラーは、OpenMP C and C++ Application Programming Interface Version 2.0 をサポートしています。

プログラム番号: 5724-K77

SC88-9977-01

日本アイ・ビー・エム株式会社
〒106-8711　東京都港区六本木3-2-12